ADBMS_UNIT-3

Introduction, Overview, and History of NoSQL Databases- The definition of Four Types of No SQL Databases.

What is a NoSQL database?

When people use the term “NoSQL database,” they typically use it to refer to any non-relational database. Some say the term “NoSQL” stands for “non-SQL” while others say it stands for “not only SQL.” Either way, most agree that NoSQL databases store data in a more natural and flexible way. NoSQL, as opposed to SQL, is a database management approach, whereas SQL is just a query language, similar to the query languages of NoSQL databases.

Types of databases — NoSQL

Over time, four major **types of NoSQL databases** have emerged: document databases, **key-value databases**, wide-column stores, and graph databases. Nowadays, multi-model databases are also becoming quite popular.

Document-oriented databases

A document-oriented database stores data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. The values can typically be a variety of types, including things like strings, numbers, booleans, arrays, or even other objects. A document database offers a flexible data model, much suited for semi-structured and typically unstructured data sets. They also support nested structures, making it easy to represent complex relationships or hierarchical data.

Examples of document databases are **MongoDB** and Couchbase. A typical document will look like the following:

```
Code Snippet
1
```

```
{2
  "_id": "12345",3
  "name": "foo bar",4
  "email": "foo@bar.com",5
  "address": {6
    "street": "123 foo street",7
    "city": "some city",8
    "state": "some state",9
    "zip": "123456"10
  },11
  "hobbies": ["music", "guitar", "reading"]12
}
```

Key-value databases

A key-value store is a simpler type of database where each item contains keys and values. Each key is unique and associated with a single value. They are used for caching and session management and provide high performance in reads and writes because they tend to store things in memory. Examples are Amazon DynamoDB and Redis. A simple view of data stored in a key-value database is given below:

```
Code Snippet
1
Key: user:123452
Value: {"name": "foo bar", "email": "foo@bar.com", "designati
on": "software developer"}
```

Wide-column stores

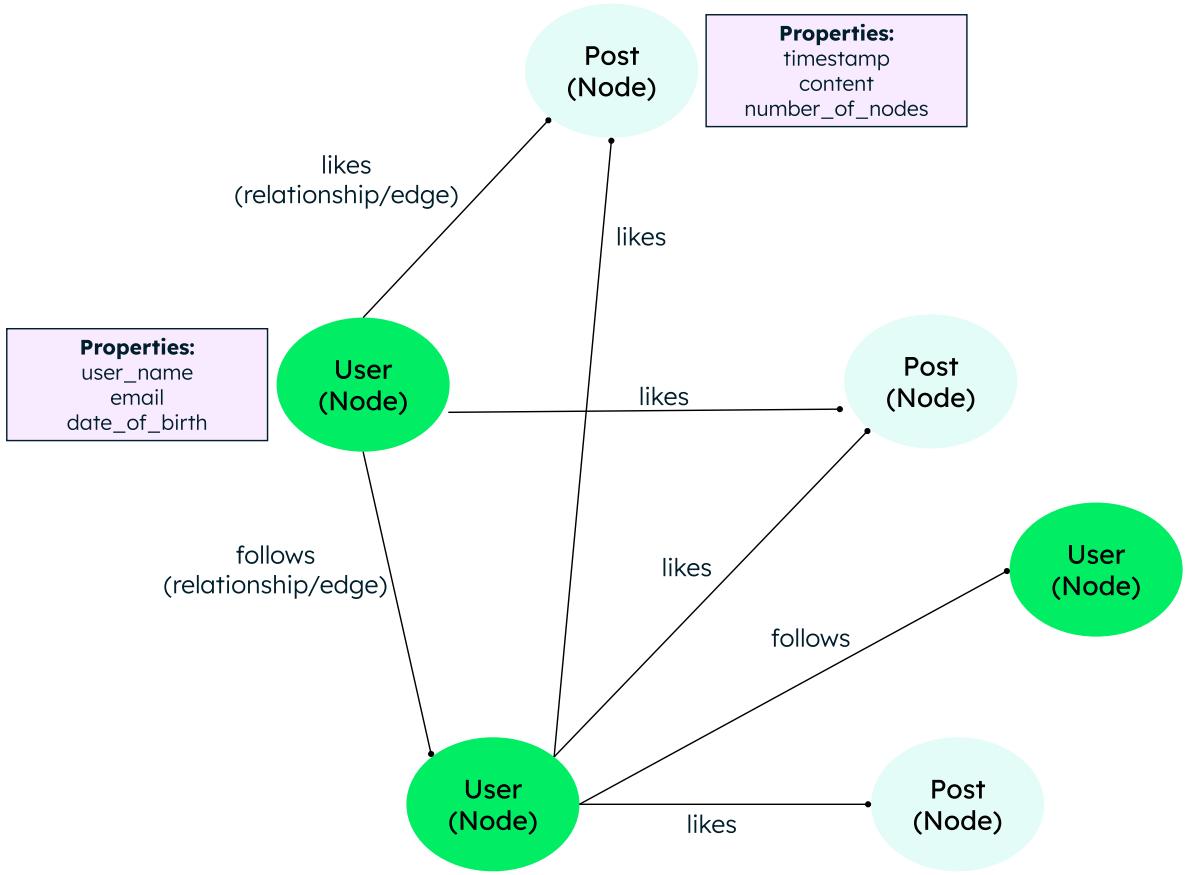
Wide-column stores store data in tables, rows, and dynamic columns. The data is stored in tables. However, unlike traditional SQL databases, wide-column stores are flexible, where different rows can have different sets of columns. These databases can employ column compression techniques to reduce the storage space and enhance performance. The wide rows and columns enable efficient retrieval of sparse and wide data. Some examples of wide-column stores are

Apache Cassandra and HBase. A typical example of how data is stored in a wide-column is as follows:

name	id	email	dob	city
Foo bar	12345	foo@bar.com		Some city
Carn Yale	34521	bar@foo.com	12-05-1972	

Graph databases

A graph database stores data in the form of nodes and edges. Nodes typically store information about people, places, and things (like nouns), while edges store information about the relationships between the nodes. They work well for highly connected data, where the relationships or patterns may not be very obvious initially. Examples of graph databases are Neo4J and Amazon Neptune. MongoDB also **provides graph traversal capabilities** using the $graphLookup stage of the aggregation pipeline. Below is an example of how data is stored:

