Unit-2

1.What are Aggregate Data Models in NoSQL?

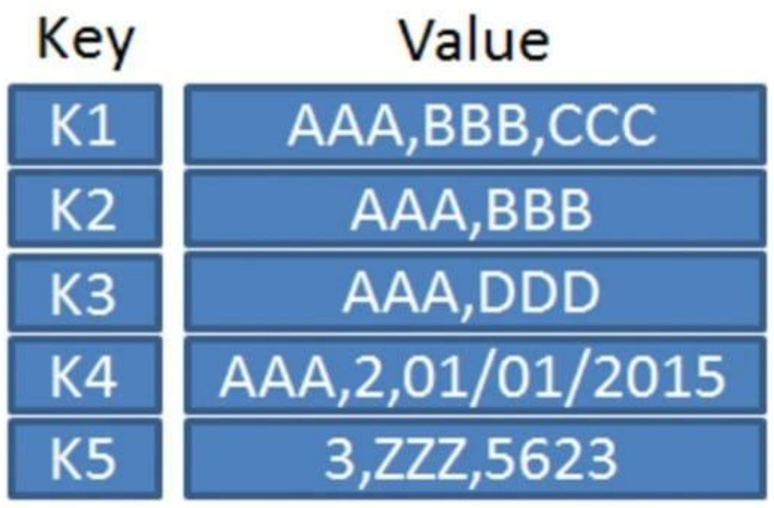
- Aggregate means a collection of objects that are treated as a unit. In NoSQL Databases, an aggregate is a collection of data that interact as a unit. Moreover, these units of data or aggregates of data form the boundaries for the ACID operations.
- Aggregate Data Models in NoSQL make it easier for the Databases to manage data storage over the clusters as the aggregate data or unit can now reside on any of the machines. Whenever data is retrieved from the Database all the data comes along with the Aggregate Data Models in NoSQL.
- Aggregate Data Models in NoSQL don't support ACID transactions and sacrifice one of the ACID properties. With the help of Aggregate Data Models in NoSQL, you can easily perform OLAP operations on the Database.
- You can achieve high efficiency of the Aggregate Data Models in the NoSQL Database if the data transactions and interactions take place within the same aggregate.

Types of Aggregate Data Models in NoSQL Databases

The Aggregate Data Models in NoSQL are majorly classified into 4 Data Models listed below:

- Key-Value Model
- Document Model
- Column Family Model
- Graph-Based Model

Key-Value Model



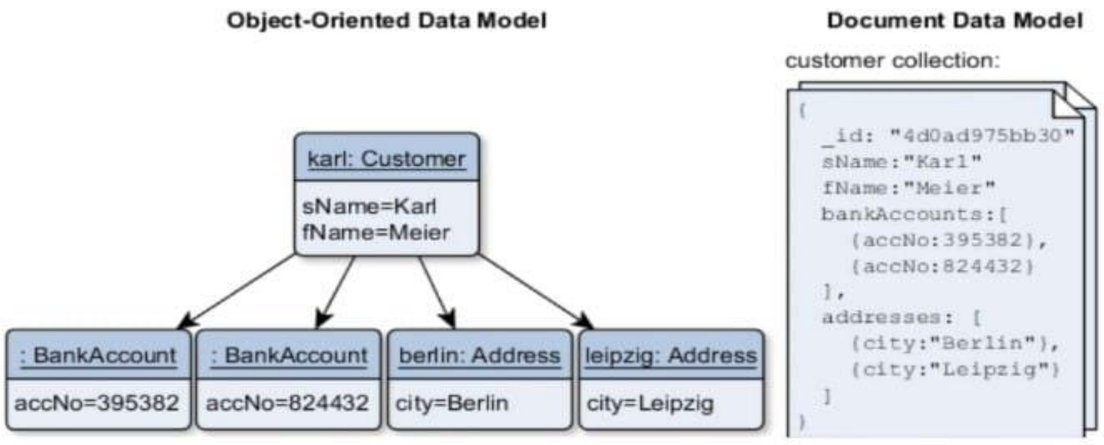
Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

- The Key-Value Data Model contains the key or an ID used to access or fetch the data of the aggregates corresponding to the key. In this Aggregate Data Models in NoSQL, the data of the aggregates are secure and encrypted and can be decrypted with a Key.

Use Cases:

- These Aggregate Data Models in NoSQL Database are used for storing the user session data.
- Key Value-based Data Models are used for maintaining schema-less user profiles.
- It is used for storing user preferences and shopping cart data.

