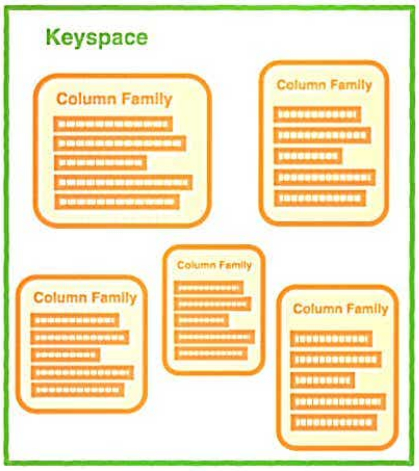
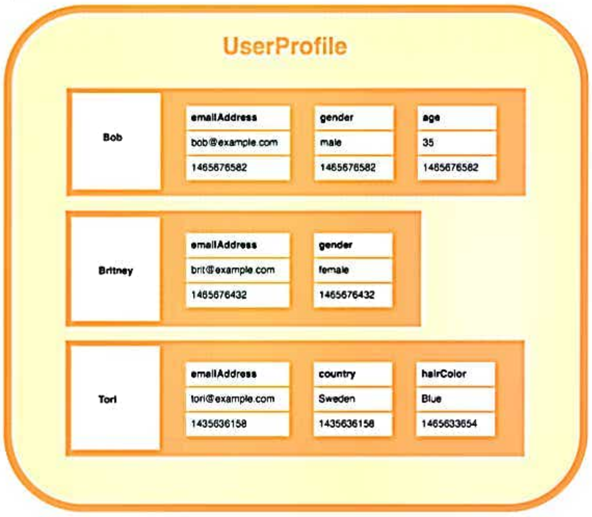
❖ **What is a column family model?**

➢ 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.



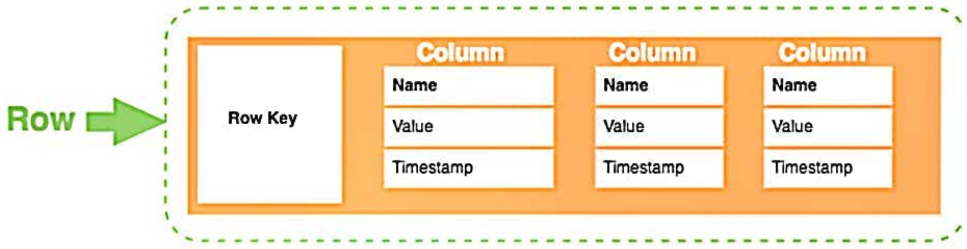
➢ For example



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.

2

CQL Type	Constants	Description
ascii	Strings	US-ascii character string
bigint	Integers	64-bit signed long
blob	blobs	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
timeuuid	uuids	Type 1 uuid
tuple		A group of 2,3 fields
uuid	uuids	Standard uuid
varchar	strings	UTF-8 encoded string
varint	Integers	Arbitrary precision integer

Document Data Model: 3

A Document Data Model is a lot different than other data models because it stores data in JSON, BSON, or XML documents. in this data model, we can move documents under one document and apart from this, any particular elements can be indexed to run queries faster. Often documents are stored and retrieved in such a way that it becomes close to the data objects which are used in many applications which means very less translations are required to use data in applications. JSON is a native language that is often used to store and query data too.

So in the document data model, each document has a key-value pair below is an example for the same.

```
{
"Name" : "Yashodhra",
"Address" : "Near Patel Nagar",
"Email" : "yahoo123@yahoo.com",
"Contact" : "12345"
}
```

ChatGPT

Advantages of Document Databases:

1. **Schema Flexibility:**
   - **Advantage:** Document databases, like MongoDB, offer schema flexibility, allowing documents within a collection to have varying structures. This accommodates dynamic and evolving data requirements without a predefined schema.
2. **Nested and Hierarchical Data:**
   - **Advantage:** Document databases support nested and hierarchical data structures, enabling the storage of complex relationships within a single document. This simplifies data retrieval and improves query performance for certain use cases.
3. **Natural Mapping to Objects:**
   - **Advantage:** Document databases often provide a natural mapping between the database documents and the objects used in the application code. This object-document mapping can streamline development and make the code more intuitive.
4. **Query Flexibility:**
   - **Advantage:** Document databases support rich query capabilities, including the ability to query based on the content of nested fields and arrays. This flexibility makes it easier to express complex queries and retrieve specific subsets of data.
5. **Horizontal Scalability:**
   - **Advantage:** Document databases are designed for horizontal scalability. By distributing documents across multiple nodes or servers, these databases can handle large amounts of data and high traffic loads.
6. **Aggregation Framework:**
   - **Advantage:** Many document databases offer powerful aggregation frameworks that allow for complex data transformations and analysis directly within the database. This can reduce the need for extensive data processing on the application side.

