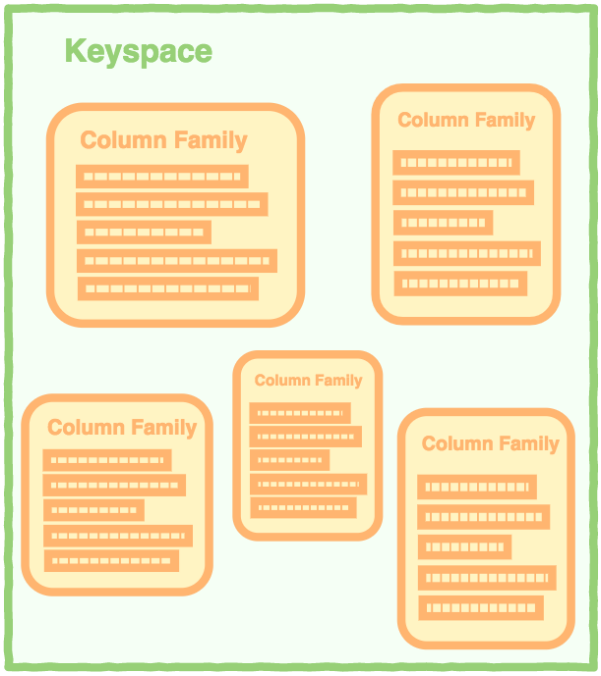
Unit - 3

Column-Oriented Databases:

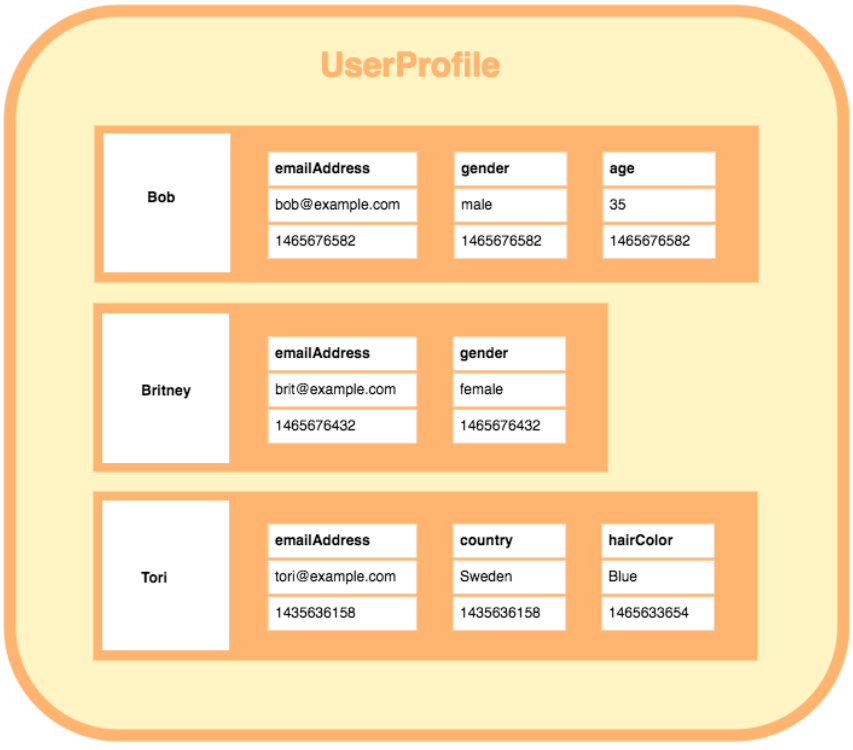
What Is a Column-Family Data Store, Cassandra Database: What is Cassandra, Cassandra Architecture, Cassandra Data types, Cassandra Query Language-CQL, Creating, Altering, Dropping a KeySpace, Cassandra CRUD Operations, Suitable Use Cases, and When Not to Use.

❖ What is a column family model?

- ➢ A column store database is a type of database that stores data using a column oriented model.
- ➢ Columns store databases use a concept called a keyspace. A keyspace is kind of like a schema in the relational model. The keyspace contains all the column families (kind of like tables in the relational model), which contain rows, which contain columns.