Document Model

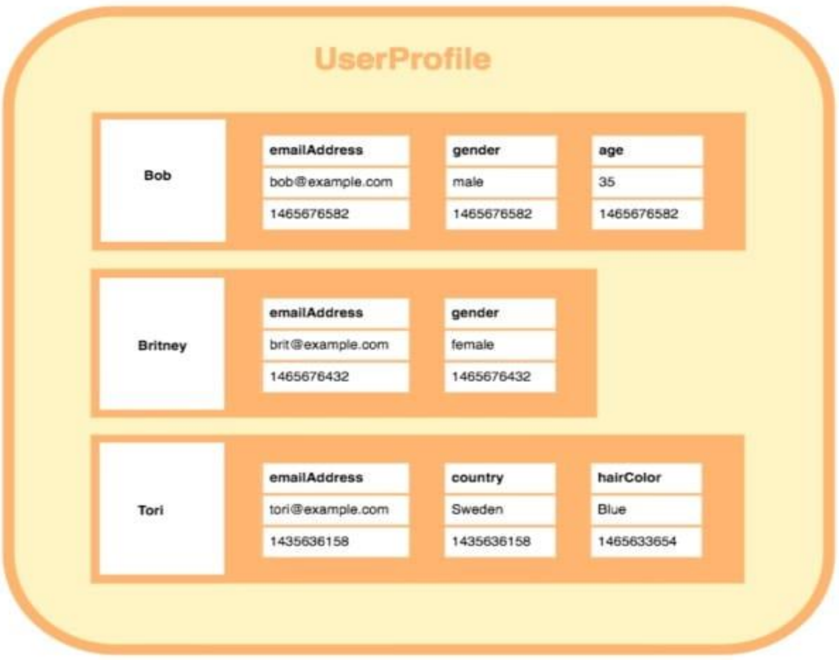


- The Document Data Model allows access to the parts of aggregates. In this Aggregate Data Models in NoSQL, the data can be accessed in an inflexible way. The Database stores and retrieves documents, which can be XML, JSON, BSON, etc. There are some restrictions on data structure and data types of the data aggregates that are to be used in this Aggregate Data Models in NoSQL Database.

Use Cases:

- Document Data Models are widely used in E-Commerce platforms
- It is used for storing data from content management systems.
- Document Data Models are well suited for Blogging and Analytics platforms.

Column Family Model



Column family is an Aggregate Data Models in NoSQL Database usually with big-table style Data Models that are referred to as column stores. It is also called a two-level map as it offers a two-level aggregate structure. In this Aggregate Data Models in NoSQL, the first level of the Column family contains the keys that act as a row identifier that is used to select the aggregate data. Whereas the second level values are referred to as columns.

Use Cases:

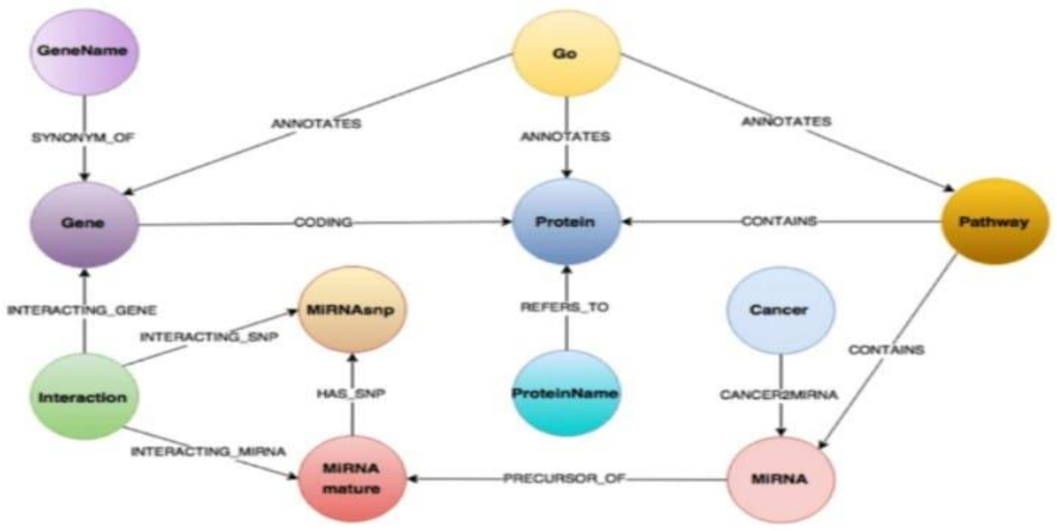
- Column Family Data Models are used in systems that maintain counters.
- These Aggregate Data Models in NoSQL are used for services that have expiring usage.
- It is used in systems that have heavy write requests.