Limitations of Document Databases:

1. **Performance for Joins:**
   - **Limitation:** Document databases are generally not optimized for complex join operations across multiple collections. Performing joins may require multiple queries or denormalization of data, impacting performance.
2. **Storage Overhead:**
   - **Limitation:** The denormalized structure of document databases can result in some level of redundancy and storage overhead, especially when dealing with large amounts of duplicated data across documents.
3. **Learning Curve for Developers:**
   - **Limitation:** Developers accustomed to relational databases may face a learning curve when transitioning to document databases, as the data modeling and query paradigms can be different.
4. **Data Consistency Trade-offs:**
   - **Limitation:** Document databases may prioritize performance and availability over strict consistency. As a result, there might be scenarios where eventual consistency is acceptable, but immediate consistency is sacrificed.
5. **Not Ideal for Tabular Data:**
   - **Limitation:** Document databases are not the most efficient choice for scenarios where data is highly tabular and the relationships between tables are well-defined. In such cases, relational databases may be more suitable.
6. **Atomic Transactions Across Documents:**
   - **Limitation:** Enforcing atomic transactions that span multiple documents can be challenging in some document databases. Maintaining consistency in distributed transactions may require careful design considerations.
7. **Potential for Over-Normalization:**
   - **Limitation:** While document databases allow for denormalization, there's a risk of over-normalization in some cases, leading to unnecessary complexity and potentially impacting performance.

4

ChatGPT

Document databases are well-suited for various real-time use cases due to their flexibility, scalability, and ability to handle semi-structured or hierarchical data. Here are some real-time use cases where document databases excel:

1. **Content Management Systems (CMS):**
   - **Use Case:** Storing and managing content for websites or applications where the content structure may evolve over time. Document databases allow for flexible document structures, making it easy to store and retrieve diverse types of content.
2. **User Profiles and Personalization:**
   - **Use Case:** Storing user profiles with dynamic and personalized attributes. Document databases can efficiently handle user data that varies from user to user, allowing for quick retrieval and modification of user profiles.
3. **E-commerce Product Catalogs:**
   - **Use Case:** Managing product catalogs with varying attributes, specifications, and images. Document databases enable the storage of product information in a hierarchical and flexible format, facilitating efficient queries for product details.
4. **Real-time Analytics and Event Logging:**
   - **Use Case:** Capturing and analyzing real-time events, logs, or telemetry data. Document databases can handle time-series data and evolving event structures, making them suitable for real-time analytics and monitoring applications.
5. **IoT (Internet of Things) Data Storage:**
   - **Use Case:** Storing data generated by IoT devices, such as sensor readings, telemetry data, and device metadata. Document databases can accommodate the diverse and dynamic nature of IoT data with flexible schemas.
6. **Collaborative Applications and Chat Platforms:**
   - **Use Case:** Building collaborative applications or chat platforms where each conversation or thread may have a different structure. Document databases can store chat messages, user information, and multimedia content in a flexible and hierarchical format.
7. **Gaming Leaderboards and Achievements:**
   - **Use Case:** Managing gaming leaderboards, player profiles, and achievements. Document databases allow for the efficient storage and retrieval of player data, including scores, achievements, and game statistics.
8. **Dynamic Forms and Surveys:**
   - **Use Case:** Storing data from dynamic forms or surveys where the structure of the data may vary based on the form's design. Document databases provide flexibility in handling form responses with different sets of fields.