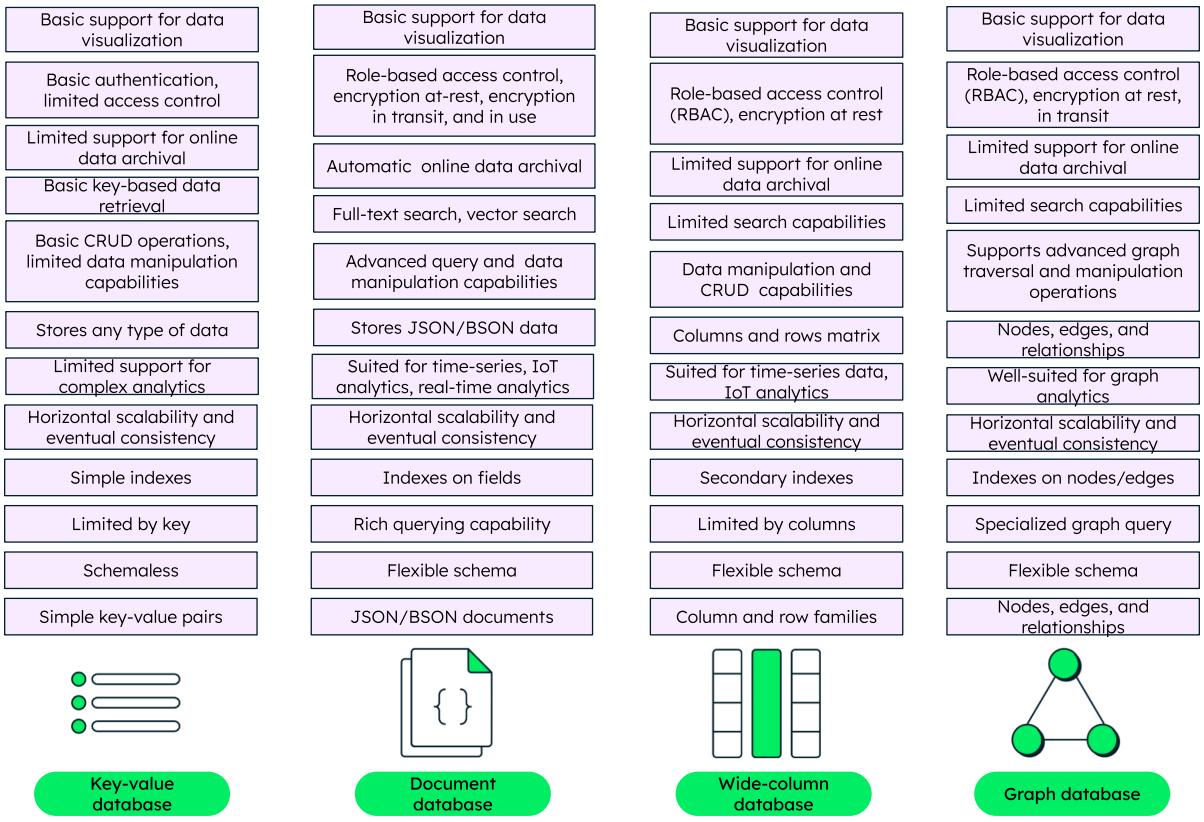


Multi-model databases

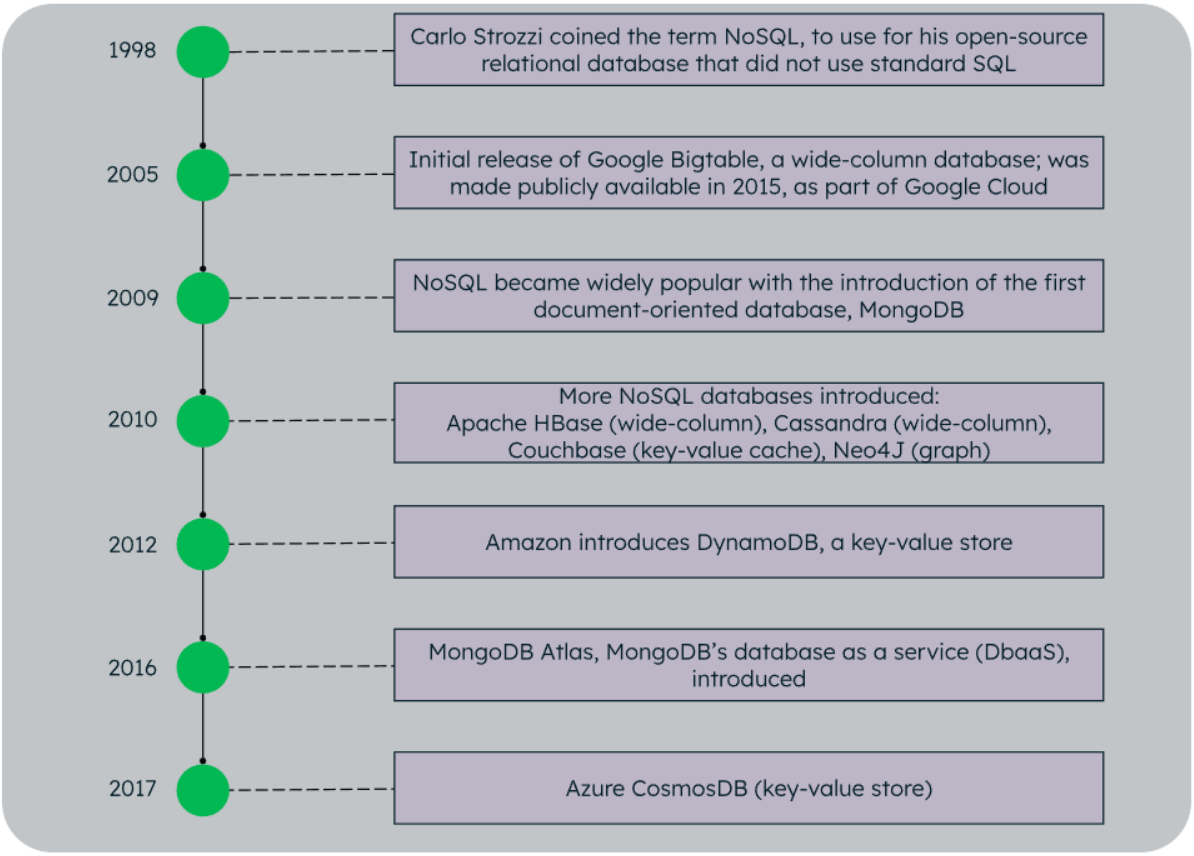
Multi-model databases support more than one type of NoSQL data model so that developers can choose based on their application requirements. These databases have a unified database engine that can handle multiple data models within a database instance. Examples are CosmosDB and ArangoDB.

Quick comparison of types of databases — NoSQL

Each of the NoSQL databases offers different features. For example, graph databases could be more suited for analyzing complex relationships and patterns between entities, while document databases provide a more flexible, natural way of storing and retrieving large data volumes of similar types as documents. The choice of database depends on the use case you want to develop.



Brief history of NoSQL databases

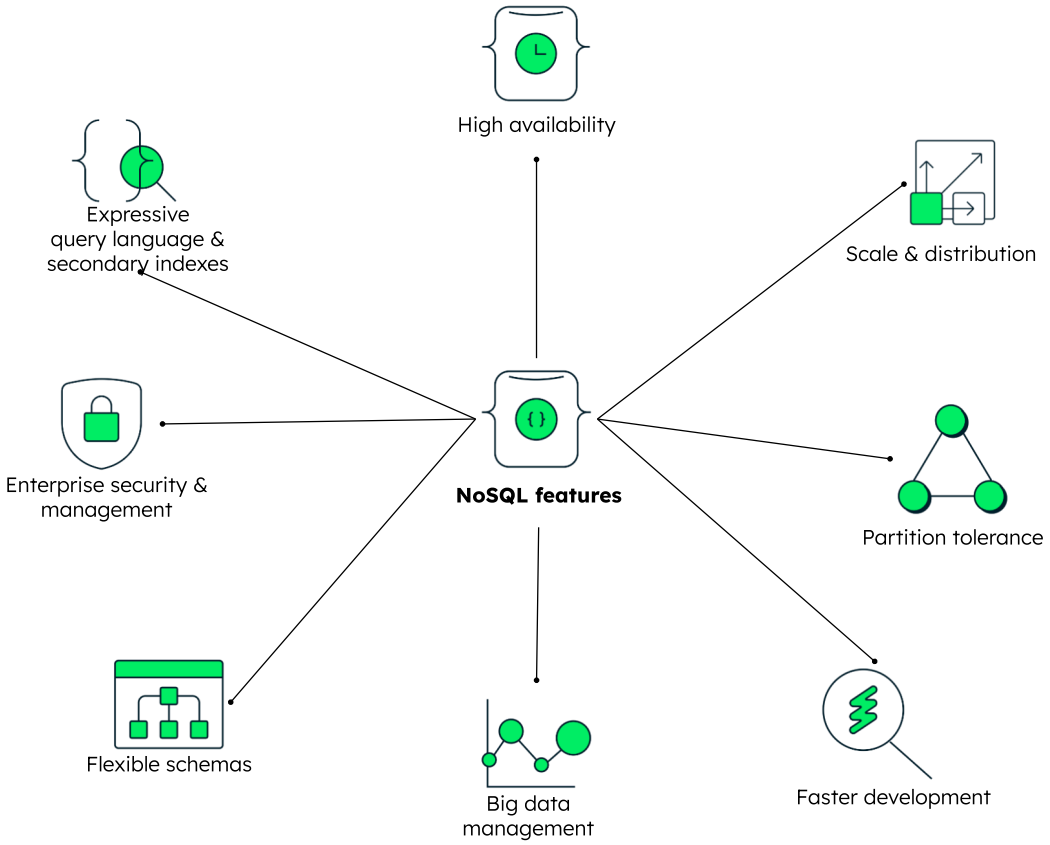


Additionally, the Agile Manifesto was rising in popularity, and software engineers were rethinking the way they developed software. They had to rapidly adapt to changing requirements, iterate quickly, and make changes throughout their software stack — all the way down to the database. NoSQL databases gave them this flexibility.

Cloud computing also rose in popularity, and developers began using public clouds to host their applications and data. They wanted the ability to distribute data across multiple servers and regions to make their applications resilient, to scale out instead of scale up, and to intelligently geo-place their data. Some NoSQL databases, like **MongoDB Atlas**, provide these capabilities.

NoSQL database features

NoSQL databases are flexible, scalable, and distributed databases. Different types of NoSQL databases have their own unique features.