Graph-Based Model

Graph-based data models store data in nodes that are connected by edges. These Aggregate Data Models in NoSQL are widely used for storing the huge volumes of complex aggregates and multidimensional data having many interconnections between them.

Use Cases:

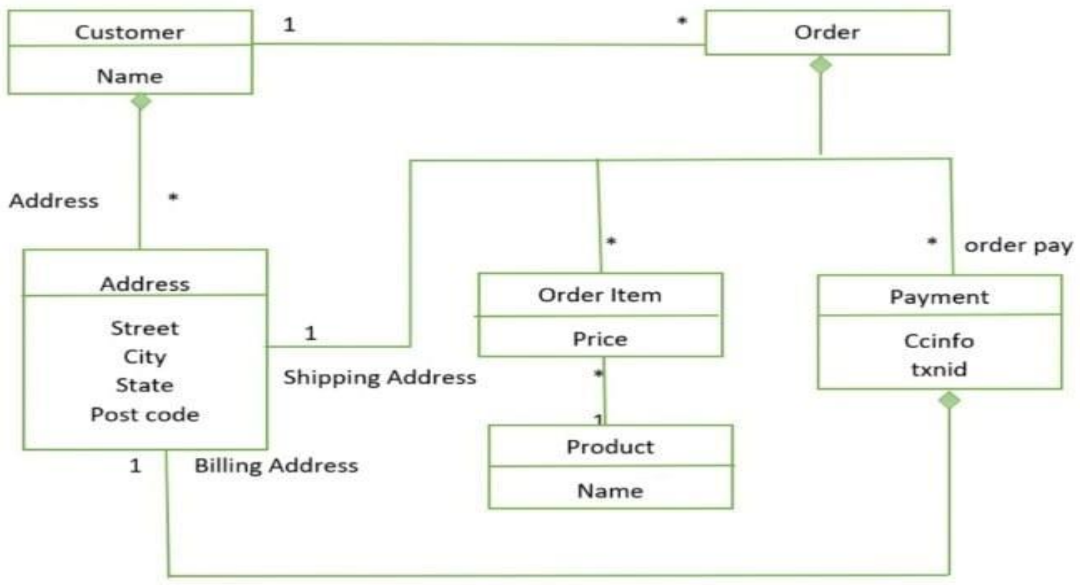
- Graph-based Data Models are used in social networking sites to store interconnections.
- It is used in fraud detection systems.
- This Data Model is also widely used in Networks and IT operations.



Steps to Build Aggregate Data Models in NoSQL Databases

Now that you have a brief knowledge of Aggregate Data Models in NoSQL Database. In this section, you will go through an example to understand how to design Aggregate Data Models in NoSQL. For this, a Data Model of an E-Commerce website will be used to explain Aggregate Data Models in NoSQL.

This example of the E-Commerce Data Model has two main aggregates – customer and order. The customer contains data related to billing addresses while the order aggregate consists of ordered items, shipping addresses, and payments. The payment also contains the billing address.



Error! Bookmark not defined.

If you notice a single logical address record appears 3 times in the data, but its value is copied each time wherever used. The whole address can be copied into an aggregate as needed. There is no pre-defined format to draw the aggregate boundaries. It solely depends on whether you want to manipulate the data as per your requirements.

The Data Model for customer and order would look like this.

```
// in customers
{
"customer": {
"id": 1,
"name": "Martin",
"billingAddress": [{"city": "Chicago"}],
"orders": [
 {
 "id":99,
 "customerId":1,
 "orderItems":[
 {
 "productId":27,
 "price": 32.45,
 "productName": "NoSQL Distilled"
 }
 ],
 "shippingAddress":[{"city":"Chicago"}]

 "orderPayment":[
 {
 "ccinfo":"1000-1000-1000-1000",
 "txnId":"abelif879rft",
 "billingAddress": {"city": "Chicago"}
 }],
 }]
```

}

}

In these Aggregate Data Models in NoSQL, if you want to access a customer along with all customer's orders at once. Then designing a single aggregate is preferable. But if you want to access a single order at a time, then you should have separate aggregates for each order. It is very content-specific.

2.Explain Cassandra Architecture.

