NOSQL DATABASE

UNIT-1

Introduction

- NoSQL is a type of database management system (DBMS) that is designed to handle and store large volumes of unstructured and semi-structured data.
- The traditional relational databases that use tables with pre-defined schemas to store data.
- NoSQL databases use flexible data models that can adapt to changes in data structures and are capable of scaling horizontally to handle growing amounts of data.
- The term NoSQL originally referred to “non-SQL” or “non-relational” databases, but the term has since evolved to mean “not only SQL,” as NoSQL databases have expanded to include a wide range of different database architectures and data models.
- NoSQL databases store data differently than relational tables. NoSQL databases come in a variety of types based on their data model.
- The main types are document, key-value, wide-column, and graph. They provide flexible schemas and scale easily with large amounts of big data and high user loads.

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.
4. **Document-based:** MongoDB, use a document-based data model
5. **Key-value-based:** Redis, use a key-value data model, where data is stored as a collection of key-value pairs.
5. **Column-based:** 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:

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.

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.

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, that they can handle large amounts of data and traffic with ease.

5. **Performance:** NoSQL databases are designed to handle large amounts of data and traffic, 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:

1. **Lack of standardization:** 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, that they do not guarantee the consistency, integrity, and durability of data.
3. **Narrow focus:** NoSQL databases have a very narrow focus as it is mainly designed for storage but it provides very little functionality.
4. **Open-source:** NoSQL is an **database** open-source database. There is no reliable standard for NoSQL yet.
5. **Lack of support for complex queries:** NoSQL databases are not designed to handle complex queries, that they are not a good fit for applications that require complex data analysis or reporting.
7. **Lack of maturity:** NoSQL databases are relatively new and lack the maturity of traditional relational databases.
8. **Management challenge:** The purpose of big data tools is to make the management of a large amount of data as simple as possible.
9. **GUI is not available:** GUI mode tools to access the database are not flexibly available in the market.
10. **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

FOUR TYPES OF NOSQL DATABASES

Types of databases — NoSQL
Four major types of NoSQL databases have emerged:
1.Document databases
2. Key-value databases
3. Wide-column stores
4. Graph databases.

1. Document-oriented databases

- A document-oriented database stores data in documents similar to JSON 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.

Example:

```
{
  "_id": "12345",
  "name": "foo bar",
  "email": "foo@bar.com",
  "address": {
    "street": "123 foo street",
    "city": "some city",
    "state": "some state",
    "zip": "123456"
  },
  "hobbies": ["music", "guitar", "reading"]
}
```

2. 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.

Example:

Key: user:12345

Value: {"name": "foo bar", "email": "foo@bar.com", "designation": "software developer"}

3. Wide-column stores

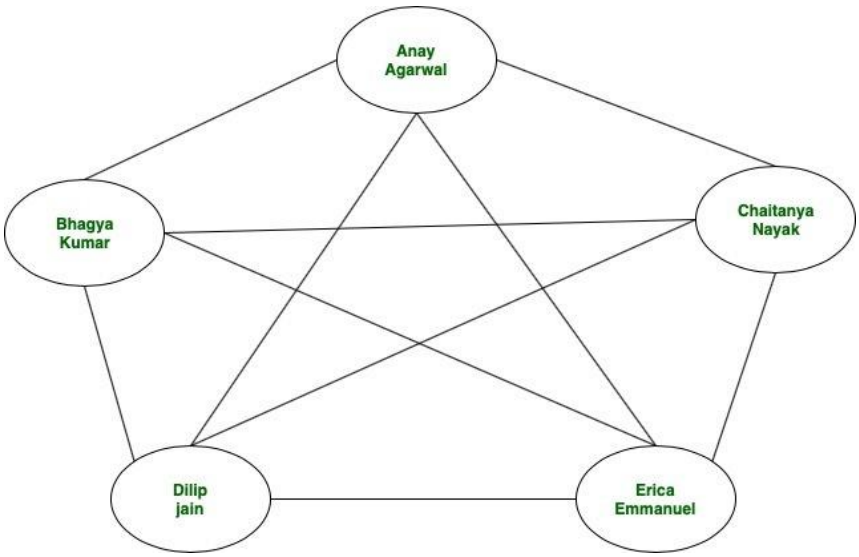
- Wide-column stores store data in tables, rows, and dynamic columns. The data is stored in tables. unlike traditional SQL databases, wide-column stores are flexible, where different rows can have different sets of columns.
- These databases can employ column compression techniques to reduce the storage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

Row A	Column 1	Column 2	Column 3
	Value	Value	Value
Row B	Column 1	Column 2	Column 3
	Value	Value	Value

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).
- 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.

Example:



NOSQL ARCHITECTURE

Architecture is a logical way of categorizing data that will be stored on the Database. NoSQL is a type of database which helps to perform operations on big data and store it in a valid format. It is widely used because of its flexibility and a wide variety of services.

Architecture Patterns of NoSQL:

The data is stored in NoSQL in any of the following four data architecture patterns.

1. Key-Value Store Database
2. Column Store Database
3. Document Database
4. Graph Database

1.Key-Value Store Database:

- This model is one of the most basic models of NoSQL databases. The data is stored in form of Key-Value Pairs.
- The key is usually a sequence of strings, integers or characters but can also be a more advanced data type.
- The value is typically linked or co-related to the key. The key-value pair storage databases generally store data as a hash table where each key is unique. The value can be of any type JSON, BLOB(Binary Large Object), strings.
- This type of pattern is usually used in shopping websites or e-commerce applications.

Advantages:

- Can handle large amounts of data and heavy load,
- Easy retrieval of data by keys.

Limitations:

- Complex queries may attempt to involve multiple key-value pairs which may delay performance.
- Data can be involving many-to-many relationships which may collide.

Examples:

- DynamoDB
- Berkeley DB

Key:1	ID:210

Key:2	ID:411	Email: geeksforgeeks@gmail.com

Key:3	UID:219	Name: Geek	Age:20

2.Column Store Database:

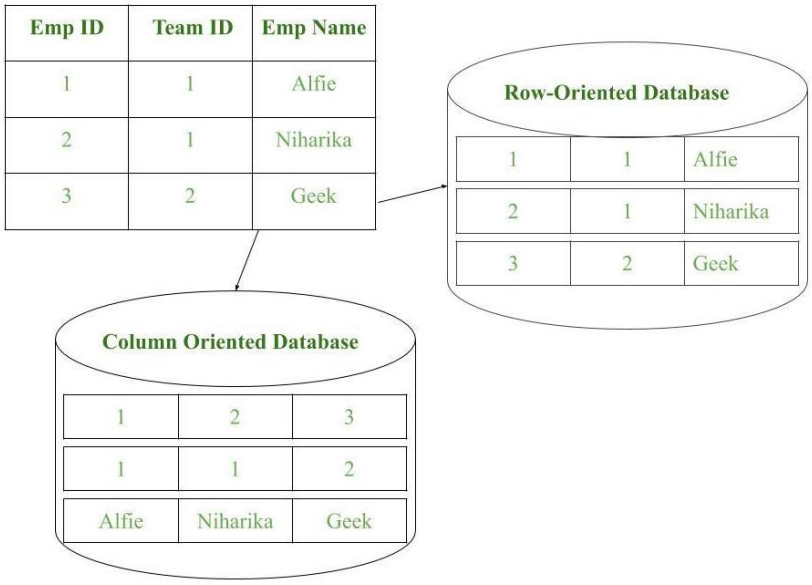
- Rather than storing data in relational tuples, the data is stored in individual cells which are further grouped into columns.
- Column-oriented databases work only on columns. They store large amounts of data into columns together.
- Format and titles of the columns can diverge from one row to other. Every column is treated separately.

Advantages:

- Data is readily available
- Queries like SUM, AVERAGE, COUNT can be easily performed on columns.

Examples:

- HBase
- Bigtable by Google
- Cassandra



3.Document Database:

- The document database fetches and accumulates data in form of key-value pairs but here, the values are called as Documents.
- Document can be stated as a complex data structure. Document here can be a form of text, arrays, strings, JSON, XML or any such format.
- The use of nested documents is also very common. It is very effective as most of the data created is usually in form of JSONs and is unstructured.

Advantages:

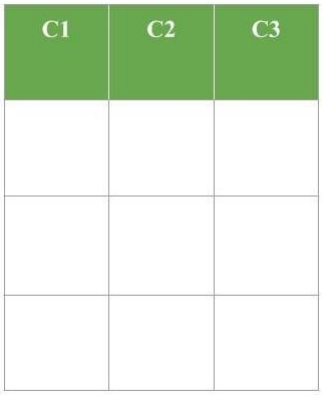
- This type of format is very useful and apt for semi-structured data.
- Storage retrieval and managing of documents is easy.

Limitations:

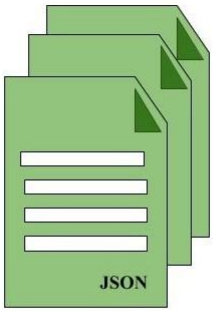
- Handling multiple documents is challenging
- Aggregation operations may not work accurately.