Relational database vs NoSQL database example

Let's consider an example of storing information about a user and their hobbies. We need to store a user's first name, last name, cell phone number, city, and hobbies.

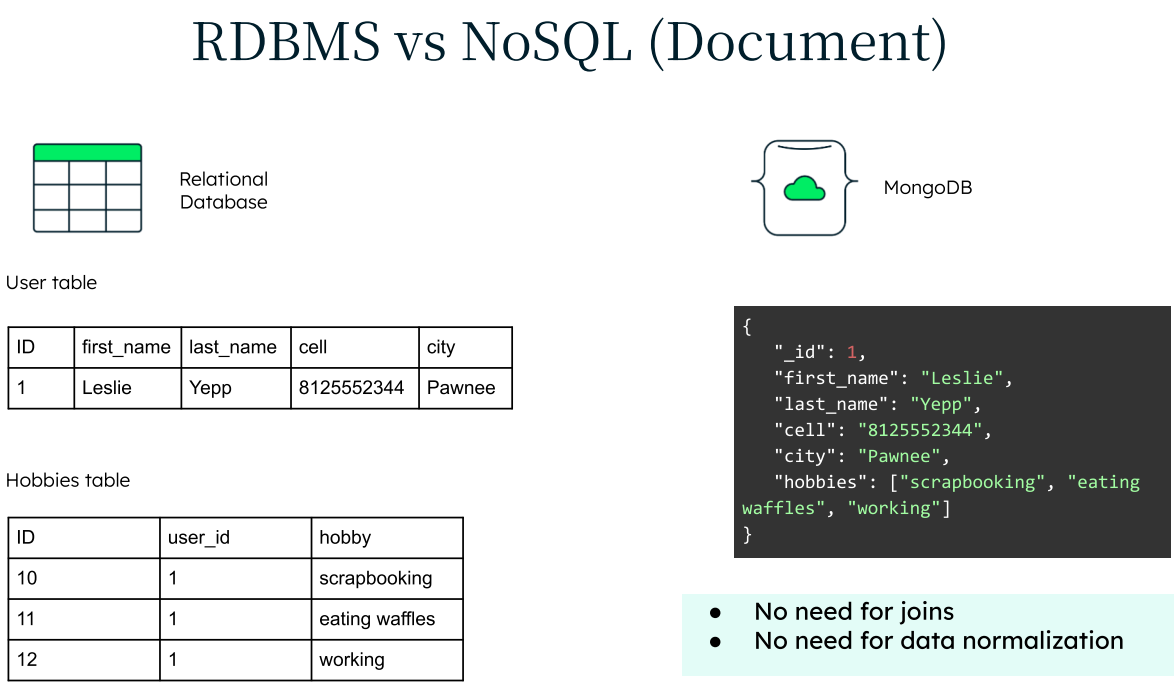
In a relational database management system (RDBMS), we'd likely create two tables: one for Users and one for Hobbies.

In order to retrieve all of the information about a user and their hobbies, information from the Users table and Hobbies table will need to be joined together.

The data model we design for a NoSQL database will depend on the type of NoSQL database we choose. Let's consider how to store the same information about a user and their hobbies in a **document database** like MongoDB.

In order to retrieve all of the information about a user and their hobbies, a single document can be retrieved from the database. No joins are required, resulting in

faster queries.



To see a more detailed version of this data modeling example, read **Mapping Terms and Concepts From SQL to MongoDB**.

Differences between RDBMS and NoSQL databases

There are a variety of differences between relational database management systems and non-relational databases. One of the key differences is the way data is modeled in the database. Some key differences of each feature is listed below:

Data modeling

NoSQL: Data models vary based on the type of NoSQL database used — for example, key-value, document, graph, and wide-column — making the model suitable for semi-structured and unstructured data.

RDBMS: RDBMS uses a tabular data structure, with data represented as a set of rows and columns, making the model suitable for structured data.

Schema

NoSQL: It provides a flexible schema where each set of documents/row-column/key-value pairs can contain different types of data. It's easier to change

schema, if required, due to the flexibility.

RDBMS: This is a fixed schema where every row should contain the same predefined column types. It is difficult to change the schema once data is stored.

Query language

NoSQL: It varies based on the type of NoSQL database used. For example, MongoDB has **MQL**, and Neo4J uses Cypher.

RDBMS: This uses structured query language (SQL).

Scalability

NoSQL: NoSQL is designed for vertical and horizontal scaling.

RDBMS: RDBMS is designed for vertical scaling. However, it can extend limited capabilities for horizontal scaling.

Data relationships

NoSQL: Relationships can be nested, explicit, or implicit.

RDBMS: Relationships are defined through foreign keys and accessed using joins.

Transaction type

NoSQL: Transactions are either **ACID**- or BASE-compliant.

RDBMS: Transactions are ACID-compliant.

Performance

NoSQL: NoSQL is suitable for real-time processing, big data analytics, and distributed environments.

RDBMS: RDBMS is suitable for read-heavy and transaction workloads.

Data consistency

NoSQL: This offers high data consistency.

RDBMS: This offers eventual consistency, in most cases.

Distributed computing

NoSQL: One of the main reasons to introduce NoSQL was for distributed computing, and NoSQL databases support distributed data storage, vertical and horizontal scaling through sharding, replication, and clustering.

RDBMS: RDBMS supports distributed computing through clustering and replication. However, it's less scalable and flexible as it's not traditionally designed to support distributed architecture.

Fault tolerance

NoSQL: NoSQL has built-in fault tolerance and high availability due to data replication.

RDBMS: RDBMS uses replication, backup, and recovery mechanisms. However, as they are designed for these, additional measures like disaster recovery mechanisms may need to be implemented during application development.

Data partitioning

NoSQL: It's done through sharding and replication.

RDBMS: It supports table-based partitioning and partition pruning.

*Learn more about **data partitioning here**.*

Data to object mapping

NoSQL: NoSQL stores the data in a variety of ways — for example, as JSON documents, wide-column stores, or key-value pairs. It provides abstraction through the ODM (object-data mapping) frameworks to work with NoSQL data in an object-oriented manner.

RDBMS: RDBMS relies more on data-to-object mapping so that there is seamless integration between the database columns and the object-oriented application code.

To learn more about the differences between relational databases and NoSQL databases, read **NoSQL vs SQL Databases**.

When should NoSQL be used?

When deciding which database to use, decision-makers typically find one or more of the following factors that lead them to select a NoSQL database:

- Fast-paced Agile development
- Storage of structured and semi-structured data
- Huge volumes of data
- Requirements for scale-out architecture
- Modern application paradigms like microservices and real-time streaming

Apache Cassandra

Apache Cassandra is an open-source NoSQL database that is used for handling big data. Apache Cassandra has the capability to handle structured, semi-structured, and unstructured data. Apache Cassandra was originally developed at Facebook after that it was open-sourced in 2008 and after that, it became one of the top-level Apache projects in 2010.

Features of Cassandra:

1. it is scalable.
2. it is flexible (can accept structured, semi-structured, and unstructured data).
3. it doesn't support **ACID** Transactions
4. it is highly available and **fault-tolerant**.
5. it is open source.

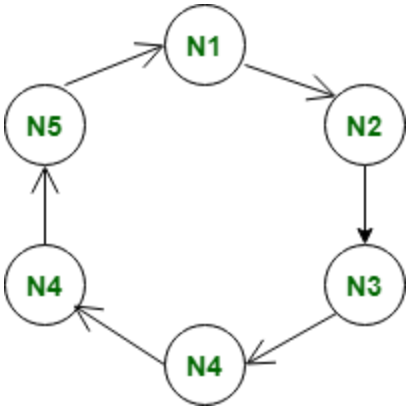


Figure-1: Masterless ring architecture of Cassandra

Apache Cassandra is a highly scalable, distributed database that strictly follows the principle of the CAP (Consistency Availability and Partition tolerance) theorem.