Cassandra Architecture

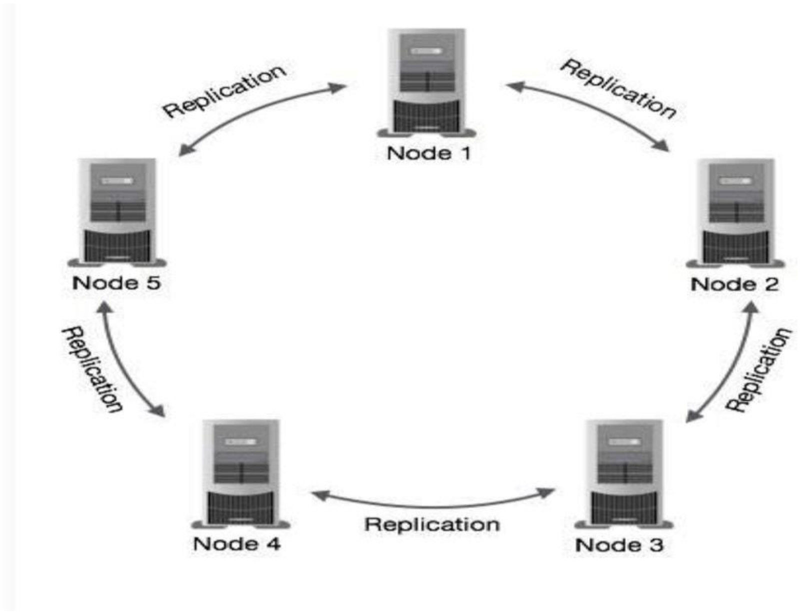
Cassandra was designed to handle big data workloads across multiple nodes without a single point of failure. It has a peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.

- 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.

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.



Components of Cassandra

The main components of Cassandra are:

- **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. 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.

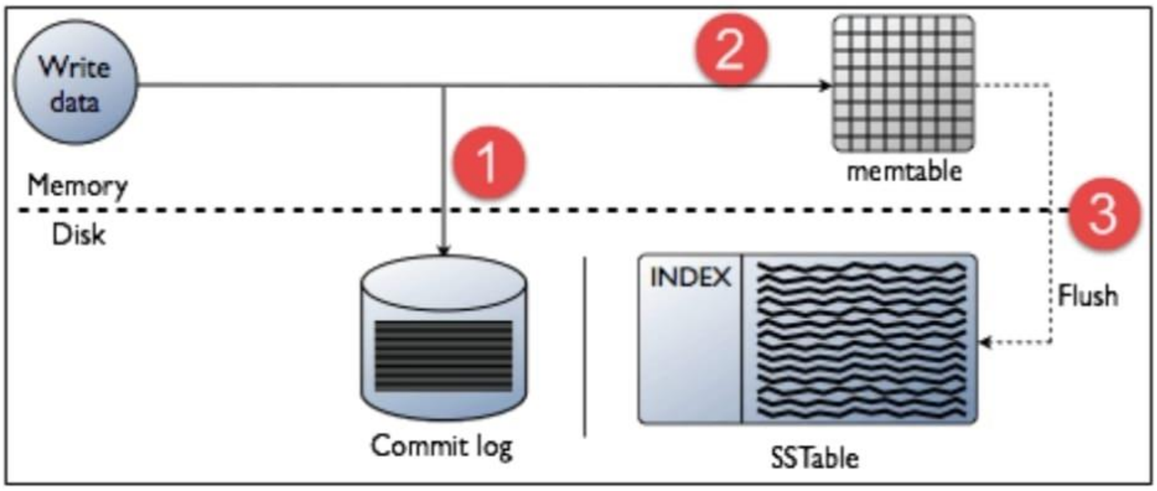
Cassandra Query Language

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.

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

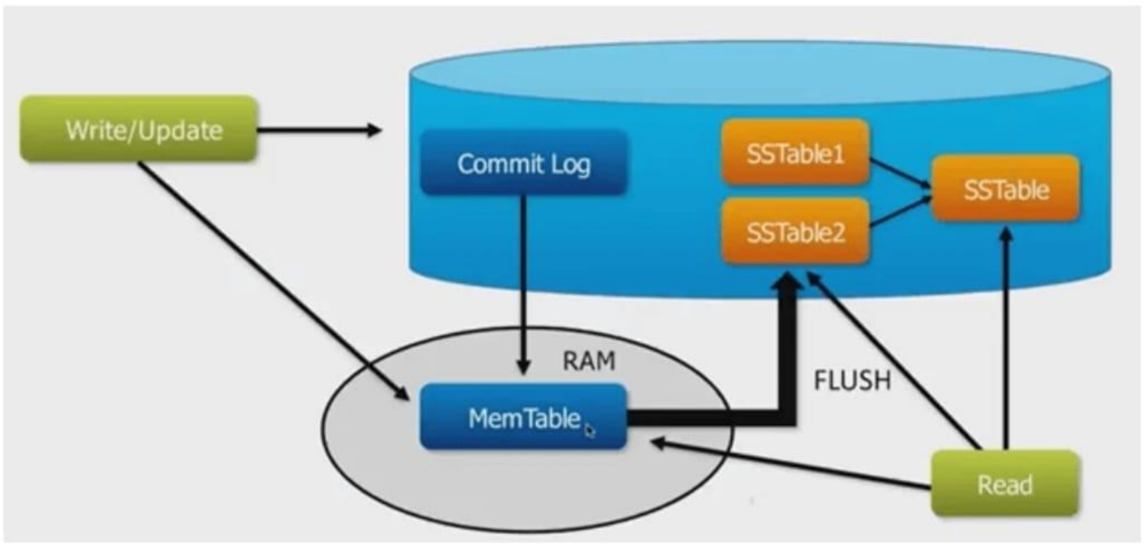
In Read operations, Cassandra gets values from the mem-table and checks the bloom filter to find the appropriate SSTable which contains the required data.