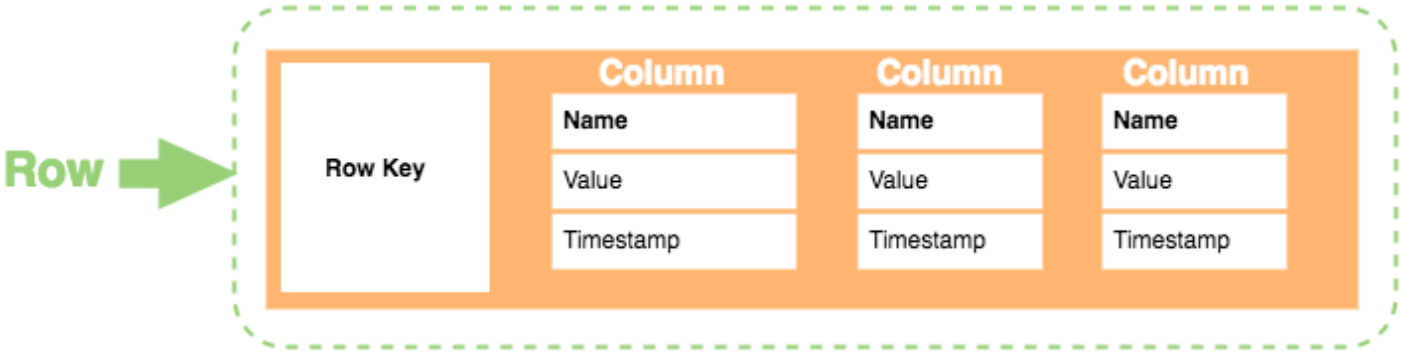
- ➢ For example



As the above diagram shows:

- A column family consists of multiple rows.
- Each row can contain a different number of columns to the other rows. And the columns don't have to match the columns in the other rows (i.e. they can have different column names, data types, etc).
- Each column is contained to its row. It doesn't span all rows like in a relational database. Each column contains a name/value pair, along with a timestamp. Note that this example uses Unix/Epoch time for the timestamp.

Here's how each row is constructed:



Here's a breakdown of each element in the row:

- Row Key. Each row has a unique key, which is a unique identifier for that row.
- Column. Each column contains a name, a value, and timestamp.
- Name. This is the name of the name/value pair.
- Value. This is the value of the name/value pair.
- Timestamp. This provides the date and time that the data was inserted. This can be used to determine the most recent version of data.

➢ Some DBMSs expand on the column family concept to provide extra functionality/storage ability. For example, Cassandra has the concept of *composite columns*, which allow you to nest objects inside a column.
Some key benefits of columnar databases include:

- Compression. Column stores are very efficient at data compression and/or partitioning.
- Aggregation queries. Due to their structure, columnar databases perform particularly well with aggregation queries (such as SUM, COUNT, AVG, etc).
- Scalability. Columnar databases are very scalable. They are well suited to massively parallel processing (MPP), which involves having data spread across a large cluster of machines – often thousands of machines.
- Fast to load and query. Columnar stores can be loaded extremely fast. A billion row table could be loaded within a few seconds. You can start querying and analysing almost immediately.

❖ Cassandra Database: What is Cassandra

➢ Apache Cassandra is a highly scalable, high-performance distributed database designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.

➢ Apache Cassandra is an open source, distributed and decentralized/distributed storage system (database), for managing very large amounts of structured data spread out across the world. It provides highly available service with no single point of failure.

➢ Listed below are some of the notable points of Apache Cassandra −

- It is scalable, fault-tolerant, and consistent.
- It is a column-oriented database.
- Its distribution design is based on Amazon's Dynamo and its data model on Google's Bigtable.
- Created at Facebook, it differs sharply from relational database management systems.
- Cassandra implements 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 such as Facebook, Twitter, Cisco, Rackspace, ebay, Twitter, Netflix, and more.