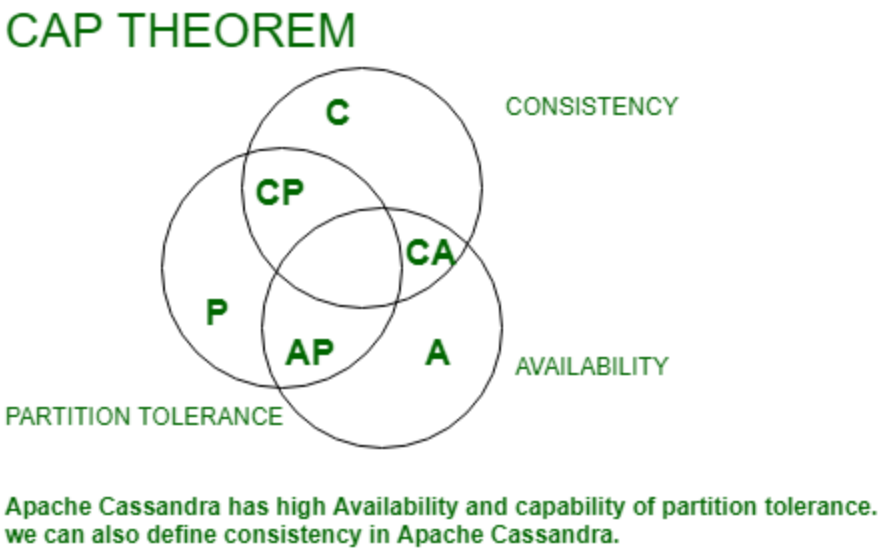


Figure-2: CAP Theorem

In Apache Cassandra, there is no master-client architecture. It has a peer-to-peer architecture. In Apache Cassandra, we can create multiple copies of data at the time of keyspace creation. We can simply define replication strategy and RF (Replication Factor) to create multiple copies of data. **Example:**

```
CREATE KEYSPACE Example
WITH replication = {'class': 'NetworkTopologyStrategy',
                            'replication_factor': '3'};
```

In this example, we define RF (Replication Factor) as 3 which simply means that we are creating here 3 copies of data across multiple nodes in a clockwise direction.

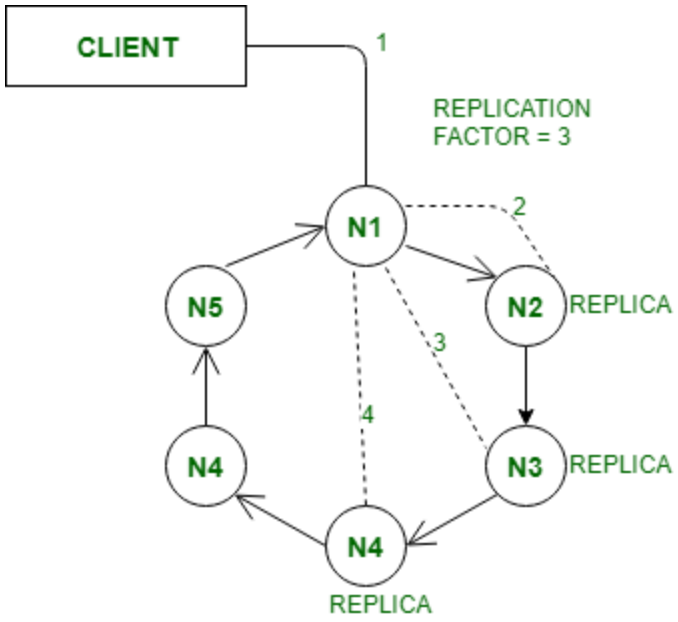


Figure-3:

RF = 3

cqlsh: CQL shell cqlsh is a command-line shell for interacting with Cassandra through CQL (Cassandra Query Language). **CQL query for Basic Operation:**

Step1: To create keyspace use the following CQL query.

```
CREATE KEYSPACE Emp
WITH replication = {'class': 'SimpleStrategy',
                            'replication_factor': '1'};
```

Step2: CQL query for using keyspace

```
Syntax:
USE keyspace-name
USE Emp;
```

Step-3: To create a table use the following CQL query.

```
Example:
CREATE TABLE Emp_table (
    name text PRIMARY KEY,
```

```
    Emp_id int,
    Emp_city text,
    Emp_email text,
    );
```

Step-4: To insert into Emp_table use the following CQL query.

```
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('ashish', 1001, 'Delhi', 'ashish05.rana05@gmail.co
m');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('Ashish Gupta', 1001, 'Bangalore', 'ashish@gmail.co
m');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('amit ', 1002, 'noida', 'abc@gmail.com');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('dhruv', 1003, 'pune', 'xyz@gmail.com');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('shivang', 1004, 'mumbai', 'test@gmail.com');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('aayush', 1005, 'gurugram', 'cass_write@gmail.co
m');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('bhagyesh', 1006, 'chandigar', 'welcome@gmail.co
m');
```

Step-5: To read data use the following CQL query.

```
SELECT * FROM Emp_table;
```

Key Differences Between Apache Cassandra and MongoDB

Feature	Apache Cassandra	MongoDB

Data Model	Wide-column store, suited for time-series and analytics. Uses tables, rows, and columns.	Document-oriented, stores data as BSON (binary JSON).
Schema	Flexible but schema-defined; tables need column definitions.	Schema-less, documents can have varying fields.
Query Language	CQL (Cassandra Query Language), similar to SQL.	Query syntax based on JSON-like structure.
Scalability	Optimized for horizontal scalability; excels in handling large-scale distributed data.	Scalable, but performance can vary under heavy write loads.
Replication Model	Supports tunable consistency with multiple replication strategies.	Replica sets for redundancy, with a primary-secondary architecture.
Consistency	Tunable consistency (eventual or strong).	Strong consistency by default; eventual consistency in sharded clusters.
Use Cases	High write throughput, IoT, real-time analytics.	Content management, catalog systems, real-time analytics.
Write Performance	Extremely high write throughput.	Moderate, can be slower under certain conditions.
Transactions	Limited support for ACID transactions.	Full ACID transactions support.
Geospatial Support	Limited geospatial features.	Extensive support for geospatial queries.
Community and Ecosystem	Strong backing from Apache, with numerous integrations.	Large community and integrations, driven by MongoDB Inc.

Features , Advantages , Disadvantages of NoSql

Key Features of NoSQL:

1. **Dynamic schema:** NoSQL databases do not have a fixed schema and can accommodate changing data structures without the need for migrations or schema alterations.

2. **Horizontal scalability:** NoSQL databases are designed to scale out by adding more nodes to a database cluster, making them well-suited for handling large amounts of data and high levels of traffic.
3. **Document-based:** Some NoSQL databases, such as MongoDB, use a document-based data model, where data is stored in a schema-less semi-structured format, such as JSON or BSON.
4. **Key-value-based:** Other NoSQL databases, such as Redis, use a key-value data model, where data is stored as a collection of key-value pairs.
5. **Column-based:** Some NoSQL databases, such as Cassandra, use a column-based data model, where data is organized into columns instead of rows.
6. **Distributed and high availability:** NoSQL databases are often designed to be highly available and to automatically handle node failures and data replication across multiple nodes in a database cluster.
7. **Flexibility:** NoSQL databases allow developers to store and retrieve data in a flexible and dynamic manner, with support for multiple data types and changing data structures.
8. **Performance:** NoSQL databases are optimized for high performance and can handle a high volume of reads and writes, making them suitable for big data and real-time applications.

Advantages of NoSQL: There are many advantages of working with NoSQL databases such as MongoDB and Cassandra. The main advantages are high scalability and high availability.

1. **High scalability:** NoSQL databases use sharding for horizontal scaling. Partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved is sharding. Vertical scaling means adding more resources to the existing machine whereas horizontal scaling means adding more machines to handle the data. Vertical scaling is not that easy to implement but horizontal scaling is easy to implement. Examples of horizontal scaling databases are MongoDB, Cassandra, etc. NoSQL can handle a huge amount of data because of scalability, as the data grows NoSQL scales**The auto** itself to handle that data in an efficient manner.

2. **Flexibility:** NoSQL databases are designed to handle unstructured or semi-structured data, which means that they can accommodate dynamic changes to the data model. This makes NoSQL databases a good fit for applications that need to handle changing data requirements.
3. **High availability:** The auto**,** replication feature in NoSQL databases makes it highly available because in case of any failure data replicates itself to the previous consistent state.
4. **Scalability:** NoSQL databases are highly scalable, which means that they can handle large amounts of data and traffic with ease. This makes them a good fit for applications that need to handle large amounts of data or traffic
5. **Performance:** NoSQL databases are designed to handle large amounts of data and traffic, which means that they can offer improved performance compared to traditional relational databases.
6. **Cost-effectiveness:** NoSQL databases are often more cost-effective than traditional relational databases, as they are typically less complex and do not require expensive hardware or software.
7. **Agility:** Ideal for agile development.

Disadvantages of NoSQL: NoSQL has the following disadvantages.