9. **Identity and Access Management:**
   - **Use Case:** Storing user authentication and authorization data, including user roles, permissions, and authentication tokens. Document databases can represent user profiles with varying access rights in a hierarchical structure.
10. **Inventory Management Systems:**
    - **Use Case:** Managing inventory for e-commerce or retail applications. Document databases can store product details, stock levels, and order history, providing a scalable solution for real-time inventory management.
11. **Location-based Services:**
    - **Use Case:** Storing and retrieving geospatial data, such as user locations, points of interest, or geographic information. Document databases can handle spatial data efficiently for applications that require real-time location-based services.
12. **Workflow and Task Management:**
    - **Use Case:** Storing and tracking workflow tasks, statuses, and associated metadata. Document databases allow for the representation of complex workflows with varying attributes for different types of tasks.

5

❖ 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"

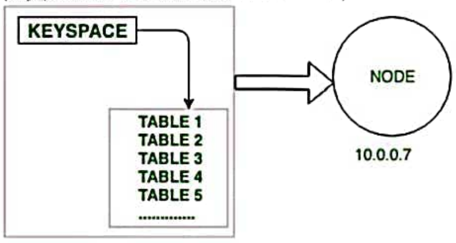
➢ Example
DROP KEYSPACE vitb;

6

❖ 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).

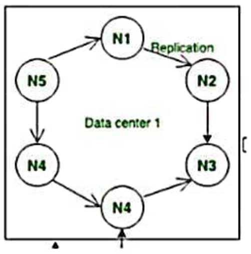


- Data Center:

  Data Center is a collection of nodes.
  For example:

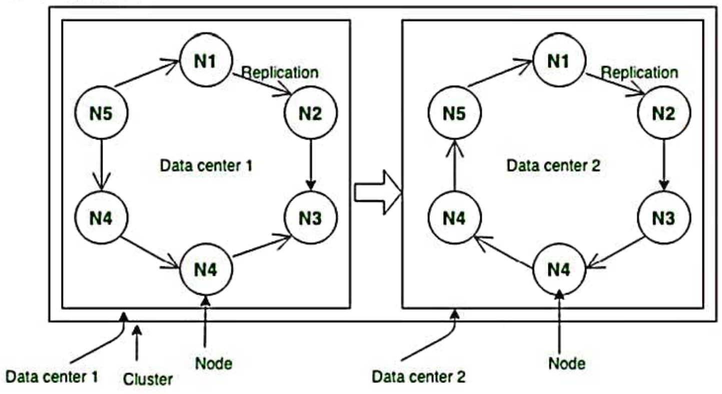
  DC – N1 + N2 + N3 ....
  DC: Data Center
  N1: Node 1
  N2: Node 2

  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

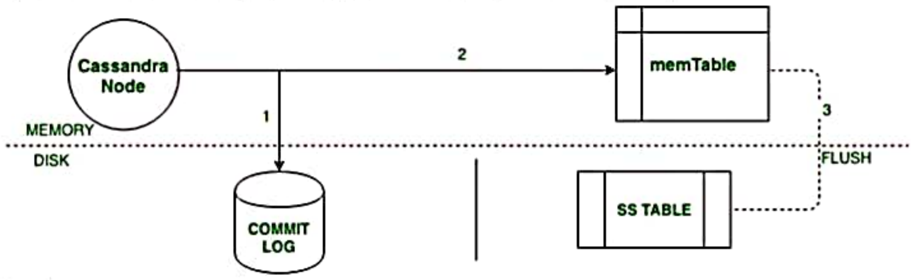


- Read Operation

  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. .



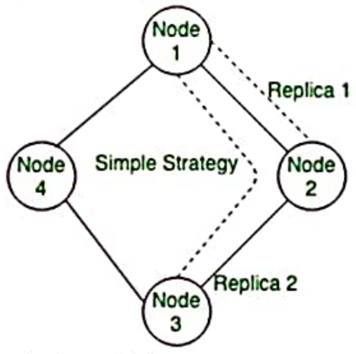
➢ Storage Engine
- Commit log:
  Commit log is the first entry point while writing to disk or memTable. The purpose of commit log in apache Cassandra is to server sync issues if a data node is down.
- Mem-table:
  After data written in Commit log then after that data is written in Mem-table. Data is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