Examples:

- MongoDB
- CouchDB



Relational Data Model



Document Store Model

4. Graph Databases:

- This architecture pattern deals with the storage and management of data in graphs. Graphs are basically structures that depict connections between two or more objects in some data.
- The objects or entities are called as nodes and are joined together by relationships called Edges. Each edge has a unique identifier. Each node serves as a point of contact for the graph.
- This pattern is very commonly used in social networks where there are a large number of entities and each entity has one or many characteristics which are connected by edges. The relational database pattern has tables that are loosely connected, whereas graphs are often very strong and rigid in nature.

Advantages:

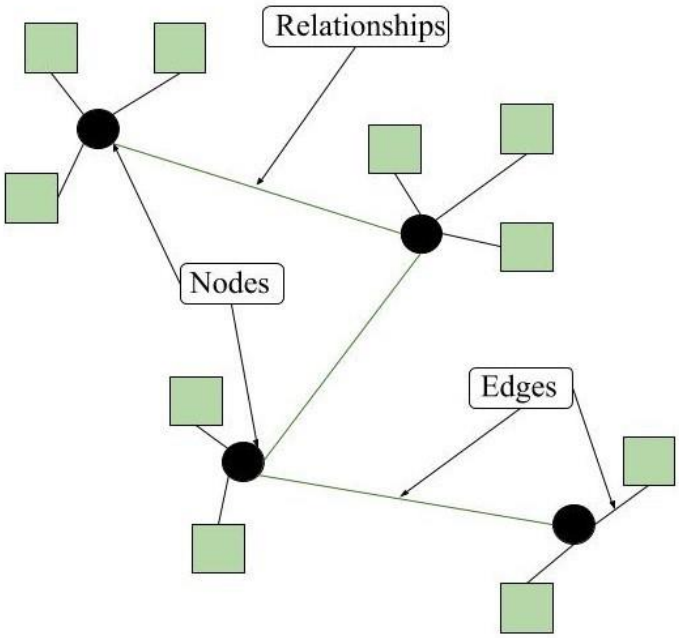
- Fastest traversal because of connections.
- Spatial data can be easily handled.

Limitations:

Wrong connections may lead to infinite loops.

Examples:

- Neo4J
- Flock DB (Used by Twitter)



DEFINE OBJECTS:

In NoSQL databases, there isn't a fixed schema (like in relational databases), so objects are defined more flexibly. Objects can be structured as key-value pairs, documents, graphs, or columns, depending on the type of NoSQL database.

- **Key-Value Store (e.g., Redis, DynamoDB)**: Objects are defined as keys associated with specific values (which can be a string, JSON object, number, etc.).
- **Document Store (e.g., MongoDB, CouchDB)**: Objects are often defined as documents, which are similar to JSON or BSON objects. Each document is typically a key-value pair where the key is a unique identifier (e.g., _id), and the value is a structured document.
- **Column Store (e.g., Cassandra, HBase)**: Objects are defined as rows with a set of columns that can store diverse data types.
- **Graph Database (e.g., Neo4j, ArangoDB)**: Objects are defined as nodes, edges, and properties that represent entities and their relationships.

Example (Document Store like MongoDB):

```
{
  "_id": "12345",
  "name": "John Doe",
  "age": 30,
```

```
  "address": { "street": "123 Elm St", "city": "Exampleville" }
}
```

Loading Data:

Loading data into a NoSQL database can vary depending on the type of database you're using. Generally, you load data by inserting documents, rows, or key-value pairs.

- **Key-Value Store**: You insert key-value pairs into the database.

```
# Example in Redis
redis.set("user:12345", '{"name": "John Doe", "age": 30}')
```

- **Document Store**: You typically load documents into collections or tables.

```
// Example in MongoDB
db.users.insertOne({ name: "John Doe", age: 30, address: "123 Elm St" });
```

- **Column Store**: You insert data into rows with a column family.

```
# Example in Cassandra
session.execute("INSERT INTO users (id, name, age) VALUES (12345, 'John Doe', 30)")
```

- **Graph Database**: You add nodes and edges.

```
# Example in Neo4j
graph.run("CREATE (u:User {id: 12345, name: 'John Doe'})")
```

Querying Data:

NoSQL databases often have flexible query languages, some similar to SQL, while others have unique query methods. Here's an overview:

- **Key-Value Store**: You query based on the key. There are no complex queries like in SQL.

```
# Example in Redis
user = redis.get("user:12345")
```

- **Document Store**: Querying is typically done using a query language (e.g., MongoDB's query language or MQL).

```
// Example in MongoDB
db.users.find({ "name": "John Doe" })
```

- **Column Store**: Querying is done based on rows and column families.

```
# Example in Cassandra
rows = session.execute("SELECT * FROM users WHERE id = 12345")
```

- **Graph Database**: You query based on relationships between nodes, typically using a graph query language like Cypher (Neo4j).

```
// Example in Neo4j
MATCH (u:User {id: 12345}) RETURN u.name
```

PERFORMANCE TUNE COLUMN-ORIENTED NOSQL DATABASES

Performance tuning for column-oriented NoSQL databases involves optimizing multiple factors to ensure high throughput, low latency, and efficient resource usage.

1. **Data Modeling**

- **Schema Design**: In column-family NoSQL databases carefully design your schema to suit your query patterns.
- **Partitioning and Clustering**: Ensure that data is partitioned and clustered efficiently. Use partition keys that balance the load evenly across nodes to avoid hot spots.
- **Wide Rows**: In querying a lot of related data, store it in a "wide row" format to minimize the number of reads that rows don't grow too large to avoid performance degradation due to storage or query time.

2. **Indexing Strategies**

- **Secondary Indexes**: Use secondary indexes wisely. While they are useful, they can introduce performance bottlenecks when data grows large.
- **Materialized Views**: Consider using materialized views in Cassandra for frequently queried patterns to avoid expensive read operations.
- **Avoid Global Secondary Indexes**: In systems like Cassandra, avoid global secondary indexes (GSI) as they can become a bottleneck under heavy write loads.

3. **Write Optimization**

- **Batch Writes**: Use batch writes to reduce overhead. However, ensure that batch sizes are optimized to prevent overwhelming the system.
- **Write Consistency Levels**: Tune the consistency level of write operations based on your application's requirements.
- **Compaction and Write-Ahead Logs (WAL)**: Tune compaction settings to ensure that data is compacted in an optimal way. Also, consider adjusting write-ahead log settings to strike a balance between durability and performance.

4. **Read Optimization**

- **Read Consistency Levels**: Similar to writes, tune the read consistency levels.
- **Caching**: Leverage caching mechanisms to reduce repeated access to frequently read data. Use tools like memcached, Redis, or the built-in caching capabilities of the NoSQL database.
- **Pre-fetching Data**: If you can predict access patterns, you can pre-fetch or pre-load data into memory to avoid cold start read latency.

5. **Concurrency & Thread Tuning**

- **Thread Pools**: Adjust thread pool sizes to better utilize available hardware resources.
- **Concurrent Operations**: Tune the number of concurrent operations per node and across nodes to maximize throughput.

6. **Data Compaction & Garbage Collection**

- **Compaction Strategy**: In column-oriented NoSQL systems, data compaction is crucial. Tune the compaction strategy
- **Garbage Collection**: If you're using a JVM-based column store (like HBase or Cassandra), garbage collection can impact performance.

7. **Replication & Fault Tolerance**

- **Replication Factor**: Tune the replication factor based on your consistency requirements and fault tolerance needs.
- **Consistency vs Availability**: Balance between consistency and availability by choosing appropriate settings like QUORUM or ONE for consistency levels, and make sure the data model supports eventual consistency for scalability.

8. **Data Compression**

- **Column-Specific Compression**: Many column-oriented NoSQL databases allow you to choose different compression algorithms per column family or table.
- **Compaction and Compression Settings**: Tune the frequency of compactions, as too frequent compactions may degrade performance, while infrequent compactions may result in excessive storage consumption.

PERFORMANCE TUNE DOCUMENT ORIENTED NOSQL DATABASES

Performance tuning for document-oriented NoSQL databases is critical to ensure optimal operation as these databases handle large volumes of unstructured or semi-structured data.

1. **Schema Design**

- **Proper Data Modelling**: Design your documents to match your application's query patterns.
- **Use of Indexing**: Create indexes on fields that are frequently queried or used for sorting. Be mindful of too many indexes, as they can degrade performance during writes.
    - **Compound Indexes**: Useful for queries that filter by multiple fields.
    - **Text Indexes**: Helpful for full-text search capabilities.
- **Avoid Large Documents**: Keep documents small and manageable in size. Large documents can slow down read and write operations due to increased I/O.
- **Use of Batching**: Store and retrieve multiple documents in a single request to reduce network overhead.