1. **Lack of standardization:** There are many different types of NoSQL databases, each with its own unique strengths and weaknesses. This lack of standardization can make it difficult to choose the right database for a specific application
2. **Lack of ACID compliance:** NoSQL databases are not fully ACID-compliant, which means that they do not guarantee the consistency, integrity, and durability of data. This can be a drawback for applications that require strong data consistency guarantees.
3. **Narrow focus:** NoSQL databases have a very narrow focus as it is mainly designed for storage but it provides very little functionality. Relational databases are a better choice in the field of Transaction Management than NoSQL.
4. **Open-source:** NoSQL is an **database**open-source database. There is no reliable standard for NoSQL yet. In other words, two database systems are

likely to be unequal.

5. **Lack of support for complex queries:** NoSQL databases are not designed to handle complex queries, which means that they are not a good fit for applications that require complex data analysis or reporting.
6. **Lack of maturity:** NoSQL databases are relatively new and lack the maturity of traditional relational databases. This can make them less reliable and less secure than traditional databases.
7. **Management challenge:** The purpose of big data tools is to make the management of a large amount of data as simple as possible. But it is not so easy. Data management in NoSQL is much more complex than in a relational database. NoSQL, in particular, has a reputation for being challenging to install and even more hectic to manage on a daily basis.
8. **GUI is not available:** GUI mode tools to access the database are not flexibly available in the market.
9. **Backup:** Backup is a great weak point for some NoSQL databases like MongoDB. MongoDB has no approach for the backup of data in a consistent manner.
10. **Large document size:** Some database systems like MongoDB and CouchDB store data in JSON format. This means that documents are quite large (BigData, network bandwidth, speed), and having descriptive key names actually hurts since they increase the document size.

Features of MongoDB

- **Schema-less Database:** It is the great feature provided by the MongoDB. A **Schema-less database means one collection can hold different types of documents in it**. Or in other words, in the MongoDB database, a single collection can hold multiple documents and these documents may consist of the different numbers of fields, content, and size. It is **not necessary that the one document is similar to another document like in the relational databases**. Due to this cool feature, MongoDB provides great flexibility to databases.

- **Document Oriented:** In MongoDB, **all the data stored in the documents instead of tables like in RDBMS**. In these documents, the data is stored in fields(key-value pair) instead of rows and columns which make the data much more flexible in comparison to RDBMS. And each document contains its unique object id.
- **Indexing:** In MongoDB database, **every field in the documents is indexed with primary and secondary indices this makes easier and takes less time to get or search data from the pool of the data**. If the data is not indexed, then database search each document with the specified query which takes lots of time and not so efficient.
- **Scalability:** MongoDB **provides horizontal scalability with the help of sharding. Sharding** means to distribute data on multiple servers, here a large amount of data is partitioned into data chunks using the shard key, and these data chunks are evenly distributed across shards that reside across many physical servers. It will also add new machines to a running database.
- **Replication:** MongoDB provides high availability and redundancy with the help of replication, it creates multiple copies of the data and sends these copies to a different server so that if one server fails, then the data is retrieved from another server.
- **Aggregation:** It **allows to perform operations on the grouped data and get a single result or computed result**. It is similar to the SQL GROUPBY clause. It provides three different **aggregations** i.e, aggregation pipeline, map-reduce function, and single-purpose aggregation methods
- **High Performance:** The performance of MongoDB is very high and data persistence as compared to another database due to its features like scalability, indexing, replication, etc.

Advantages of MongoDB

- It is a schema-less NoSQL database. You need not to design the schema of the database when you are working with MongoDB.
- It does not support join operation.
- It provides great flexibility to the fields in the documents.

- It contains heterogeneous data.
- It provides high performance, availability, scalability.
- It supports **Geospatial** efficiently.
- It is a document oriented database and the data is stored in BSON documents.
- It also supports multiple document ACID transition(string from MongoDB 4.0).
- It does not require any SQL injection.
- It is easily integrated with **Big Data Hadoop**

Disadvantages of MongoDB

- It uses high memory for data storage.
- You are not allowed to store more than 16MB data in the documents.
- The nesting of data in BSON is also limited you are not allowed to nest data more than 100 levels.

Advantages of column-oriented databases

Organizations that handle big data and invest in analytics should consider the strengths of column-oriented databases. They excel in efficiently storing and querying large data sets for the following reasons:

- **Scalability.** A column database's primary advantage is the ability to handle big data. Depending on the scale of the database, it can cover hundreds of different machines. Columnar databases support massively parallel processing, employing many processors to work on the same set of computations simultaneously.
- **Compression.** Compressing large amounts of data saves storage space.
- **Very responsive.** The load time is minimal and columnar databases perform queries quickly, making them practical for big data and analytics.
- **Aggregation performance.** Columnar databases can scan and aggregate large volumes of data in columns efficiently. Aggregations such as averages

and sums are faster because the database engine reads only the necessary columns, not entire rows. Aggregations that need to read the entire column before updating a value -- such as a distinct count or sorting -- are generally much faster than row-oriented databases.

- **Flexibility.** In general, column-oriented databases are less flexible than row-oriented architectures for general-purpose database work. However, the columnar architecture is more adaptable in certain situations. For example, adding or removing columns as schemas evolve is generally easier than in a row-based system because only the affected column must change. In a traditional database, every row of data needs updates. Schema flexibility can be valuable when analytic requirements change frequently or evolve over time.

Disadvantages of column-oriented databases

Column-oriented databases have several downsides that users must navigate. In addition to having potential security vulnerabilities, they struggle to support the following:

- **Online transactional processing.** Column databases are not as efficient with online transactional processing as they are for online analytical processing (OLAP). They can analyze transactions, but struggle to update them. A common strategy is to have the column database hold the data required for business analysis and have a relational database store the data in the back end.
- **Incremental data loading.** Column-oriented databases can quickly retrieve data for analysis, even when processing complex queries. Incremental data loads are not impossible, but columnar databases do not perform them in the most efficient way. It must scan the columns to identify the right rows and then conduct another scan to locate the modified data that requires overwriting.
- **Row-specific queries.** The disadvantages of column databases all boil down to the same issue -- using the right type of database for the right purposes. Row-specific queries introduce an extra step of scanning the columns to identify the rows and then locating the data to retrieve. It takes more time to

get to individual records scattered in multiple columns, rather than accessing grouped records in a single column. Frequent row-specific queries might cause performance issues by slowing down a column-oriented database, which defeats its purpose of providing information quickly.

- **Security.** Columnar databases are slightly more vulnerable to some security issues than other types of databases because they rely on data compression -- especially of commonly repeated values -- to improve performance. Compression can conflict with encryption. Encryption might be less effective if compression occurs first because patterns in the compressed data might remain. However, encrypting the data first can limit the performance benefits from compression. One mitigating factor for potential vulnerabilities is that the most sensitive data, such as unique identifiers, are less likely to effectively compress.

Features of Column-Oriented Databases:

1. **Data Storage:**
    - Stores data by columns rather than rows.
    - Each column is stored in a separate file or storage block, enabling efficient compression and retrieval for columnar data.
2. **Optimized for Analytical Workloads:**
    - Well-suited for read-heavy workloads, such as analytical queries and reporting.
    - Allows fast aggregation of values in a single column (e.g., summing up sales).
3. **High Compression Ratios:**
    - Columns often have similar data types and values, allowing high compression rates.
    - Reduces storage costs and improves performance for large datasets.
4. **Efficient Query Performance:**

- Queries that access only a subset of columns are significantly faster.
- Avoids reading irrelevant data, unlike row-oriented databases.

5. **Scalability:**
    - Supports distributed storage and processing, making it ideal for handling large-scale datasets.
6. **Schema Flexibility:**
    - Often schema-less or supports flexible schemas, allowing easy adjustments to data structures without major overhead.
7. **Parallel Processing:**
    - Columns can be processed independently, enabling better utilization of multicore processors and parallel computing resources.

Features , adv, disadv of cassandra

Main Features of Apache Cassandra:

1. **Decentralized Architecture:**
    - Cassandra operates on a peer-to-peer distributed system with no single point of failure.
    - Each node in the cluster has equal responsibility and can handle any request, ensuring reliability.
2. **High Scalability:**
    - Horizontally scalable by simply adding more nodes to the cluster.
    - Ideal for managing large-scale, high-velocity data workloads without downtime.
3. **Continuous Availability:**
    - Designed for applications requiring 24/7 uptime.

- Features automatic replication and failover, ensuring data remains accessible even in case of node failures.