➢ Features of Cassandra

➢ Cassandra has become so popular because of its outstanding technical features. Given below are some of the features of Cassandra:

- Elastic scalability − Cassandra is highly scalable; it allows to add more hardware to accommodate more customers and more data as per requirement.
- Always on architecture − Cassandra has no 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, i.e., it increases your throughput as you increase the number of nodes in the cluster. Therefore it maintains a quick response time.
- Flexible data storage − Cassandra accommodates all possible data formats including: structured, semi-structured, and unstructured. It can dynamically accommodate changes to your data structures according to your need.
- Easy data distribution − Cassandra provides the flexibility to distribute data where you need by replicating data across multiple data centers.
- Transaction support − Cassandra supports properties like Atomicity, Consistency, Isolation, and Durability (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.

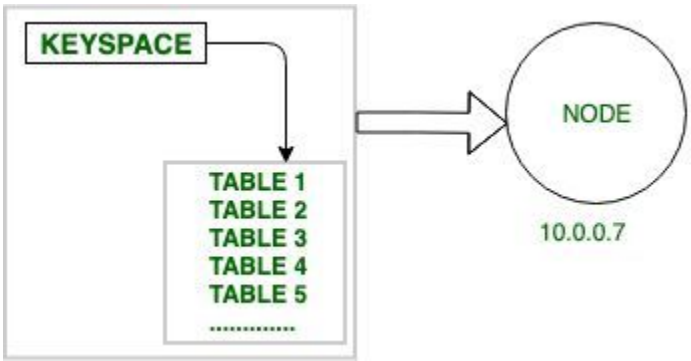
❖ Cassandra Architecture

➢ The following are the components of Cassandra

- Basic Terminology:
  - Node
  - Data center
  - Cluster
- Operations:
  - Read Operation
  - Write Operation
- Storage Engine
  - CommitLog
  - Memtables
  - SSTables
- Data Replication Strategies

➢ Node

Node is the basic component in Apache Cassandra. It is the place where actually data is stored. For Example:As shown in diagram node which has IP address 10.0.0.7 contain data (keyspace which contain one or more tables).

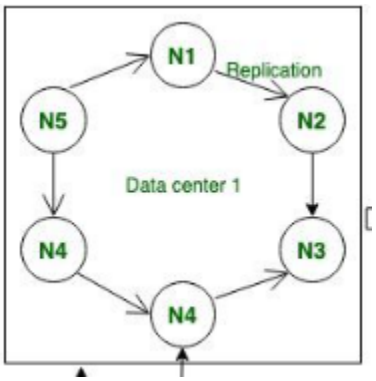


➢ Data Center:

Data Center is a collection of nodes.
For example:

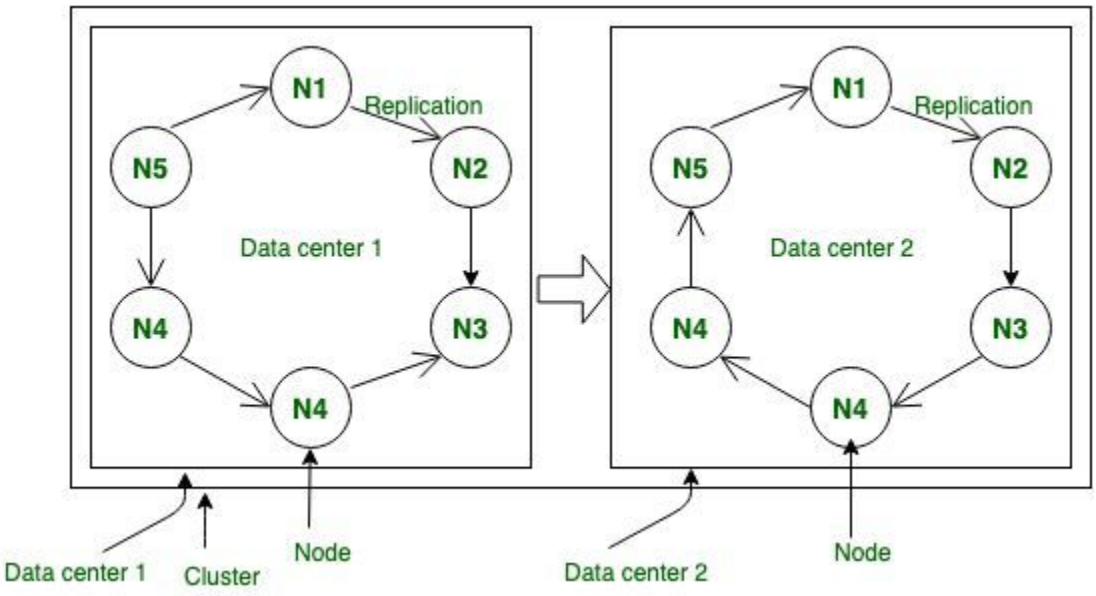
DC – N1 + N2 + N3 ….
DC: Data Center
N1: Node 1
N2: Node 2