2. Index Optimization

- **Limit Indexes**: Indexes are useful, but they come at a cost. Having too many indexes can slow down write operations. Focus only on essential indexes for your queries.
- **Index Maintenance**: Periodically review and drop unused indexes.
- **Sparse Indexes**: When some documents do not have the indexed field, sparse indexes can help save space and improve performance.
- **TTL Indexes**: Time-to-live (TTL) indexes are valuable for automatically deleting expired documents, reducing the overhead of manually managing data.

3. Read and Write Optimization

- **Read Concern**: In systems like MongoDB, set the appropriate read concern to control consistency. Use "eventual consistency" if strict consistency is not necessary to achieve better performance.
- **Write Concern**: Similar to read concern, write concern determines the level of acknowledgment required for write operations. Lower write concern values can speed up writes but may risk data integrity.
- **Bulk Operations**: When inserting, updating, or deleting large sets of documents, use bulk operations to minimize overhead and improve throughput.
- **Sharding**: Sharding distributes data across multiple servers, which improves scalability and performance for large datasets. Choose an appropriate shard key based on access patterns.

4. Memory and Caching

- **Memory Allocation**: Ensure sufficient memory is allocated to the database. If the working set doesn't fit into memory, performance will degrade due to disk I/O.
- **Data Caching**: Some NoSQL databases provide built-in caching . Leverage these caches effectively to avoid excessive disk I/O.

- **Cache Management**: Adjust cache size based on your application needs. Keep frequently accessed data in memory to improve response times.

5. **Concurrency and Locking**

- **Concurrency Control**: Ensure that your database's concurrency control mechanism is properly tuned. This can help prevent bottlenecks and improve performance under high load.
- **Optimize Locking**: Locking can hinder performance, especially with write-heavy workloads.

  Example: MongoDB, uses a document-level lock to minimize contention, but it's important to avoid large updates that may lock the entire document.

- **Transactions**: Use transactions judiciously. While transactions provide strong consistency guarantees, they can impact performance. Make sure to limit the scope and duration of transactions.

6. Monitoring and Profiling.

- **Query Profiling**: Use query profiling tools to identify slow queries and optimize them.
- **System Monitoring**: Keep an eye on system metrics like CPU, memory, disk I/O, and network performance to identify resource bottlenecks.
- **Slow Query Logs**: Monitor slow query logs to identify problematic queries and optimize them by adding indexes or adjusting the query logic.

7. **Data Replication and Fault Tolerance**

- **Replica Sets**: Set up replica sets for high availability. Replication helps with failover and read scaling by distributing read requests across secondaries, but ensure it's properly configured to balance consistency and performance.
- **Secondary Read Preference**: Direct read operations to secondary replicas for non-critical data to offload the primary replica.

8. **Compression and Data Storage**

- **Data Compression**: Enable compression on data to reduce storage overhead. This is especially important for large datasets in document databases.
- **Data Compaction**: Periodically compact data to reclaim disk space.

9**. Handling High Write Throughput**

- **Write-Heavy Workloads**: For applications with heavy write loads, consider adjusting the write concern, using batch processing, and taking advantage of background writes where possible.
- **Avoid Hotspots in Sharding**: Ensure that data is evenly distributed across shards to avoid situations where one shard gets overloaded with writes.

10. **Optimize Networking**

- **Connection Pooling**: Use connection pools to avoid the overhead of establishing new connections repeatedly. Most NoSQL databases support connection pooling out of the box.
- **Replication Lag**: Monitor replication lag in replica sets, especially when using secondaries for read scaling. High replication lag can cause stale reads.

Example: MongoDB

- **Indexing**: Use compound indexes to support queries that filter by multiple fields.
- **Sharding**: Choose a shard key that distributes documents evenly to avoid unbalanced shards.
- **Aggregation Pipeline**: Optimize aggregation pipelines to reduce the number of stages and unnecessary computations.

UNIT – II

- Comparison of relational databases to new NoSQL stores
- MongoDB,
- Cassandra,
- HBASE,
- Neo4j use and deployment,
- Application,
- RDBMS approach,
- Challenges NoSQL approach,
- Key-Value and Document Data Models,
- Column-Family Stores,
- Aggregate-Oriented Databases

I.

COMPARISON OF RELATIONAL DATABASES TO NEW NOSQL STORES :

- **Relational Database :**
  RDBMS stands for Relational Database Management Systems.
- It is most popular database. In it, data is store in the form of row that is in the form of tuple.
- It contain numbers of table and data can be easily accessed because data is store in the table. This Model was proposed by E.F. Codd.
- **NoSQL :**
  NoSQL Database stands for a non-SQL database.
- NoSQL database doesn't use table to store the data like relational database.
- It is used for storing and fetching the data in database and generally used to store the large amount of data.
- It supports query language and provides better performance.

Difference between Relational database and NoSQL :

Relational Database	NoSQL
It is used to handle data coming in low velocity.	**It is used to handle data coming in high velocity.**
It gives only read scalability.	**It gives both read and write scalability.**
It manages structured data.	**It manages all type of data.**
Data arrives from one or few locations.	**Data arrives from many locations.**
It supports complex transactions.	**It supports simple transactions.**
It has single point of failure.	**No single point of failure.**
It handles data in less volume.	**It handles data in high volume.**
Transactions written in one location.	**Transactions written in many locations.**
support ACID properties compliance	**doesn't support ACID properties**
Its difficult to make changes in database once it is defined	**Enables easy and frequent changes to database**

schema is mandatory to store the data	**schema design is not required**
Deployed in vertical fashion.	**Deployed in Horizontal fashion.**

II.MONGODB:

MongoDB stands out as a leading **NoSQL** database, offering an open-source, document-oriented approach that diverges from traditional relational databases. Unlike **SQL** databases, MongoDB stores data in BSON format, akin to JSON, allowing for more flexible data storage and retrieval. In this article, We will get a in **detailed** knowledge about **MongoDB.**

What is MongoDB?

- **MongoDB** the most popular **NoSQL** database, is an **open-source document-oriented** database. The term 'NoSQL' means '**non-relational**'.
- It means that MongoDB isn't based on the **table-like relational database** structure but provides an altogether different mechanism for the **storage** and **retrieval** of data. This format of storage is called **BSON** (similar to **JSON** format).
- **A simple MongoDB document Structure:**

```
{
  title: 'Geeksforgeeks',
  by: 'Harshit Gupta',
  url: 'https://www.geeksforgeeks.org',
  type: 'NoSQL'
}
```

- SQL databases store data in tabular format. This data is stored in a predefined data model which is not very much flexible for today's real-world highly growing applications.
- Modern applications are more networked, social and interactive than ever. Applications are storing more and more data and are accessing it at higher rates.
- Relational Database Management System(RDBMS) is not the correct choice when it comes to handling big data by the virtue of their design since they are not horizontally scalable. If the database runs on a single server, then it will reach a scaling limit.

- NoSQL databases are more scalable and provide superior performance. MongoDB is such a NoSQL database that scales by adding more and more servers and increases productivity with its flexible document model.

MongoDB database features

- **Document Oriented**: MongoDB stores the main subject in the minimal number of documents and not by breaking it up into multiple relational structures like RDBMS. For example, it stores all the information of a computer in a single document called Computer and not in distinct relational structures like **CPU, RAM, Hard disk** etc.
- **Indexing**: Without **indexing**, a database would have to scan every document of a collection to select those that match the query which would be inefficient. So, for efficient searching Indexing is a must and MongoDB uses it to process huge volumes of data in very less time.
- **Scalability**: MongoDB scales **horizontally** using sharding (partitioning data across various **servers**). 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. Also, new machines can be added to a running database.
- **Replication and High Availability**: MongoDB increases the **data availability** with multiple copies of data on different servers. By providing redundancy, it protects the database from hardware failures. If one server goes down, the data can be retrieved easily from other active servers which also had the data stored on them.
- **Aggregation**: Aggregation operations process data records and return the computed results. It is similar to the **GROUPBY** clause in SQL. A few aggregation expressions are **sum, avg, min, max**, etc

MongoDB Uses:

MongoDB is preferred over RDBMS in the following scenarios:

- **Big Data**: If we have huge amount of data to be stored in tables, think of MongoDB before RDBMS databases. MongoDB has built-in solution for partitioning and sharding our database.
- **Unstable Schema**: **Adding a new column** in RDBMS is hard whereas MongoDB is **schema-less**. Adding a new field does not effect old documents and will be very easy.
- **Distributed data** Since multiple copies of data are stored across different servers, recovery of data is instant and safe even if there is a hardware failure.