4. **Replication and Fault Tolerance:**
    - Supports configurable replication strategies, such as replication across multiple data centers for disaster recovery.
    - Even if a node fails, replicated data ensures fault tolerance and data availability.
5. **Flexible Data Model:**
    - Uses a wide-column store model that supports dynamic data structures.
    - Allows for structured, semi-structured, or unstructured data storage.
6. **Low-Latency Operations:**
    - Provides high performance for both read and write operations with minimal latency.
    - Tailored for time-series data, IoT applications, and other scenarios with high-throughput requirements.

What are the advantages of using Apache Cassandra for big data applications?

The advantages of using Cassandra for big data applications include:

a. High performance and low latency for read and write operations.

b. Horizontal scalability by adding more nodes to the cluster.

c. Cost-effective and low-maintenance due to open-source nature.

d. Support for high-throughput and concurrent user access.

e. Suitable for handling vast amounts of unstructured data across distributed systems.

What are some limitations of Apache Cassandra?

While Cassandra is powerful, it has certain limitations:

a. It doesn't support joins, aggregates, or advanced querying functions natively.

b. Data migration to other databases can be complex.

c. Not ideal for transactional data requiring ACID compliance.

d. Limited built-in support for analytics and complex queries.

e. Requires careful planning and management to ensure optimal performance and data

Detailed Overview of MongoDB Architecture and Components:

1. **Drivers & Storage Engine**:
    - **Drivers**: These are client libraries that enable applications (written in languages like Python, Java, Node.js, etc.) to interact with MongoDB. They manage the communication between the application and the database, translating data between the application's format and MongoDB's **BSON** (Binary JSON) format, which is more efficient than JSON for storage and transmission.
    - **Storage Engine**: This is responsible for how data is stored on disk and directly impacts performance.
        - **MMAPv1**: Older engine using memory-mapped files. It is read-optimized but not as efficient for write-heavy workloads.
        - **WiredTiger**: Default engine (since MongoDB 3.0), offering better performance, especially for write-heavy operations, and it supports data compression.
        - **InMemory**: Stores data in RAM rather than on disk for low-latency access, but it's less suitable for large datasets since it requires a significant amount of memory.
2. **Security**:

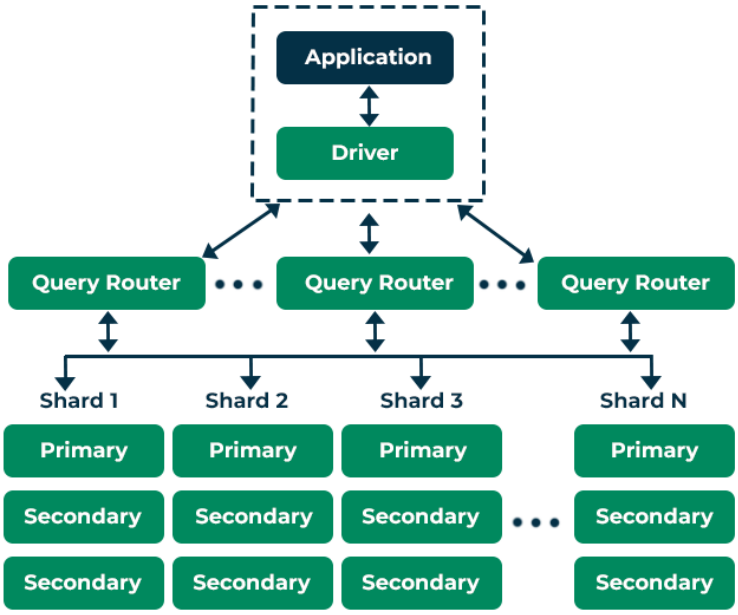
- **Authentication**: Ensures that only authorized users can access the MongoDB instance by verifying credentials (using methods like SCRAM, LDAP, or Kerberos).
- **Authorization**: Once authenticated, authorization controls which actions a user can perform (e.g., reading, writing, and administrative operations).
- **Encryption**: MongoDB supports encryption of data both at rest (when stored on disk) and in transit (during communication between clients and servers) to ensure data security.

3. **MongoDB Server**:
   - The **Mongod** process is the heart of MongoDB, managing data storage, indexing, and querying. Multiple instances of `mongod` can work together in a **cluster** to distribute the database across different servers.
   - Each **mongod instance** is responsible for handling client requests and ensuring the proper storage and retrieval of data.
4. **MongoDB Shell**:
   - A command-line interface (CLI) tool that allows direct interaction with MongoDB. The **Mongo Shell** uses JavaScript-based syntax and provides an interactive environment for managing databases, running queries, and performing administrative tasks.
5. **Data Storage in MongoDB**:
   - **Collections**: Equivalent to tables in relational databases, collections are groups of documents stored together. A database can contain multiple collections.
   - **Documents**: The individual data records in MongoDB are stored as documents, which are similar to JSON objects (key-value pairs). These documents are stored in **BSON** format, which is binary-encoded JSON.
6. **Indexes**:
   - **Single-field Indexes**: Indexes created on a single field to improve query performance (e.g., for searching or sorting by that field).

- **Compound Indexes**: Indexes created on multiple fields (e.g., `db.students.createIndex({item: 1, stock: 1})`) to improve queries that filter or sort by those fields together.
- **Multi-Key Indexes**: Automatically created when indexing fields that contain arrays, allowing MongoDB to create an index entry for each element in the array.
- **Geo Spatial Indexes**: Specialized indexes for geospatial data:
  - **2d Indexes**: For queries involving flat, 2D geometry.
  - **2dsphere Indexes**: For queries involving spherical (earth-like) geometry, useful for location-based queries like proximity searches.
- **Hashed Indexes**: Used in sharded clusters to ensure an even distribution of data across different nodes, based on the hash of the indexed field.

7. **Replication**:
   - MongoDB supports **replication** to provide high availability. Data is duplicated across multiple nodes in a **replica set**.
     - **Primary Node**: The primary node accepts all write operations. It is the source of truth for the data.
     - **Secondary Nodes**: These nodes replicate data from the primary and handle read requests, providing redundancy and distributing read workload.
     - **Automatic Failover**: If the primary node fails, one of the secondaries is automatically promoted to primary to continue operations without downtime.
8. **Sharding**:
   - **Sharding** is MongoDB's method of horizontal scaling. It splits large datasets into smaller, manageable chunks, called **shards**, and distributes them across multiple servers.
     - Each shard holds a subset of the data, and the dataset is divided based on a **shard key** (a field used to distribute data).

- Shards can be stored across different machines to balance load and improve performance.
- **Sharding Strategies**: MongoDB supports **range-based** sharding, where data is divided into ranges based on the shard key values (e.g., dates or numeric ranges).
- Sharding allows MongoDB to handle large datasets efficiently, distributing both storage and query load.



NoSQL Database Development Tools

NoSQL databases are designed for handling large volumes of unstructured and semi-structured data, and they offer various tools to manage, process, and analyze data effectively. These tools support tasks like data storage, querying, and analysis. Some of the key development tools associated with NoSQL databases include **MapReduce** and **Hive**.

- **MapReduce**: This is a programming model used for processing and generating large datasets that can be parallelized across a distributed cluster. It's

commonly used with NoSQL databases like MongoDB and Cassandra for distributed data processing.

- **Hive**: Hive is a data warehouse software built on top of Hadoop. It allows querying and managing large datasets stored in Hadoop's distributed storage. Hive uses a SQL-like language for querying, known as **HiveQL**.

Other NoSQL Database Development Tools

Besides **MapReduce** and **Hive**, NoSQL databases offer various other tools and technologies to enhance functionality and facilitate development. Here are some notable ones:

a. Cassandra Query Language (CQL):

- **CQL** is the query language used in **Apache Cassandra**, a distributed NoSQL database. It's similar to SQL but designed for Cassandra's distributed nature. It allows users to perform CRUD (Create, Read, Update, Delete) operations and schema management tasks within Cassandra.

b. MongoDB Tools:

- **MongoDB Compass**: A graphical user interface for MongoDB that provides data visualization and query-building capabilities. It allows developers to explore the data stored in MongoDB, build complex queries visually, and optimize performance.
- **Mongoshell**: The MongoDB shell is an interactive JavaScript interface for managing MongoDB databases. It allows you to run queries, perform administrative tasks, and automate operations.
- **MongoDB Atlas**: A fully managed cloud service for deploying and managing MongoDB databases, which automates tasks such as backups, scaling, and monitoring.

c. Redis Tools:

- **Redis Insight**: A GUI tool for managing and exploring Redis data. It helps visualize key-value data and run Redis commands interactively.

d. Elasticsearch:

- **Kibana**: A visualization tool for **Elasticsearch**, which is used for searching and analyzing large volumes of text data. Kibana provides real-time insights and visualizations through dashboards, making it an essential tool for monitoring and data analysis.