N3: Node 3



- Cluster

  It is the collection of many data centers.
  For example:
  C = DC1 + DC2 + DC3….
  C: Cluster
  DC1: Data Center 1
  DC2: Data Center 2
  DC3: Data Center 3



- Read Operation

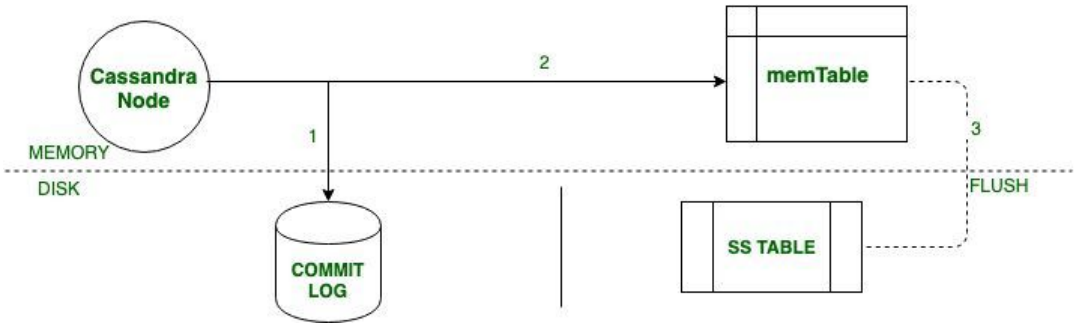
  In Read Operation there are three types of read requests that a coordinator can send to a replica. The node that accepts the write requests called coordinator for that particular operation.

  Direct Request:
  In this operation coordinator node sends the read request to one of the replicas.
  Digest Request:
  In this operation coordinator will contact to replicas specified by the consistency level. For Example: CONSISTENCY TWO; It simply means that Any two nodes in data center will acknowledge.
  Read Repair Request:
  If there is any case in which data is not consistent across the node then background Read Repair Request initiated that makes sure that the most recent data is available across the nodes.

- Write Operation
  - Step-1:

    In Write Operation as soon as we receives request then it is first dumped into commit log to make sure that data is saved.

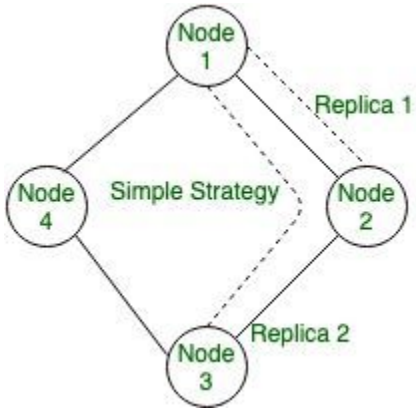
- Step-2:
  Insertion of data into table that is also written in MemTable that holds the data till it's get full.
- Step-3:
  If MemTable reaches its threshold then data is flushed to SS Table. .