III. 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.

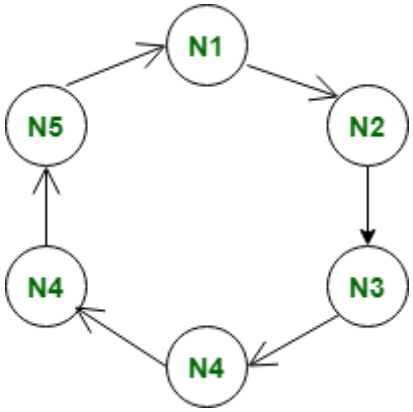


Figure-1: Masterless ring architecture of Cassandra

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

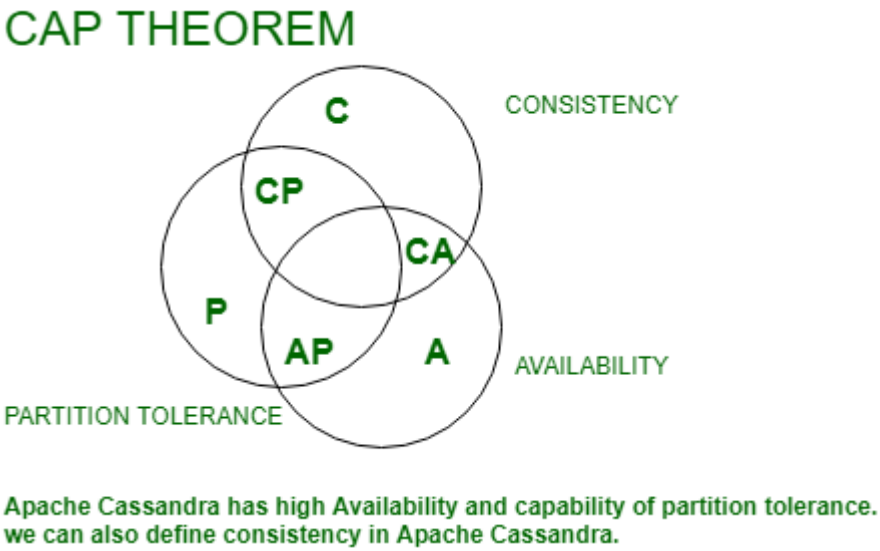


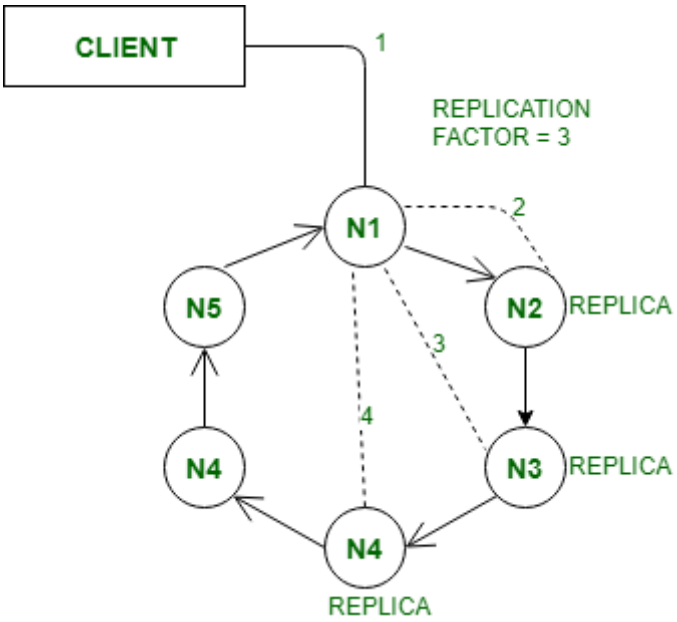
Figure-2:
CAP Theorem

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.

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.



Example :

CREATE KEYSPACE Example

WITH replication = {'class': 'NetworkTopologyStrategy',

'replication_factor': '3'};

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.com');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('Ashish Gupta', 1001, 'Bangalore', 'ashish@gmail.com');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('amit ', 1002, 'noida', 'abc@gmail.com');
```

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