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.



3.Explain about NoSQL briefly.

NoSQL stands for "Not Only SQL" and refers to a type of database management system that is designed to handle large volumes of unstructured and semi-structured data. Unlike traditional SQL databases that use a tabular format with predefined schemas, NoSQL databases are schema-less and allow for flexible and dynamic data structures.

NoSQL databases are required because they can handle the large volumes and complex data types associated with Big Data. They are designed to scale horizontally by distributing data across many servers, making them well-suited for handling large and growing datasets. Additionally, NoSQL databases are often faster and more efficient than SQL databases for certain types of queries, such as those involving large amounts of data and complex data structures.

NoSQL databases are also used in modern web applications that require fast and flexible data storage, such as social media platforms, online marketplaces, and content management systems.

They are particularly useful for applications that require high levels of availability and scalability, as they can handle large amounts of traffic and data without sacrificing performance.

Different Types of NoSQL Databases

There are several types of NoSQL databases, each designed to handle different types of data and workloads. Some common types of NoSQL databases include:

Document Databases

These databases store and manage semi-structured data as documents, typically in JSON or XML formats. Document databases are well-suited for managing unstructured data, such as user profiles, product catalogs, or content management systems. Examples of document databases include MongoDB, Elasticsearch, and Couchbase.

Key-Value Databases

These databases store data as key-value pairs, making them ideal for simple lookups and high-speed data retrieval. Key-value databases are often used for caching, session management, and message queues. Examples of key-value databases include Redis and Riak.

Column-Family Databases

Also known as column-oriented databases, these databases store data as columns instead of rows, making them ideal for handling large amounts of data and complex queries. Column-family databases are often used for analytics, content management, and data warehousing. Examples of column-family databases include Apache Cassandra and HBase.

Graph Databases

These databases store and manage data as nodes and edges, making them well-suited for managing complex relationships and hierarchies. Graph databases are often used for social networks, recommendation engines, and fraud detection. Examples of graph databases include Neo4j and OrientDB.

CAP Theorem for NoSQL Database

The CAP theorem, also known as Brewer's theorem, is a fundamental concept in distributed computing that applies to NoSQL databases. The CAP theorem states that in any distributed system, it is impossible to simultaneously provide all three of the following guarantees:

1. **Consistency:** Every read request from a node in the system will return the most recent write request.
2. **Availability:** Every request to the system will receive a response without guaranteeing that it contains the most recent written request.
3. **Partition tolerance:** The system can continue to operate and function correctly even if there are network partitions or messages are lost between nodes.

In other words, when designing a distributed system like a NoSQL database, developers have to make trade-offs between consistency, availability, and partition tolerance. NoSQL databases are typically designed to prioritize either availability or partition tolerance while sacrificing some degree of consistency. This means that in certain failure scenarios, a NoSQL database may not provide the most up-to-date data to all nodes in the system but instead might return stale or conflicting data.

For example, in a partitioned network, a NoSQL database may prioritize partition tolerance and continue to accept writes from multiple nodes, but these nodes may have different versions of the same data. In contrast, a traditional relational database might prioritize consistency and reject writes until it can guarantee that all nodes have the most recent data.

Overall, the CAP theorem is an important consideration when designing and choosing a NoSQL database, as it helps to identify the trade-offs between consistency, availability, and partition tolerance that must be made in a distributed system.