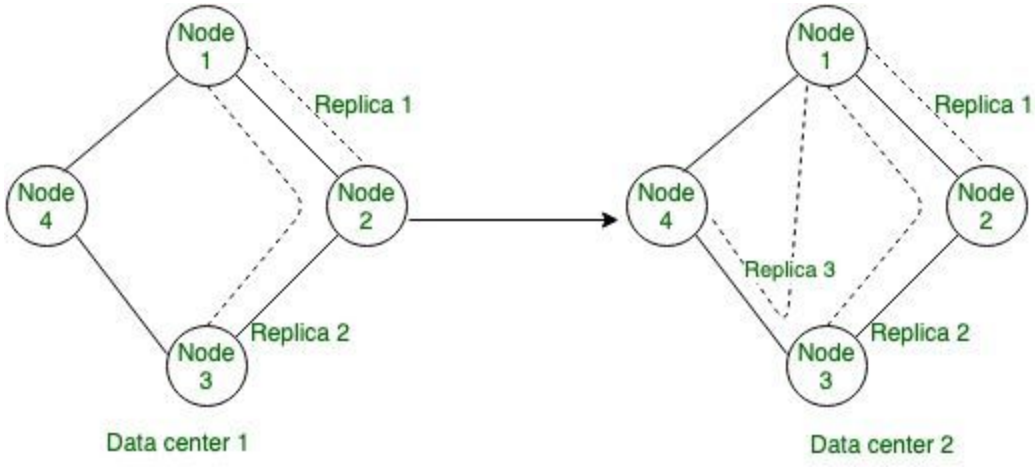
➢ Storage Engine
- Commit log:
  Commit log is the first entry point while writing to disk or memTable. The purpose of commit log in apache Cassandra is to server sync issues if a data node is down.
- Mem-table:
  After data written in Commit log then after that data is written in Mem-table. Data is written in Mem-table temporarily.
- SSTable:
  Once Mem-table will reach a certain threshold then data will flushed to the SSTable disk file.

➢ Data Replication Strategy:
  Basically it is used for backup to ensure no single point of failure. In this strategy Cassandra uses replication to achieve high availability and durability. Each data item is replicated at N hosts, where N is the replication factor configured \per-instance".

  There are two type of replication Strategy: Simple Strategy, and Network Topology Strategy. These are explained as following below.
- Simple Strategy:
  In this Strategy it allows a single integer RF (replication_factor) to be defined. It determines the number of nodes that should contain a copy of each row. For example, if replication_factor is 2, then two different nodes should store a copy of each row.



- Network Topology Strategy:
  In this strategy it allows a replication factor to be specified for each datacenter in the cluster. Even if your cluster only uses a single datacenter. This Strategy should be preferred over SimpleStrategy to make it easier to add new physical or virtual datacenters to the cluster later.



❖ CQL :

Cassandra CQL, a simple alternative to Structured Query Language (SQL), is a declarative language developed to provide abstraction in accessing Apache Cassandra.

Basic Cassandra (CQL) constructs include:

- **Keyspace** — Similar to an RDBMS database, a keyspace is a container for application data that must have a name and a set of associated attributes. Cassandra keyspace is a SQL database.
- **Column Families/Tables** — A keyspace consists of a number of Column Families/Tables. A Cassandra column family is a SQL table.
- **Primary Key / Tables** — A Primary Key consists of a Row/Partition Key and a Cluster Key, and functions to enable users to uniquely identify internal rows of data. A Row/Partition Key determines the node on which data is stored. A Cluster Key determines the sort order of data within a particular row.

❖ Creating a Keyspace

➢ A keyspace in Cassandra is a namespace that defines data replication on nodes. A cluster contains one keyspace per node. Given below is the syntax for creating a keyspace using the statement CREATE KEYSPACE.

Syntax
CREATE KEYSPACE <identifier> WITH <properties>

CREATE KEYSPACE “KeySpace Name”
WITH replication = {'class': ‘Strategy name’, 'replication_factor' : ‘No.Of replicas’};

➢ Example
  - CREATE KEYSPACE vitb
    WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 2};

➢ We can enter into a keyspace with the following syntax
Use <keyspace name>
Ex:
Use vitb

❖ Altering a Keyspace

➢ ALTER KEYSPACE can be used to alter properties such as the number of replicas and the durable_writes of a KeySpace. Given below is the syntax of this command.