```
SELECT * FROM Emp_table;
```

IV. HBASE

HBase is a data model that is similar to Google's big table.
It is an open source, distributed database developed by *Apache* software foundation written in Java.
HBase is an essential part of our Hadoop ecosystem.
HBase runs on top of HDFS (Hadoop Distributed File System).
It can store massive amounts of data from terabytes to petabytes.
It is column oriented and horizontally scalable.

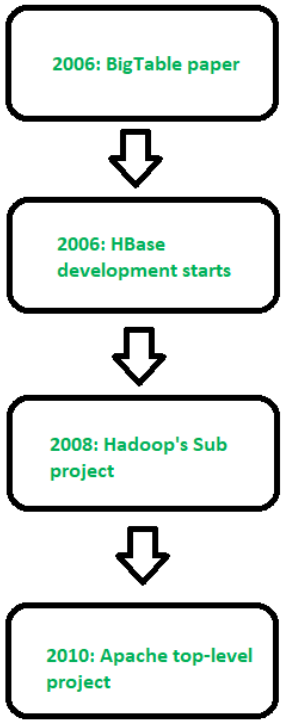


Figure – History of HBase

Applications of Apache HBase:

- **Real-time analytics:** HBase is an excellent choice for real-time analytics applications that require low-latency data access. It provides fast read and write performance and can handle large amounts of data, making it suitable for real-time data analysis.
- **Social media applications:** HBase is an ideal database for social media applications that require high scalability and performance. It can handle the large volume of data generated by social media platforms and provide real-time analytics capabilities.
- **IoT applications:** HBase can be used for Internet of Things (IoT) applications that require storing and processing large volumes of sensor data. HBase's scalable architecture and fast write performance make it a suitable choice for IoT applications that require low-latency data processing.
- **Online transaction processing:** HBase can be used as an online transaction processing (OLTP) database, providing high availability, consistency, and low-latency data access. HBase's distributed architecture and automatic failover capabilities make it a good fit for OLTP applications that require high availability.
- **Ad serving and clickstream analysis:** HBase can be used to store and process large volumes of clickstream data for ad serving and clickstream analysis. HBase's column-oriented data storage and indexing capabilities make it a good fit for these types of applications.

Features of HBase –

- It is linearly scalable across various nodes as well as modularly scalable, as it divided across various nodes.
- HBase provides consistent read and writes.
- It provides atomic read and write means during one read or write process, all other processes are prevented from performing any read or write operations.
- It provides easy to use Java API for client access.
- It supports Thrift and REST API for non-Java front ends which supports XML, Protobuf and binary data encoding options.
- It supports a Block Cache and Bloom Filters for real-time queries and for high volume query optimization.
- HBase provides automatic failure support between Region Servers.
- It support for exporting metrics with the Hadoop metrics subsystem to files.
- It doesn't enforce relationship within your data.
- It is a platform for storing and retrieving data with random access.

RDBMS Vs HBase –

- RDBMS is mostly Row Oriented whereas HBase is Column Oriented.
- RDBMS has fixed schema but in HBase we can scale or add columns in run time also.
- RDBMS is good for structured data whereas HBase is good for semi-structured data.
- RDBMS is optimized for joins but HBase is not optimized for joins.
-

Advantages Of Apache HBase:

- **Scalability:** HBase can handle extremely large datasets that can be distributed across a cluster of machines. It is designed to scale horizontally by adding more nodes to the cluster, which allows it to handle increasingly larger amounts of data.
- **High-performance:** HBase is optimized for low-latency, high-throughput access to data. It uses a distributed architecture that allows it to process large amounts of data in parallel, which can result in faster query response times.
- **Flexible data model:** HBase's column-oriented data model allows for flexible schema design and supports sparse datasets. This can make it easier to work with data that has a variable or evolving schema.
- **Fault tolerance:** HBase is designed to be fault-tolerant by replicating data across multiple nodes in the cluster. This helps ensure that data is not lost in the event of a hardware or network failure.

Disadvantages Of Apache HBase:

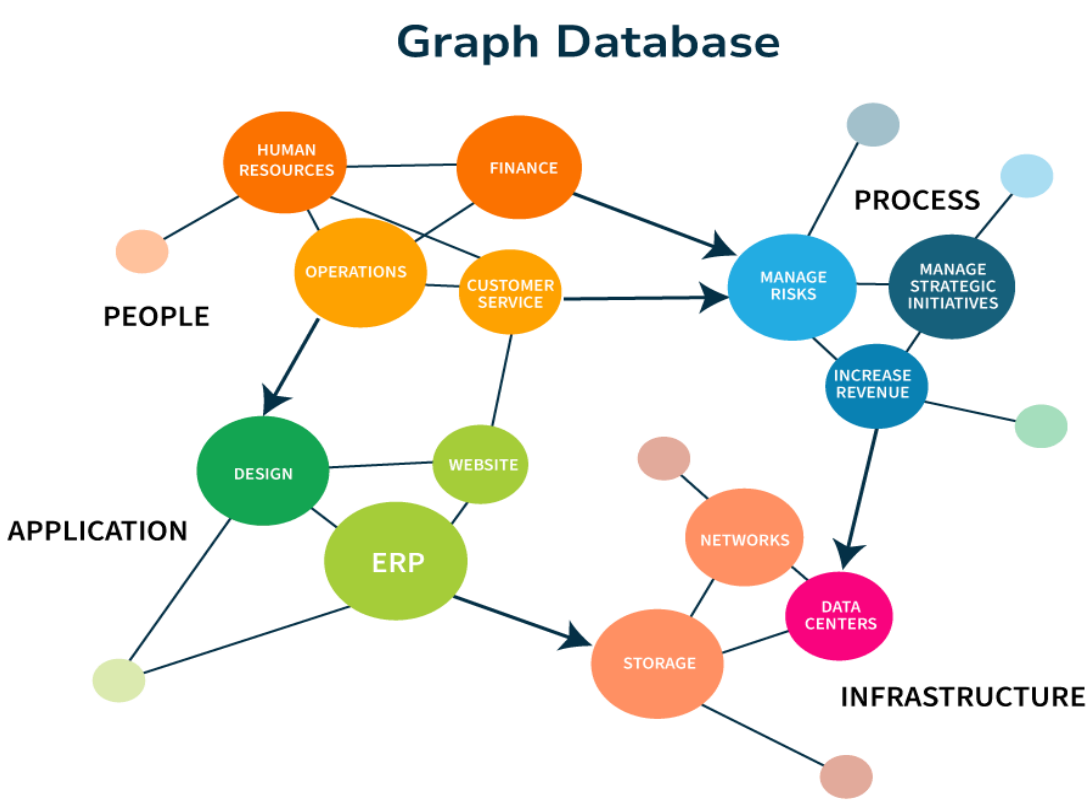
- **Complexity:** HBase can be complex to set up and manage. It requires knowledge of the Hadoop ecosystem and distributed systems concepts, which can be a steep learning curve for some users.
- **Limited query language:** HBase's query language, HBase Shell, is not as feature-rich as SQL. This can make it difficult to perform complex queries and analyses.
- **No support for transactions:** HBase does not support transactions, which can make it difficult to maintain data consistency in some use cases.
- **Not suitable for all use cases:** HBase is best suited for use cases where high throughput and low-latency access to large datasets is required. It may not be the best choice for applications that require real-time processing or strong consistency guarantees.

V.NEO4J USE AND DEPLOYMENT

- Neo4j is a powerful, high-performance, open-source graph database that enables the efficient management and querying of highly connected data. Unlike traditional relational databases, Neo4j uses graph structures to represent and store data, making it uniquely suited for applications involving complex relationships and dynamic, interconnected data.
- As the world's leading graph database, Neo4j has become essential for organizations looking to leverage the power of graph technology for a variety of use cases.
- Neo4j is a powerful and flexible graph database management system, designed to efficiently store and query highly interconnected data. Unlike traditional relational databases, which store data in tables, Neo4j uses a graph structure to represent and navigate relationships between data entities.

Neo4j structure

Neo4j stores and present the data in the form of graph not in tabular format or not in a Json format.
The whole data is represented by nodes and there you can create a relationship between nodes.
That means the whole database collection will look like a graph, that's why it is making it unique from other database management system.



Graph Database:

A graph database uses graph theory to store, map, and query relationships. It consists of nodes, edges, and properties, where:

- **Nodes** represent entities such as people, businesses, or any data item.
- **Edges** (or relationships) connect nodes and illustrate how entities are related.
- **Properties** provide additional information about nodes and relationships.

This structure allows graph databases to model real-world scenarios more naturally and intuitively than traditional relational databases.

Features of Neo4J

- **High Performance and Scalability**
- Neo4j is designed to handle massive amounts of data and complex queries quickly and efficiently. Its native graph storage and processing engine ensure high performance and scalability, even with billions of nodes and relationships.
- **Cypher Query Language**
- Neo4j uses Cypher, a powerful and expressive query language tailored for graph databases. Cypher makes it easy to create, read, update, and delete data, allowing users to perform complex queries with concise and readable syntax.
- **ACID Compliance**
- Neo4j ensures data integrity and reliability through ACID (Atomicity, Consistency, Isolation, Durability) compliance. This guarantees that all database transactions are

processed reliably and ensures the consistency of the database even in the event of failures.

- **Flexible Schema**
- Unlike traditional databases, Neo4j offers a flexible schema, allowing users to add or modify data models without downtime. This adaptability makes it ideal for evolving data structures and rapidly changing business requirements.

Neo4j Usage

- Database Management System has so many interconnecting relationships then Neo4j will be the best choice.
- **Neo4j is highly preferable to store data that contains multiple connections between nodes**.
- Neo4j is surrounded by relationships but there is no need to set up primary key or foreign key constraints to any data.
- Neo4j extremely suited for Networking data, below is the list of data areas
- used in Database Management System.
- Social network analysis like in Facebook, Twitter or in Instagram
- Network Diagram
- Fraud Detection
- Graph based searched of digital assets
- Data Management
- Real-time product recommendation

Advantages of Neo4j:

- Representation of connected data is very easy.
- Retrieval or traversal or navigation of connected data is very fast.
- It uses simple and powerful data model.
- It can represent semi-structured data is easy.

Disadvantages of Neo4j:

- OLAP support for these types of databases is not well executed.
- In this area, still there are lots of research happening around.

Neo4j Deployment:

Neo4j is a popular graph database that uses graph structures to store and query data.
Deploying Neo4j can be done in several ways, depending on your use case and environment.