NoSQL Database Development Tool: MapReduce

MapReduce is a distributed data processing tool primarily used for processing and generating large datasets. It is part of the Hadoop ecosystem, but it can be used with NoSQL databases like MongoDB and Cassandra for parallel processing. Here's how **MapReduce** works:

Working of MapReduce:

MapReduce works in two stages: **Map** and **Reduce**.

1. **Map Stage**:
    - The map function processes input data (often in key-value pairs) and produces intermediate key-value pairs.
    - For example, in a MongoDB collection of documents, the map function might take a collection of documents, extract specific data, and emit key-value pairs for processing.
2. **Shuffle & Sort Stage**:
    - The system groups all intermediate key-value pairs generated by the map function.
    - It sorts the data based on keys so that all values associated with the same key are grouped together.
3. **Reduce Stage**:
    - The reduce function processes the grouped data and generates final output.
    - For example, it might sum up values associated with the same key or perform other aggregation tasks.

Advantages:

- **Scalability**: MapReduce can process vast amounts of data in parallel across many machines, making it highly scalable.
- **Fault Tolerance**: It is fault-tolerant, meaning that if one node in the cluster fails, another node can take over the task.
- **Flexibility**: It can be used with a wide range of data sources, making it ideal for processing unstructured or semi-structured data.

Use Cases of MapReduce:

- **Data Aggregation**: For example, counting the frequency of words in a large dataset.
- **Data Transformation**: Converting one format of data into another (e.g., extracting structured data from logs).
- **Distributed Data Processing**: Running computations on large datasets stored in a distributed system like Hadoop.

Hive and Its Architecture

Hive is a data warehousing tool built on top of **Apache Hadoop** for managing and querying large datasets. It provides a **SQL-like interface** (HiveQL) for querying data stored in **HDFS** (Hadoop Distributed File System). Hive is primarily used to summarize, query, and analyze large datasets, especially when a user is familiar with SQL.

The major components of Hive and its interaction with the Hadoop is demonstrated in the figure below and all the components are described further:

- **User Interface (UI) –** As the name describes User interface provide an interface between user and hive. It enables user to submit queries and other operations to the system. Hive web UI, Hive command line, and Hive HD Insight (In windows server) are supported by the user interface.
- **Hive Server** – It is referred to as Apache Thrift Server. It accepts the request from different clients and provides it to Hive Driver.
- **Driver –** Queries of the user after the interface are received by the driver within the Hive. Concept of session handles is implemented by driver.

Execution and Fetching of APIs modelled on JDBC/ODBC interfaces is provided by the user.

- **Compiler –** Queries are parses, semantic analysis on the different query blocks and query expression is done by the compiler. Execution plan with the help of the table in the database and partition metadata observed from the metastore are generated by the compiler eventually.
- **Metastore –** All the structured data or information of the different tables and partition in the warehouse containing attributes and attributes level information are stored in the metastore. Sequences or de-sequences necessary to read and write data and the corresponding HDFS files where the data is stored. Hive selects corresponding database servers to stock the schema or Metadata of databases, tables, attributes in a table, data types of databases, and HDFS mapping.
- **Execution Engine –** Execution of the execution plan made by the compiler is performed in the execution engine. The plan is a DAG of stages. The dependencies within the various stages of the plan is managed by execution engine as well as it executes these stages on the suitable system components.

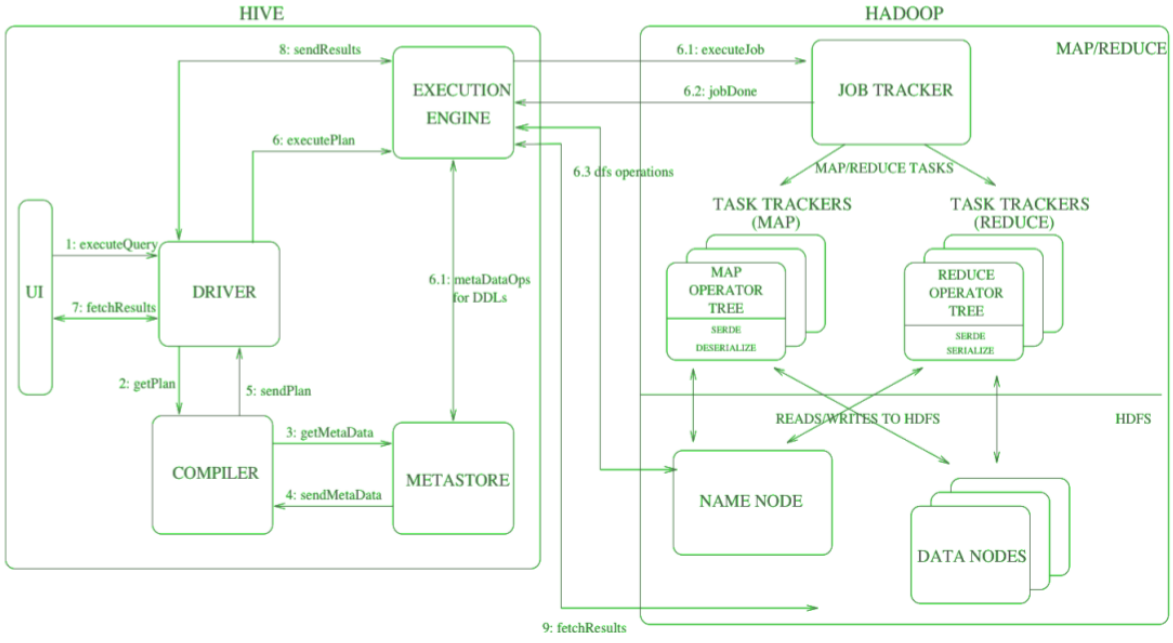


Diagram – Architecture of Hive that is built on the top of Hadoop

In the above diagram along with architecture, *job execution flow in Hive with Hadoop is demonstrated step by step*.

- **Step-1: Execute Query –** Interface of the Hive such as Command Line or Web user interface delivers query to the driver to execute. In this, UI calls the execute interface to the driver such as ODBC or JDBC.
- **Step-2: Get Plan –** Driver designs a session handle for the query and transfer the query to the compiler to make execution plan. In other words, driver interacts with the compiler.
- **Step-3: Get Metadata –** In this, the compiler transfers the metadata request to any database and the compiler gets the necessary metadata from the metastore.
- **Step-4: Send Metadata –** Metastore transfers metadata as an acknowledgment to the compiler.
- **Step-5: Send Plan –** Compiler communicating with driver with the execution plan made by the compiler to execute the query.
- **Step-6: Execute Plan –** Execute plan is sent to the execution engine by the driver.
    - Execute Job
    - Job Done
    - Dfs operation (Metadata Operation)
- **Step-7: Fetch Results –** Fetching results from the driver to the user interface (UI).
- **Step-8: Send Results –** Result is transferred to the execution engine from the driver. Sending results to Execution engine. When the result is retrieved from data nodes to the execution engine, it returns the result to the driver and to user interface (UI).

Advantages of Hive Architecture:

Scalability: Hive is a distributed system that can easily scale to handle large volumes of data by adding more nodes to the cluster.

Data Accessibility: Hive allows users to access data stored in Hadoop without the need for complex programming skills. SQL-like language is used for queries and HiveQL is based on SQL syntax.

Data Integration: Hive integrates easily with other tools and systems in the Hadoop ecosystem such as Pig, HBase, and MapReduce.

Flexibility: Hive can handle both structured and unstructured data, and supports various data formats including CSV, JSON, and Parquet.

Security: Hive provides security features such as authentication, authorization, and encryption to ensure data privacy.

Disadvantages of Hive Architecture:

High Latency: Hive's performance is slower compared to traditional databases because of the overhead of running queries in a distributed system.

Limited Real-time Processing: Hive is not ideal for real-time data processing as it is designed for batch processing.

Complexity: Hive is complex to set up and requires a high level of expertise in Hadoop, SQL, and data warehousing concepts.

Lack of Full SQL Support: HiveQL does not support all SQL operations, such as transactions and indexes, which may limit the usefulness of the tool for certain applications.

Debugging Difficulties: Debugging Hive queries can be difficult as the queries are executed across a distributed system, and errors may occur in different nodes.