Use of NoSQL Database

NoSQL databases are widely used for a variety of reasons, including:

- **Scalability:** NoSQL databases are highly scalable, allowing them to handle large amounts of data and high-traffic loads more easily than traditional relational databases.
- **Flexibility:** NoSQL databases allow for flexible data modeling, making it easier to handle unstructured or semi-structured data such as social media posts, documents, and sensor data.
- **Performance:** NoSQL databases are often faster than traditional relational databases, particularly when handling large volumes of data.
- **Availability:** NoSQL databases are designed to be highly available and fault-tolerant, ensuring that data is always accessible, even in the event of hardware or network failures.
- **Cost-effectiveness:** NoSQL databases can be more cost-effective than traditional relational databases, particularly for large-scale applications that require significant amounts of data storage and processing.

Common Use Cases for NoSQL Databases

Web applications: NoSQL databases are often used to power web applications, which require scalability, performance, and flexibility.

- **Big Data:** NoSQL databases are commonly used in big data applications, where traditional relational databases can struggle to handle the massive volumes of data involved.
- **Internet of Things (IoT):** NoSQL databases are used to store and process data from IoT devices, which can generate massive amounts of data in real time.
- **Real-Time Analytics:** NoSQL databases can be used for real-time analytics, enabling businesses to make faster, data-driven decisions.
- **Content Management:** NoSQL databases are often used for content management applications, which require the ability to handle unstructured or semi-structured data such as documents, images, and videos.

Big Data Technologies Using NoSQL

Big data technologies rely on NoSQL databases due to their scalability and ability to handle large volumes of unstructured and semi-structured data. Here are some of the most used big data technologies that leverage NoSQL databases:

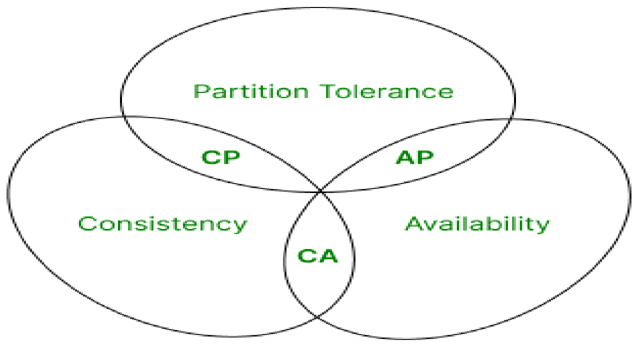
- **Hadoop:** Hadoop is a popular open-source big data platform that includes the Hadoop Distributed File System (HDFS) for storing and processing large amounts of data, and Apache HBase, a NoSQL column-family database that provides low-latency access to Hadoop data.
- **Cassandra:** Apache Cassandra is a highly scalable NoSQL column-family database that is often used in big data applications. Cassandra can handle massive amounts of data across multiple nodes and data centers, making it ideal for distributed systems.
- **MongoDB:** MongoDB is a popular document-oriented NoSQL database that is often used in big data applications. MongoDB can store and process large amounts of data, and its flexible data model makes it well-suited for handling unstructured data.
- **Couchbase:** Couchbase is a NoSQL document-oriented database that provides a distributed key-value store with high performance and scalability. It is often used in big data applications where real-time data access and processing are critical.
- **Neo4j:** Neo4j is a graph database that is often used in big data applications that require the processing of complex relationships between data points. Neo4j is well-suited for applications such as social networks, recommendation engines, and fraud detection systems.

Overall, NoSQL databases are a critical component of many big data architectures, enabling organizations to store and process large volumes of data efficiently and effectively.

4.what is CAP theorem in Big data?