Syntax
ALTER KEYSPACE <identifier> WITH <properties>

ALTER KEYSPACE “KeySpace Name”
WITH replication = {'class': ‘Strategy name’, 'replication_factor' : ‘No.Of replicas’};

- Example:
  - ALTER KEYSPACE vitb
    WITH replication = {'class':'NetworkTopologyStrategy', 'replication_factor' : 3};

❖ Dropping a keyspace

- A KeySpace can be dropped using the command DROP KEYSPACE. Given below is the syntax for dropping a KeySpace.

Syntax

- DROP KEYSPACE <identifier>
  DROP KEYSPACE “KeySpace name”

- Example
  DROP KEYSPACE vitb;

❖ Cassandra CRUD operations on table(also called as column family)

- CRUD stands for create, read, update, delete
- Create
  - You can create a table using the command CREATE TABLE. Given below is the syntax for creating a table.

    Syntax
    CREATE (TABLE | COLUMNFAMILY) <tablename>
    ('<column-definition>' , '<column-definition>')
    (WITH <option> AND <option>)
  - Example

```
CREATE TABLE emp(
  emp_id int PRIMARY KEY,
  emp_name text,
  emp_city text,
  emp_sal varint,
  emp_phone varint
  );
```

- Read
  - SELECT clause is used to read data from a table in Cassandra. Using this clause, you can read a whole table, a single column, or a particular cell. Given below is the syntax of SELECT clause.

    SELECT FROM <tablename>
  - Example

```
Select * from emp;
Select * from emp where emp_id=123;
Select emp_sal from emp where emp_city=’Hyd’;
```

- Update
  - UPDATE is the command used to update data in a table −

    Where − This clause is used to select the row to be updated.
    Set − Set the value using this keyword.
    Must − Includes all the columns composing the primary key.

  - While updating rows, if a given row is unavailable, then UPDATE creates a fresh row. Given below is the syntax of UPDATE command −

    UPDATE <tablename>
    SET <column name> = <new value>

<column name> = <value>....
WHERE <condition>

- Example
  - UPDATE emp SET emp_city='Delhi',emp_sal=50000 WHERE emp_id=2;

➢ Delete

- You can delete data from a table using the command DELETE. Its syntax is as follows −

  DELETE FROM <identifier> WHERE <condition>;
- Example
  - DELETE emp_sal FROM emp WHERE emp_id=3;

❖ Suitable use cases of column-family databases

➢ Event Logging

- Column-family databases with their ability to store any data structures are a great choice to store event information, such as application state or errors encountered by the application. Within the enterprise, all applications can write their events to Cassandra with their own columns and the rowkey of the form appname:timestamp. Since we can scale writes, Cassandra would work ideally for an event logging system

➢ Content Management Systems, Blogging Platforms

- Using column families, you can store blog entries with tags, categories, links, and trackbacks in different columns. Comments can be either stored in the same row or moved to a different keyspace; similarly, blog users and the actual blogs can be put into different column families

➢ Counters

- Often, in web applications you need to count and categorize visitors of a page to calculate analytics. You can use the CounterColumnType during creation of a column family.

➢ Expiring Usage

- You may provide demo access to users, or may want to show ad banners on a website for a specific time. You can do this by using expiring columns: Cassandra allows you to have columns which, after a given time, are deleted automatically. This time is known as TTL (Time To Live) and is defined in seconds. The column is deleted after the TTL has elapsed; when the column does not exist, the access can be revoked or the banner can be removed.

❖ When not to use

➢ ACID transactions

- There are problems for which column-family databases are not the best solutions, such as systems that require ACID transactions for writes and reads. If you need the database to aggregate the data using queries (such as SUM or AVG), you have to do this on the client side using data retrieved by the client from all the rows.

➢ Early prototypes

- Cassandra is not great for early prototypes or initial tech spikes: During the early stages, we are not sure how the query patterns may change, and as the query patterns change, we have to change the column family design. This causes friction for the product innovation team and slows down developer productivity. RDBMS impose high cost on schema change, which is traded off for a low cost of query change; in Cassandra, the cost may be higher for query change as compared to schema change.