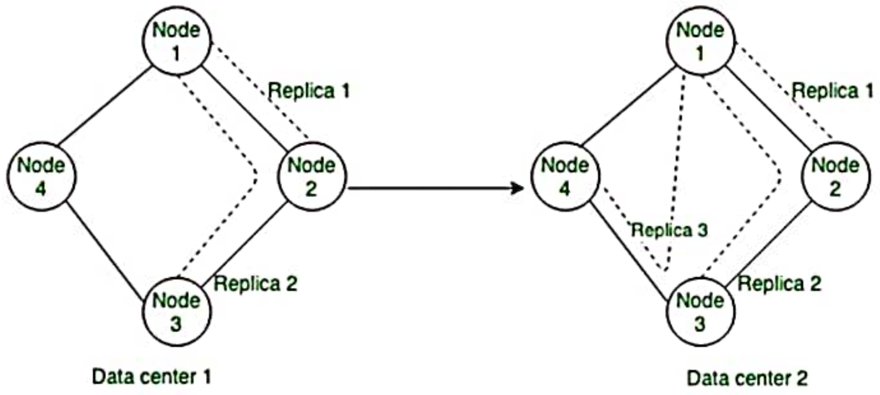
Basically it is used for backup to ensure no single point of failure. In this strategy Cassandra uses replication to achieve high availability and durability. Each data item is replicated at N hosts, where N is the replication factor configured \per-instance".

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 Query Language):

CQL, or Cassandra Query Language, is a query language for interacting with Apache Cassandra, a highly scalable and distributed NoSQL database. CQL is designed to resemble SQL in its syntax but is specifically tailored for working with Cassandra's data model and distributed architecture.

Cqlsh Commands:

1. `DESCRIBE`:
   - **Purpose:** Provides information about keyspaces, tables, and columns.
   - **Example:**

```cql
DESCRIBE KEYSPACES;
DESCRIBE TABLE mykeyspace.mytable;
```

2. `USE`:
   - **Purpose:** Switches to a specific keyspace, making it the default keyspace for subsequent queries.
   - **Example:**

```cql
USE mykeyspace;
```

3. `SELECT`:
   - **Purpose:** Retrieves data from one or more columns in a table.
   - **Example:**

```cql
SELECT column1, column2 FROM mytable WHERE condition;
```

4. `INSERT`:
   - **Purpose:** Adds new rows of data to a table.
   - **Example:**

```cql
INSERT INTO mytable (column1, column2) VALUES ('value1', 'value2');
```

5. `UPDATE`:
   - **Purpose:** Modifies existing data in a table.
   - **Example:**

```cql
UPDATE mytable SET column1 = 'new_value' WHERE condition;
```

6. `DELETE`:
   - **Purpose:** Removes rows of data from a table based on a specified condition.
   - **Example:**

```cql
DELETE FROM mytable WHERE condition;
```

7. `CREATE`:
   - **Purpose:** Creates a new keyspace, table, or index.
   - **Example:**

```cql
CREATE KEYSPACE mykeyspace WITH replication = {'class': 'SimpleStrat
CREATE TABLE mykeyspace.mytable (column1 TEXT PRIMARY KEY, column2 I
```

8. `ALTER`:
   - **Purpose:** Modifies the structure of a keyspace, table, or column family.
   - **Example:**

```cql
ALTER TABLE mykeyspace.mytable ADD column3 BOOLEAN;
```

9. `DROP`:
   - **Purpose:** Deletes a keyspace, table, or index.
   - **Example:**

```cql
DROP KEYSPACE mykeyspace;
DROP TABLE mykeyspace.mytable;
```

Explanation of Three Cqlsh Commands:

1. `DESCRIBE`:
   - **Purpose:** Used to obtain information about the Cassandra schema, including keyspaces, tables, and column families.
   - **Example:**

   ```cql
   DESCRIBE KEYSPACES;
   ```

   This command lists all the keyspaces available in the Cassandra database.
2. `USE`:
   - **Purpose:** Sets the current keyspace for the session, making it the default keyspace for subsequent queries.
   - **Example:**

   ```cql
   USE mykeyspace;
   ```

   This command switches the current keyspace to "mykeyspace," allowing subsequent queries to reference tables within this keyspace.
3. `SELECT`:
   - **Purpose:** Retrieves data from one or more columns in a table based on specified conditions.
   - **Example:**

   ```cql
   SELECT column1, column2 FROM mytable WHERE condition;
   ```

   This command retrieves the values of "column1" and "column2" from the table "mytable" based on a specified condition.

- 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';
```