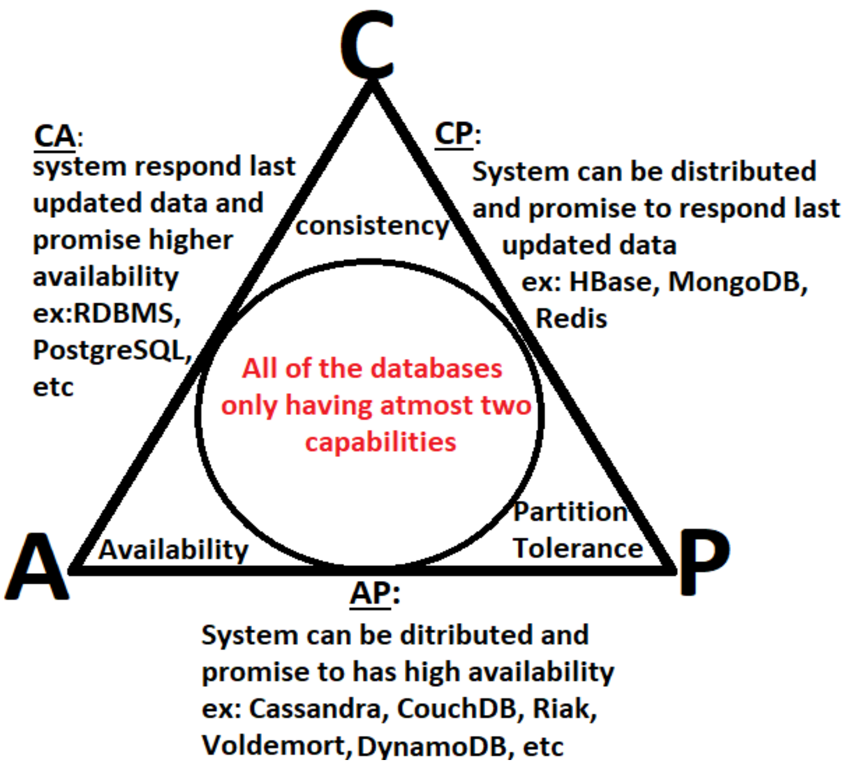
The CAP theorem (also known as Brewer's theorem) is a fundamental principle in distributed data systems, which are common in big data analytics frameworks. The theorem states that in a distributed system, you can guarantee only two out of the following three properties at the same time:

- **Consistency (C):** Every read receives the most recent write, or an error. This means that all nodes in the system have the same data at any given time, which is crucial for scenarios where real-time accuracy is required.
- **Availability (A):** Every request (read or write) receives a response, but without a guarantee that it contains the latest write. This ensures the system is always available, even if some nodes in the cluster are not functioning properly.
- **Partition Tolerance (P):** The system continues to operate, even if there is a network partition (a communication breakdown between nodes). This is essential for systems spread across multiple servers, regions, or data centers, as network issues are unavoidable in large distributed systems.



CAP Theorem in Big Data Analytics

The CAP theorem states that distributed databases can have at most two of the three properties: consistency, availability, and partition tolerance. As a result, database systems prioritize only two properties at a time.


C
CA:
system respond last updated data and promise higher availability
ex:RDBMS, PostgreSQL, etc
CP:
System can be distributed and promise to respond last updated data
ex: HBase, MongoDB, Redis
consistency
All of the databases only having atmost two capabilities
Availability
Partition Tolerance
A
P
AP:
System can be ditributed and promise to has high availability
ex: Cassandra, CouchDB, Riak, Voldemort, DynamoDB, etc

CP (Consistency and Partition Tolerance):

- The combination of consistency and availability is not possible in distributed systems and for achieving CA, the system has to be monolithic such that when a user updates the state of the system, all other users accessing it are also notified about the new changes which means that the consistency is maintained. And since it follows monolithic architecture, all users are connected to single system which means it is also available. These types of systems are generally not preferred due to a requirement of distributed computing which can be only done when consistency or availability is sacrificed for partition tolerance.
- Example databases: MySQL, PostgreSQL

AP (Availability and Partition Tolerance):

- The system prioritizes availability over consistency and can respond with possibly stale data which was replicated from other nodes before the partition was created due to some technical failure. Such design choices are generally used while building social media websites such as Facebook, Instagram, Reddit, etc. and online content websites like YouTube, blog, news, etc. where consistency is usually not required, and a bigger problem arises if the service is unavailable causing corporations to lose money since the users may shift to new platform. The system can be distributed across multiple nodes and is designed to operate reliably even in the face of network partitions.

- Example databases: Amazon DynamoDB, Google Cloud Spanner.

CA (Consistency and Availability):

- This combination is usually theoretical in distributed systems since partition tolerance is a given in large-scale distributed systems. CA might be feasible in smaller, localized systems where partitions are unlikely, but it's not practical in big data environments.
- The system prioritizes consistency over availability and does not allow users to read crucial data from the stored replica which was backed up prior to the occurrence of network partition. Consistency is chosen over availability for critical applications where latest data plays an important role such as stock market application, ticket booking application, banking, etc. where problem will arise due to old data present to users of application.
- Example databases: Apache HBase, MongoDB, Redis.

Practical Application in Big Data

In big data analytics frameworks, like Hadoop and Spark, consistency and availability trade-offs are managed depending on the task. For example:

- **Hadoop:** Prioritizes partition tolerance and availability, as it's designed for batch processing and large data stores across distributed systems. Temporary inconsistencies may be acceptable in exchange for high availability and the ability to manage partitions.
- **Spark:** Can handle some consistency by maintaining lineage (data history for fault recovery), allowing it to balance between consistency and availability, but it still emphasizes partition tolerance to manage data across clusters.