1. Local Deployment (On a Single Machine)

Neo4j run locally on your machine either using its official distributions or Docker.

Using the Official Distribution:

1. **Download Neo4j:**
   - Go to the official Neo4j download page: https://neo4j.com/download/
   - Download the appropriate package for your OS (Windows, Mac, Linux).
2. **Installation:**
   - **Windows:** Run the .exe file and follow the setup instructions.

Mac: Use Homebrew:

Command: brew install neo4j

Linux (Debian-based):

wget -O - https://debian.neo4j.org/neotechnology.gpg.key | sudo apt-key add - echo "deb http://debian.neo4j.org/repo stable 4.x" | sudo tee -a /etc/apt/sources.list.d/neo4j.list sudo apt-get update sudo apt-get install neo4j

Start Neo4j:

Windows: The Neo4j desktop application will provide a user interface, or you can use the command prompt:

Command: neo4j console

Linux/Mac:

If installed via package manager, you can start with:

sudo systemctl start neo4j

Accessing Neo4j:

Open your web browser and go to http://localhost:7474/ to access the Neo4j Browser interface (default credentials: neo4j/neo4j).

Using Docker:

You can also run Neo4j inside a Docker container:

docker run --name neo4j -d -p 7474:7474 -p 7687:7687 --env NEO4J_AUTH=neo4j/neo4j neo4j:latest

After the container is running, open your browser and go to http://localhost:7474.

Backup and Monitoring

For production environments, it's essential to set up backup and monitoring systems for Neo4j.

- **Backup:**
  - Neo4j supports manual backups using the neo4j-admin tool.
  - Neo4j also supports automated backups, which can be scheduled based on your needs.
- **Monitoring:**
  - Neo4j integrates with several monitoring tools, such as **Prometheus** and **Grafana**, to collect and visualize metrics related to database performance.

Scaling Neo4j

Scaling Neo4j can be done vertically (increasing resources on a single node) or horizontally (adding more nodes).

- **Vertical Scaling:**
  - Increase CPU, RAM, and disk storage on a single server.
- **Horizontal Scaling:**
  - **Clustering** (Causal Clusters) allows you to add more nodes for load balancing, data replication, and high availability.

VI .NOSQL APPLICATION:

1. **Session Store**
   - Managing session data using relational database is very difficult.
   - The right approach is to use a global session store, which manages session information for every user who visits the site.
   - NOSQL is suitable for storing such web application session information very is large in size.
   - Since the session data is unstructured in form, it is easy to store it in schema less documents rather than in relation database record.
2. **User Profile Store**
   - To enable online transactions, user preferences, authentication of user and more, it is required to store the user profile by web and mobile application.
   - In recent time users of web and mobile application are grown very rapidly. The relational database could not handle such large volume of user profile data which growing rapidly, as it is limited to single server.

- Using NOSQL capacity can be easily increased by adding server, which makes scaling cost effective

3. **Content and Metadata Store**
   - Many companies like publication houses require a place where they can store large amount of data, which include articles, digital content and e-books, in order to merge various tools for learning in single platform
   - The applications which are content based, for such application metadata is very frequently accessed data which need less response times.
   - For building applications based on content, use of NoSQL provide flexibility in faster access to data and to store different types of contents
4. **Mobile Applications**
   - Since the smartphone users are increasing very rapidly, mobile applications face problems related to growth and volume.
   - Using NoSQL database mobile application development can be started with small size and can be easily expanded as the number of user increases, which is very difficult if you consider relational databases.
   - Since NoSQL database store the data in schema-less for the application developer can update the apps without having to do major modification in database.
   - The mobile app companies like Kobo and Playtika, uses NOSQL and serving millions of users across the world.
5. **Third-Party Data Aggregation**
   - Frequently a business require to access data produced by third party. For instance, a consumer packaged goods company may require to get sales data from stores as well as shopper's purchase history.
   - In such scenarios, NoSQL databases are suitable, since NoSQL databases can manage huge amount of data which is generating at high speed from various data sources.
6. **Internet of Things**
   - Today, billions of devices are connected to internet, such as smartphones, tablets, home appliances, systems installed in hospitals, cars and warehouses. For such devices large volume and variety of data is generated and keep on generating.
   - Relational databases are unable to store such data. The NOSQL permits organizations to expand concurrent access to data from billions of devices and systems which are connected, store huge amount of data and meet the required performance.
7. **E-Commerce**
   - E-commerce companies use NoSQL for store huge volume of data and large amount of request from user.
8. **Social Gaming**

- Data-intensive applications such as social games which can grow users to millions. Such a growth in number of users as well as amount of data requires a database system which can store such data and can be scaled to incorporate number of growing users NOSQL is suitable for such applications.
- NOSQL has been used by some of the mobile gaming companies like, electronic arts, zynga and tencent.

9. **Ad Targeting**
   - Displaying ads or offers on the current web page is a decision with direct income To determine what group of users to target, on web page where to display ads, the platforms gathers behavioral and demographic characteristics of users.
   - A NoSQL database enables ad companies to track user details and also place the very quickly and increases the probability of clicks.
   - AOL, Mediamind and PayPal are some of the ad targeting companies which uses NoSQL

VII. **RDBMS APPROACH**

- RDBMS stands for Relational Database Management Systems.
- It is a program that allows us to create, delete, and update a relational database.
- A Relational Database is a database system that stores and retrieves data in a tabular format organized in the form of rows and columns.
- RDBMS technology has continued to evolve, incorporating advancements in scalability, performance, and support for complex queries, cementing its role as a cornerstone of modern database management.

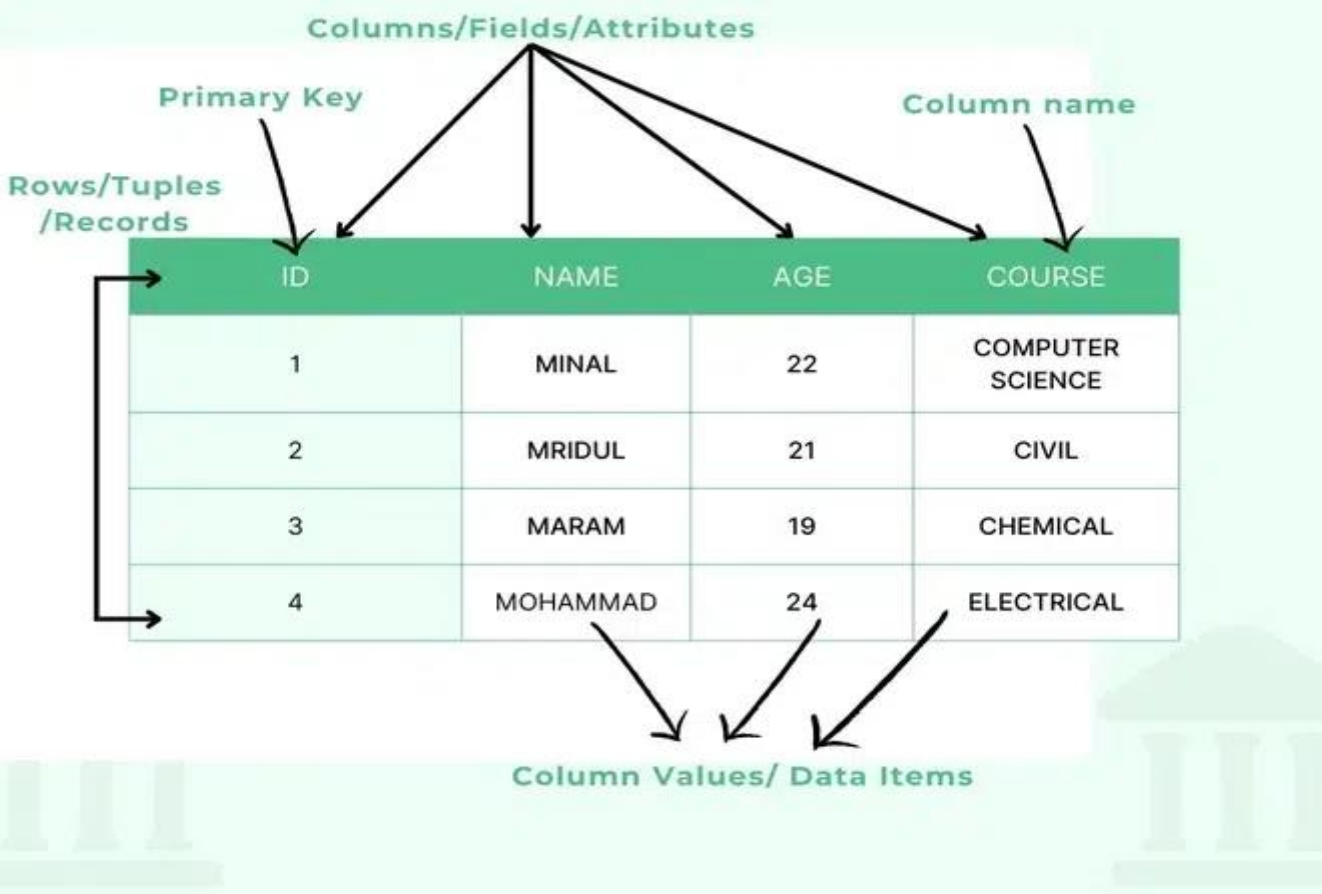
Relational Database Management Systems maintains data integrity by simulating the following features:

- **Entity Integrity:** No two records of the database table can be completely duplicate.
- **Referential Integrity:** Only the rows of those tables can be deleted which are not used by other tables. Otherwise, it may lead to data inconsistency.
- **User-defined Integrity:** Rules defined by the users based on confidentiality and access.
- **Domain integrity:** The columns of the database tables are enclosed within some structured limits, based on default values, type of data or ranges.

Database Table:

- A table is a **collection of related data** in an organized manner in the form of rows and columns.

- It is an organized arrangement of data and information in **tabular form** containing rows and columns, making it easier to understand and compare data.
- The pictorial representation of the table and its different components containing the data about different students that is ID, name, Age, and course.



Features of RDBMS

- Data must be stored in tabular form in DB file, it should be organized in the form of rows and columns.
- Each row of table is called record/tuple .
- Collection of such records is known as the cardinality of the table
- Each column of the table is called an attribute/field.
- Collection of such columns is called the arity of the table.
- No two records of the DB table can be same.
- Data duplicity is therefore avoided by using a candidate key.
- Candidate Key is a minimum set of attributes required to identify each record uniquely.
- Tables are related to each other with the help for foreign keys.
- Database tables also allow NULL values, if the values of any of the element of the table are not filled or are missing, it becomes a NULL value, which is not equivalent to zero. (NOTE: Primary key cannot have a NULL value).

Uses of RDBMS

- RDBMS is used in Customer Relationship Management.
- It is used in Online Retail Platforms.
- It is used in Hospital Management Systems.
- It is used in Business Intelligence.
- It is used in Data Warehousing

SQL Query in RDBMS

Creating a Table

Syntax:

```
CREATE TABLE table_name (
column1_name datatype constraint,
column2_name datatype constraint,
);
```

Example:

```
CREATE TABLE Employees (
EmployeeID INT PRIMARY KEY,
FirstName VARCHAR(50),
LastName VARCHAR(50),
BirthDate DATE,
Salary DECIMAL(10, 2)
);
```

2. Inserting Data into a Table

Syntax:

```
INSERT INTO table_name (column1_name, column2_name, ...)
VALUES (value1, value2, ...);
```

Example:

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, BirthDate, Salary)
VALUES (1, 'John', 'Doe', '1985-06-15', 55000.00);
```

3. Querying Data (SELECT)

Syntax:

```
SELECT column1_name, column2_name, ...
FROM table_name
WHERE condition;
```

Example:

```
SELECT FirstName, LastName, Salary
FROM Employees
WHERE Salary > 50000;
```

4. Deleting Data from a Table

Syntax:

```
DELETE FROM table_name
WHERE condition;
```

Example:

```
DELETE FROM Employees
WHERE EmployeeID = 1;
```

5. . Dropping a Table

Syntax:

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE Employees;
```

Advantages of RDBMS

- **Easy to Manage:** Each table can be independently manipulated without affecting others.
- **Security:** It is more secure consisting of multiple levels of security. Access of data shared can be limited.
- **Flexible:** Updating of data can be done at a single point without making amendments at multiple files. Databases can easily be extended to incorporate more records, thus providing greater scalability. Also, facilitates easy application of SQL queries.
- **Users:** RDBMS supports client-side architecture storing multiple users together.
- Facilitates storage and retrieval of large amount of data.
- **Easy Data Handling:**
    - Data fetching is faster because of relational architecture.
    - Data redundancy or duplicity is avoided due to keys, indexes, and normalization principles.
    - Data consistency is ensured because RDBMS is based on ACID properties for data transactions(Atomicity Consistency Isolation Durability).
- **Fault Tolerance:** Replication of databases provides simultaneous access and helps the system recover in case of disasters, such as power failures or sudden shutdowns.

Disadvantages of RDBMS

- **High Cost and Extensive Hardware and Software Support:** Huge costs and setups are required to make these systems functional.
- **Scalability:** In case of addition of more data, servers along with additional power, and memory are required.
- **Complexity:** Voluminous data creates complexity in understanding of relations and may lower down the performance.
- **Structured Limits:** The fields or columns of a relational database system is enclosed within various limits, which may lead to loss of data.

VIII.CHALLENGES NOSQL APPROACH :

- **Back-ups in Application Consistent:**
- Determining the sequence of changes across replicas, which is critical in selecting which values should compose a snapshot, is a typical difficulty in quorum-based replication systems. For example, determining a rigorous ordering between two write requests to the same database object that arrived at two distinct nodes at the same time is challenging. As a result of the absence of ordering, determining the most recent value of a database item at any given time is difficult.
- **Maturity:** NoSQL proponents would agree that increasing demands will inevitably lead to obsolescence, but the RDBMS maturity cycle

appears to be more secure. RDBMS is more reliable and functional since it has been around for decades. Many NoSQL alternatives are still in the pre-production stage and do not yet have all of their essential features implemented.

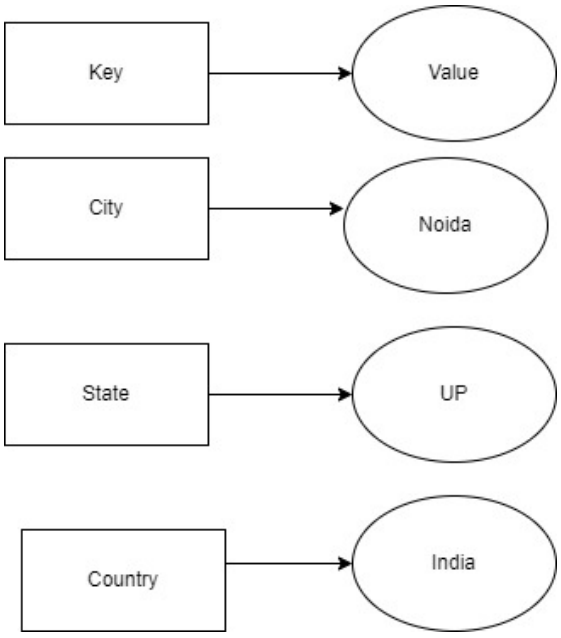
- **Database and node failures during backups and restores :** Node failures are common in NoSQL databases since they are designed to grow to hundreds of nodes. As a result, any backup method must be able to account for data collection failures from down nodes and their influence on quorum consistency. On the other hand, restorations must take into consideration failed cluster nodes and modify data re population correspondingly.
- **Analytics and business intelligence:** NoSQL was created to fulfill the needs of Web 2.0 applications, and as a result, all of its characteristics are geared toward that goal. Other commercial systems, on the other hand, necessitate moving beyond the insert-read-update-delete cycle. Even the most basic queries need extensive programming knowledge, and integrated BI tools are insufficient.
- **Performance :** Consider the same system as before in terms of performance (RPOs and RTOs). Because it is impossible to parallelize database file processing beyond a certain point, it will be impossible to achieve customer SLAs in terms of RTO if per row processing is sluggish. Due to the same parallelization limitations, recovery procedures are also impacted. This means that backup and recovery processing for 100 billion rows over 10 million files must be quick, and a solution must dramatically minimize per-row and per-file database file processing.
- **Data Integrity:** Verifying data integrity at the block level is another issue with distributed NoSQL databases. Checksum work for scale-up databases because the restored data is physically identical to the backup data. The restored data in scale-out databases is semantically comparable to the backup data, but it is not physically identical. In this situation, we'll need to come up with a unique method for identifying semantic equivalence between recovered and backup data, which will allow us to spot data corruption issues that may arise throughout the backup and restoration process.
- **Expertise:** All NoSQL developers should be considered to be in the early stages of their careers and have yet to achieve expert status. While this problem will eventually be resolved, selecting the correct developer is now difficult.
- **De-duplication:** In traditional database systems, de-duplication is simple—a block with identical bitwise values. Data copies are not always identical in modern NoSQL database storage systems because the sequencing and flushing of changes vary between nodes, creating a new challenge for de-duplication.

KEY-VALUE DATA MODEL:

- A key-value data model or database is also referred to as a key-value store.
- It is a non-relational type of database.
- In this, an associative array is used as a basic database in which an individual key is linked with just one value in a collection.
- For the values, keys are special identifiers.
- Any kind of entity can be valued.
- The collection of key-value pairs stored on separate records is called key-value databases and they do not have an already defined structure.

Working of key-value databases :

- A number of easy strings or even a complicated entity are referred to as a value that is associated with a key by a key-value database, which is utilized to monitor the entity.
- In many programming paradigms, a key-value database resembles a map object or array, or dictionary, which is controlled by a DBMS.



Usage a key-value database:

- User session attributes in an online app like finance or gaming, which is referred to as real-time random data access.
- Caching mechanism for repeatedly accessing data or key-based design.
- The application is developed on queries that are based on keys.

Features:

One of the most un-complex kinds of NoSQL data models.

- For storing, getting, and removing data, key-value databases utilize simple functions.
- Querying language is not present in key-value databases.
- Built-in redundancy makes this database more reliable.

Advantages:

- It is very easy to use. Due to the simplicity of the database, data can accept any kind, or even different kinds when required.
- Its response time is fast due to its simplicity, given that the remaining environment near it is very much constructed and improved.
- Key-value store databases are scalable vertically as well as horizontally.
- Built-in redundancy makes this database more reliable.

Disadvantages:

- As querying language is not present in key-value databases, transportation of queries from one database to a different database cannot be done.
- The key-value store database is not refined.

Examples of key-value databases:

- **Couchbase:** It permits SQL-style querying and searching for text.
- **Amazon DynamoDB:** The key-value database which is mostly used is Amazon DynamoDB as it is a trusted database used by a large number of users. It can easily handle a large number of requests every day and it also provides various security options.
- **Riak:** It is the database used to develop applications.
- **Aerospike:** It is an open-source and real-time database working with billions of exchanges.
- **Berkeley DB:** It is a high-performance and open-source database providing scalability.

DOCUMENT DATA MODEL:

- A Document Data Model is a lot different than other data models because it stores data in JSON, BSON, or XML documents.
- 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.

Example:

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

Working of Document Data Model:

- Data model works as a semi-structured data model in which the records and data associated with them are stored in a single document i.e data model is not completely unstructured.
- The main thing is data is stored in a document.

Features:

- **Document Type Model:** As we all know data is stored in documents rather than tables or graphs, so it becomes easy to map things in many programming languages.
- **Flexible Schema:** Overall schema is very much flexible to support this statement one must know that not all documents in a collection need to have the same fields.
- **Distributed and Resilient:** Document data models are very much dispersed which is the reason behind horizontal scaling and distribution of data.
- **Manageable Query Language:** These data models are the ones in which query language allows the developers to perform CRUD (Create Read Update Destroy) operations on the data model.

Examples of Document Data Models :

- Amazon DocumentDB
- MongoDB
- Cosmos DB
- ArangoDB
- Couchbase Server
- CouchDB

Advantages:

- **Schema-less:** These are very good in retaining existing data at massive volumes because there are absolutely no restrictions in the format and the structure of data storage.

- **Faster creation of document and maintenance:** It is very simple to create a document and apart from this maintenance requires is almost nothing.
- **Open formats:** It has a very simple build process that uses XML, JSON, and its other forms.
- **Built-in versioning:** It has built-in versioning which means as the documents grow in size there might be a chance they can grow in complexity. Versioning decreases conflicts.

Disadvantages:

- **Weak Atomicity:** It lacks in supporting multi-document ACID transactions. A change in the document data model involving two collections will require us to run two separate queries i.e. one for each collection. This is where it breaks atomicity requirements.
- **Consistency Check Limitations:** One can search the collections and documents that are not connected to an author collection but doing this might create a problem in the performance of database performance.
- **Security:** Nowadays many web applications lack security which in turn results in the leakage of sensitive data. So it becomes a point of concern, one must pay attention to web app vulnerabilities.

Applications of Document Data Model :

- **Content Management: D**ata models are very much used in creating various video streaming platforms, blogs, and similar services Because each is stored as a single document and the database here is much easier to maintain as the service evolves over time.
- **Book Database:** These are very much useful in making book databases because as we know this data model lets us nest.
- **Catalog:** When it comes to storing and reading catalog files these data models are very much used because it has a fast reading ability if incase Catalogs have thousands of attributes stored.
- **Analytics Platform:** These data models are very much used in the Analytics Platform.

IX.COLUMN-FAMILY STORES

The relational database stores data in rows also reads the data row by row, column store is organized as a set of columns.
Columns are of the same type and gain from more efficient compression, which makes reads faster than before.
Examples of Columnar Data Model: Cassandra and Apache Hadoop Hbase.

Working of Columnar Data Model:

In Columnar Data Model instead of organizing information into rows, it does in columns.
Data model is more flexible because it is a type of NoSQL database.
Example:

Row-Oriented Table:

S.No.	Name	Course	Branch	ID
01.	Tanmay	B-Tech	Computer	2
02.	Abhishek	B-Tech	Electronics	5
03.	Samriddha	B-Tech	IT	7
04.	Aditi	B-Tech	E & TC	8

S.No.	Branch	ID
01.	Computer	2
02.	Electronics	5
03.	IT	7
04.	E & TC	8

Column – Oriented Table:

S.No.	Name	ID
01.	Tanmay	2
02.	Abhishek	5
03.	Samriddha	7
04.	Aditi	8

S.No.	Course	ID
01.	B-Tech	2
02.	B-Tech	5
03.	B-Tech	7
04.	B-Tech	8

Columnar Data Model uses the concept of keyspace, which is like a schema in relational models.

Advantages of Columnar Data Model :

- **Well structured:** Since these data models are good at compression so these are very structured or well organized in terms of storage.
- **Flexibility:** A large amount of flexibility as it is not necessary for the columns to look like each other, which means one can add new and different columns without disrupting the whole database
- **Aggregation queries are fast:** The most important thing is aggregation queries are quite fast because a majority of the information is stored in a column. An example would be Adding up the total number of students enrolled in one year.
- **Scalability:** It can be spread across large clusters of machines, even numbering in thousands.
- **Load Times:** Since one can easily load a row table in a few seconds so load times are nearly excellent.

Disadvantages of Columnar Data Model:

- **Designing indexing Schema:** To design an effective and working schema is too difficult and very time-consuming.
- **Suboptimal data loading:** incremental data loading is suboptimal and must be avoided, but this might not be an issue for some users.
- **Security vulnerabilities:** If security is one of the priorities then it must be known that the Columnar data model lacks inbuilt security features in this case, one must look into relational databases.
- **Online Transaction Processing (OLTP):** Online Transaction Processing (OLTP) applications are also not compatible with columnar data models because of the way data is stored.

Applications of Columnar Data Model:

- Columnar Data Model is very much used in various Blogging Platforms.
- It is used in Content management systems like WordPress, Joomla, etc.
- It is used in Systems that maintain counters.
- It is used in Systems that require heavy write requests.
- It is used in Services that have expiring usage.

X.AGGREGATE-ORIENTED DATABASES :

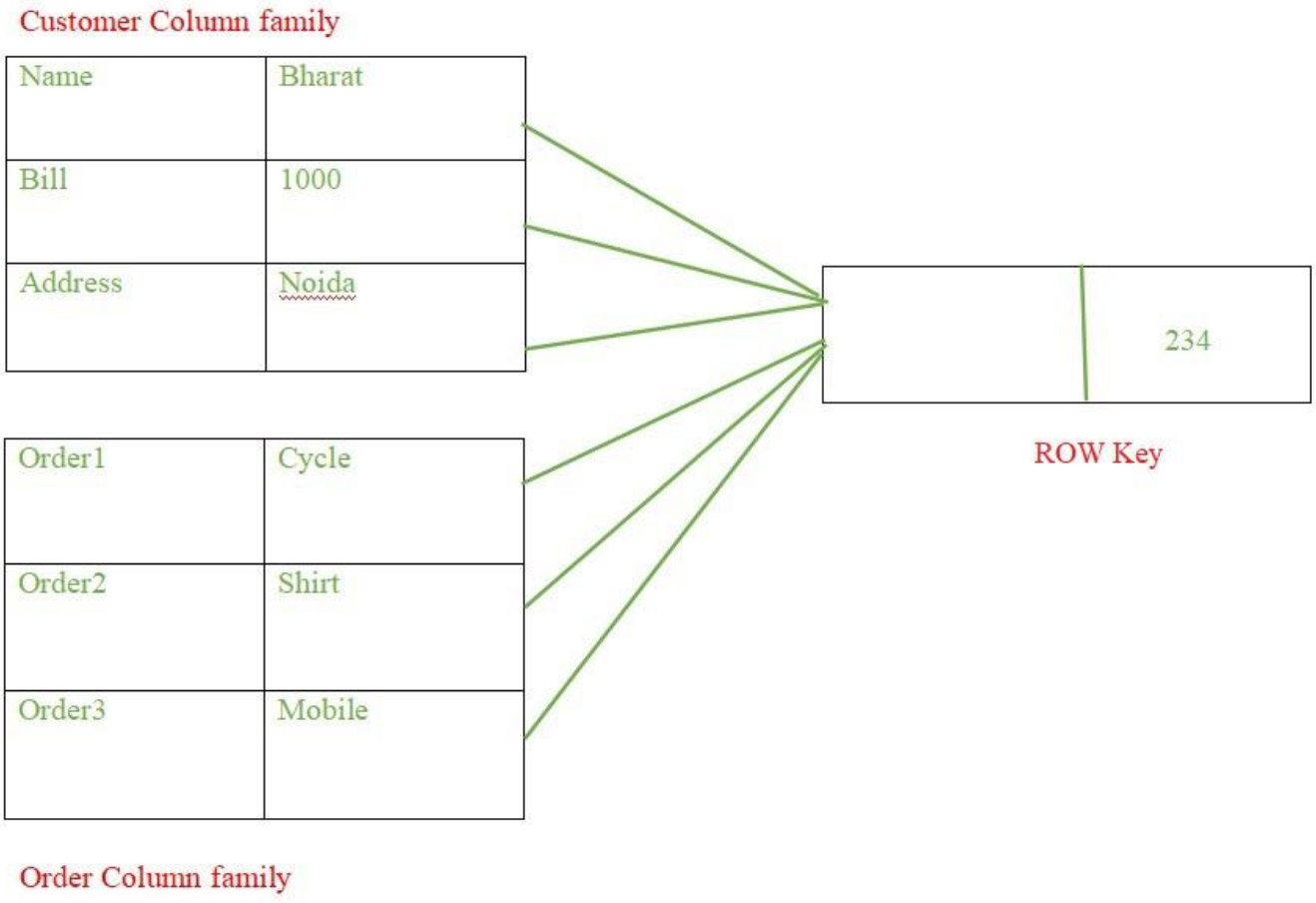
- The aggregate-Oriented database is the NoSQL database does not support ACID transactions.
- Aggregate orientation operations are different compared to relational database operations.
- The efficiency of the Aggregate-Oriented database is high if the data transactions and interactions take place within the same aggregate.
- Manipulations can be a single aggregate at a time. Multiple manipulate aggregates cannot be done at a time in an atomic way.

Aggregate – Oriented databases are classified into four major data models. They are as follows:

- Key-value
- Document
- Column family
- Graph-based

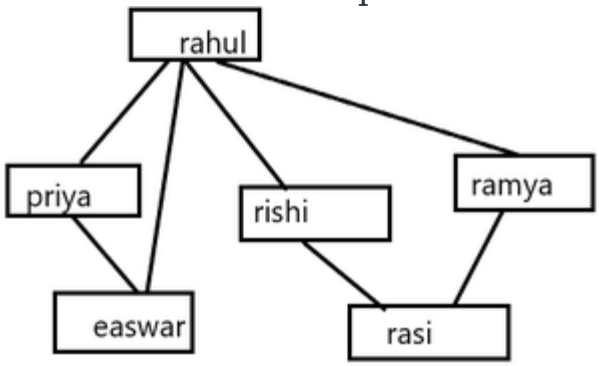
- **key-value Data Model:** Key-value and document databases were strongly aggregate-oriented. The key-value data model contains the key or Id which is used to access the data of the aggregates. key-value Data Model is very secure as the aggregates are opaque to the database. Aggregates are encrypted as the big blog of bits that can be decrypted with key or id. In the key-value Data Model, we can place data of any structure and datatypes in it.
- The advantage of the key-value Data Model is that we can store the sensitive information in the aggregate.

- The disadvantage of this model the database has some general size limits.
- We can store only the limited data.
- **Document Data Model:** In Document Data Model we can access the parts of aggregates. The data in this model can be accessed inflexible manner. we can submit queries to the database based on the fields in the aggregate. There is a restriction on the structure and data types of data to be paced in this data model. The structure of the aggregate can be accessed by the Document Data Model.
- **Column family Data Model:** The Column family is also called a two-level map. But, however, we think about the structure, it has been a model that influenced later databases such as HBase and Cassandra. These databases with a big table-style data model are often referred to as column stores. Column-family models divide the aggregate into column families. The Column-family model is a two-level aggregate structure. The first level consists of keys that act as a row identifier that selects the aggregate. The second-level values in the Column family Data Model are referred to as columns.



- In the above example, the row key is 234 which selects the aggregate. Here the row key selects the column families customer and orders. Each column family contains the columns of data. In the orders column family, we have the orders placed by the customers.
- **Graph Data Model:** In a graph data model, the data is stored in nodes that are connected by edges. This model is preferred to store a huge amount of complex aggregates and multidimensional data with many interconnections between them. Graph Data Model has the application

like we can store the Facebook user accounts in the nodes and find out the friends of the particular user by following the edges of the graph.



- Friends can find of a person by observing this graph data model.
- If there is an edge between two nodes then they are friends.
- The indirect links between the nodes to determine the friend suggestions.