Comparison between MapReduce and Hive

Feature	MapReduce	Hive
Purpose	General-purpose parallel data processing.	Data warehousing and querying in Hadoop.
Language	Requires writing custom MapReduce code (Java, Python, etc.).	Uses SQL-like language, HiveQL.
Complexity	Requires understanding of distributed programming and	Easier for SQL users to learn and use.

	MapReduce APIs.	
Data Processing	Ideal for complex and custom data processing tasks.	Best suited for structured or semi-structured data querying and batch processing.
Use Cases	Large-scale, custom data processing.	Data summarization, reporting, and analytics.
Scalability	Highly scalable in parallel across a cluster.	Scales using Hadoop's distributed architecture.
Output	Custom outputs based on the MapReduce job.	Output can be directly queried and analyzed.

Programming Languages (XML/JSON)

XML (eXtensible Markup Language) and **JSON (JavaScript Object Notation)** are two widely used technologies for representing and exchanging data. Both serve to structure data, making it easier to transmit and share between different systems and applications. They play a crucial role in NoSQL databases and other modern applications that handle large-scale data, particularly in distributed systems.

1. What is XML? Why is XML Important? What are the Benefits of Using XML?

What is XML?

- **XML (eXtensible Markup Language)** is a markup language used to define rules for encoding documents in a format that is both human-readable and machine-readable. Unlike HTML, which is used to define the structure of web pages, XML is designed to store and transport data.
- XML documents are made up of elements enclosed in tags, allowing users to define their own tags based on the data being represented. It is primarily used for representing structured data, including metadata, configuration files, and in communication between web services.

Why is XML Important?

- **Interoperability**: XML allows data to be shared between different systems regardless of their underlying architecture. This is particularly important in web services, where data needs to be exchanged between different platforms (e.g., between a web server and a database).
- **Self-Descriptive**: The tags in XML provide context to the data, which makes it easy to understand. This feature is essential when data is passed between systems or applications that might not share the same schema.
- **Platform Independent**: XML can be used across different systems and platforms. It is not tied to any specific programming language or technology.
- **Extensibility**: Users can create their own tags to suit their specific data needs, providing flexibility in data representation.

Benefits of Using XML:

- **Standardized Data Representation**: XML allows the structured representation of data that can be read and processed by both humans and machines.
- **Self-Describing**: XML tags provide context, making it easier for different systems to understand the data.
- **Wide Adoption**: XML is supported by virtually all major programming languages, making it highly compatible for data exchange.
- **Hierarchical Structure**: XML allows data to be represented in a tree-like structure, which makes it suitable for representing complex data relationships.
- **Validation**: With **XML Schema** or **DTD (Document Type Definition)**, XML documents can be validated to ensure that they conform to a specific structure, providing data integrity.
- **Data Transport**: XML is often used in web services (e.g., SOAP) and APIs for transmitting data across the internet.

---

2. What is JSON? Explain Features and Data Types of JSON

What is JSON?

- **JSON (JavaScript Object Notation)** is a lightweight, text-based format used for representing structured data, typically in the form of key-value pairs. JSON

is easy to read and write and is commonly used for transmitting data between a server and web application.

- JSON is particularly popular in **web development** due to its simplicity and its close alignment with JavaScript, the language of the web. Many web APIs and NoSQL databases like MongoDB use JSON for data exchange and storage.

Features of JSON:

- **Lightweight**: JSON has a minimalistic format, making it efficient for both data storage and transmission over networks.
- **Human-Readable**: JSON's syntax is simple and easy to read and write, making it accessible for both developers and non-developers.
- **Language-Agnostic**: JSON is text-based and independent of any programming language, though it is native to JavaScript. It is supported by most programming languages, including Python, Java, C#, and Ruby.
- **Easy to Parse**: JSON data is easily parsed into native data structures (e.g., objects, arrays) by most programming languages, especially JavaScript.
- **Data Interchange**: JSON is commonly used for data interchange between client-side and server-side applications. It is especially useful in web APIs, mobile applications, and microservices architecture.

Data Types in JSON:

JSON supports the following data types:

1. **String**: A sequence of characters, enclosed in double quotes.
   - Example: `"name": "John"`
2. **Number**: Numeric values, which can be integers or floating-point numbers.
   - Example: `"age": 30`, `"temperature": 98.6`
3. **Boolean**: Represents either `true` or `false`.
   - Example: `"isActive": true`
4. **Array**: An ordered collection of values, enclosed in square brackets. The values can be of any data type, including other arrays or objects.

- Example: `"colors": ["red", "green", "blue"]`

5. **Object**: A collection of key-value pairs, where the key is a string and the value can be any data type. Objects are enclosed in curly braces.
   - Example: `"person": {"name": "John", "age": 30}`
6. **Null**: Represents an empty or null value.
   - Example: `"address": null`

Advantages of JSON:

- **Simplicity**: JSON's simple structure makes it easier to understand and use.
- **Integration with JavaScript**: Since JSON is based on JavaScript object syntax, it integrates seamlessly with web applications and APIs that use JavaScript.
- **Lightweight**: JSON's compact format reduces the overhead of transmitting data over the network.
- **Support for Complex Data**: JSON allows for nested objects and arrays, which makes it suitable for representing hierarchical data.

---

3. Explain Schemas in XML

In XML, **schema** refers to the set of rules that define the structure, content, and constraints of XML documents. XML schemas provide a way to validate XML documents, ensuring that they follow a predefined structure and meet certain data requirements.

XML Schema Definition (XSD):

- An **XML Schema (XSD)** defines the structure of an XML document by specifying elements, attributes, and their data types. It is used to ensure that the XML data adheres to the desired format and rules.
- XSD provides a formal way to validate an XML document, ensuring that the document contains the correct elements in the correct order and with the appropriate data types.

Key Components of XML Schema:

1. **Elements**: Define the data that will appear in the document.
   - Example: `<name>John</name>`
2. **Attributes**: Provide additional information about an element.
   - Example: `<person age="30">John</person>`
3. **Complex Types**: Allow elements to contain other elements or attributes.
   - Example: A `<person>` element might contain a `<name>` and an `<age>` element.
4. **Simple Types**: Define the allowable data types for an element or attribute (e.g., string, integer, date).
5. **Data Types**: Each element or attribute can have a specific data type (e.g., string, integer, date).
6. **Constraints**: Specify the rules for the data (e.g., minimum or maximum value, length of text).

Why XML Schema is Important:

- **Validation**: Ensures data integrity by validating XML documents against a defined structure.
- **Consistency**: Guarantees that XML documents follow a consistent format, making it easier to process the data across different systems.
- **Documentation**: XML schemas serve as documentation for the structure of XML data, helping developers understand how to use the XML data.

---

4. NoSQL Programming Languages: XML and JSON

NoSQL databases, such as **MongoDB**, **Cassandra**, and **Couchbase**, store data in formats like **JSON** and sometimes **XML**. These formats allow for flexible, dynamic data storage and retrieval, which is one of the main advantages of NoSQL databases.

- **JSON** is commonly used in NoSQL databases like **MongoDB**, where data is stored as **BSON (Binary JSON)**. BSON is an extension of JSON that supports additional data types and is more efficient for storage and transmission.

- **XML**, while not as common as JSON in NoSQL databases, may still be used in certain NoSQL systems like **Couchbase**, which supports storing XML documents for specific use cases such as document-oriented storage.

Why NoSQL Databases Use JSON and XML:

1. **Flexibility**: JSON and XML allow for dynamic, schema-less data storage. Data can be added, removed, or modified without affecting other records, making them ideal for evolving data models.
2. **Performance**: JSON's lightweight format enables faster data transmission, and BSON (used by MongoDB) improves performance with its binary encoding.
3. **Hierarchical Structure**: Both XML and JSON can represent nested data structures, making them suitable for complex, hierarchical data.

Key Differences Summary Table

Feature	**JSON**	**BSON**
Format	Text-based (human-readable)	Binary format (machine-readable)
Data Types	Limited to string, number, boolean, array, object, null	Includes all JSON types plus Date, Binary, ObjectId, RegEx, JavaScript
Storage	Less efficient for storage, larger size	More compact and efficient for storage
Parsing	Slower parsing, requires conversion from text	Faster parsing and processing
Use Case	Data interchange (APIs, web services)	Internal storage in MongoDB, optimized for large-scale data
Flexibility	Highly flexible with schema-less design	Flexible with support for additional complex types
Performance	Lower performance in large datasets	Optimized for high-performance, large datasets