The CAP theorem thus guides the design and setup of data storage and processing in big data analytics systems, helping teams decide on the best approach for their use case and requirements.

5.Explain in detail about schemaless databases in NoSQL.

A schemaless database in the context of NoSQL refers to a database that does not require a predefined schema or structure for the data it stores. Unlike traditional relational databases (RDBMS), where the data is stored in tables with predefined columns and data types, NoSQL databases allow for more flexible and dynamic data models. This can be advantageous in situations where the structure of the data is expected to change frequently, or when you are dealing with complex and unstructured data.

Key Features of Schemaless Databases:

- **No Fixed Schema:** There's no need to define tables or fields ahead of time. The database can store different kinds of data, and each document or entry can have different attributes.
- **Flexible Data Models**: The data can be stored as key-value pairs, documents, graphs, or wide-column stores, depending on the NoSQL database type.
- **Adaptability:** As the data model evolves, the database can easily accommodate new attributes or fields without requiring changes to the entire database structure.
- **Horizontal Scalability:** Many NoSQL databases (like MongoDB, Cassandra, and Couchbase) are designed for horizontal scaling, meaning they can distribute data across multiple servers or nodes to handle large volumes of data.

Types of NoSQL Databases and Their Models

Document-Based (e.g., MongoDB, CouchDB):

- Stores data in flexible JSON-like documents (BSON in MongoDB).
- Each document can have a unique structure.
- You don't need to define fields and types upfront; documents within the same collection can have different structures.

Key-Value Stores (e.g., Redis, DynamoDB):

- Data is stored as key-value pairs, where the key is unique and points to a value that can be anything (string, object, list, etc.).
- No schema is enforced for the values, so they can change in structure at any time.

Column-Based (e.g., Cassandra, HBase):

- Data is stored in tables, but the structure is more flexible than relational databases.
- Columns can vary across rows, and new columns can be added without altering the entire database structure.

Graph Databases (e.g., Neo4j, ArangoDB):

- Data is stored as nodes, edges, and properties, which makes it suitable for representing relationships.
- No predefined schema is required for storing graph data, and relationships between entities can be dynamically adjusted.
-

Benefits of Schemaless Databases

- **Flexibility:** You can easily store diverse types of data without having to fit everything into a rigid schema.
- **Agility:** Since you don't need to define a strict schema, development cycles are faster, and it's easier to adapt to new data types or changing business needs.
- **Easy to Scale:** Many NoSQL databases are optimized for horizontal scaling, making them ideal for handling large amounts of data across distributed systems.
- **Handling Unstructured Data:** Schemaless databases are well-suited for unstructured or semi-structured data, such as documents, logs, multimedia content, and sensor data.
- **Evolving Data Models**: As the application evolves, data models can be adjusted without causing disruptions in the existing data or requiring database migrations.

Drawbacks of Schemaless Databases

- **Lack of Consistency:** Without a schema, there is no guarantee that the data will be consistent. This can lead to data anomalies, especially in large-scale applications.
- **Complex Queries:** Since data doesn't have a predefined structure, querying and indexing can be less efficient, especially for complex queries that require JOIN operations (a common feature in relational databases).
- **Data Validation:** Without a schema, ensuring the integrity of data can be harder. In relational databases, constraints like NOT NULL, UNIQUE, or CHECK are enforced at the schema level, but in NoSQL, this often has to be handled at the application level.

- **Data Redundancy:** Schemaless models may lead to data duplication, as each document or record can store its own full set of data, even if some of it is redundant across entries.

Use Cases for Schemaless Databases

- **Rapid Development:** Applications that are in the early stages or are frequently changing, such as start-ups or prototypes.
- **Content Management**: Systems that need to handle diverse and evolving content types, such as blogs, social media posts, or user-generated content.
- **Real-Time Analytics**: Applications that need to ingest large volumes of diverse and unstructured data in real-time, such as logs, sensor data, or IoT applications.
- **Big Data:** Projects that need to store and process massive datasets with complex, unstructured data.

Popular Schemaless NoSQL Databases

- **MongoDB:** A widely used document store that supports flexible, JSON-like documents.
- **Cassandra:** A distributed, wide-column store optimized for handling large amounts of data across many nodes.
- **Redis:** A key-value store known for its speed and support for various data types.
- **DynamoDB:** Amazon's managed key-value store, designed for high availability and scalability.
- **Couchbase:** A distributed NoSQL database that supports both key-value and document-oriented storage.
- **Neo4j:** A graph database for modeling relationships between entities.