Course

Database Management System

By
Prof. Ashvini Swami
Assistant Professor
Dept. of AI & DS Engg.

Unit – V

NoSQL Databases

Outline

- Introduction to Distributed Database System, Advantages, Disadvantages, CAP Theorem.

- Types of Data: Structured, Unstructured Data and Semi-Structured Data.

- NoSQL Database: Introduction, Need, Features. Types of NoSQL Databases: Key-value store, document store, graph, wide column stores,

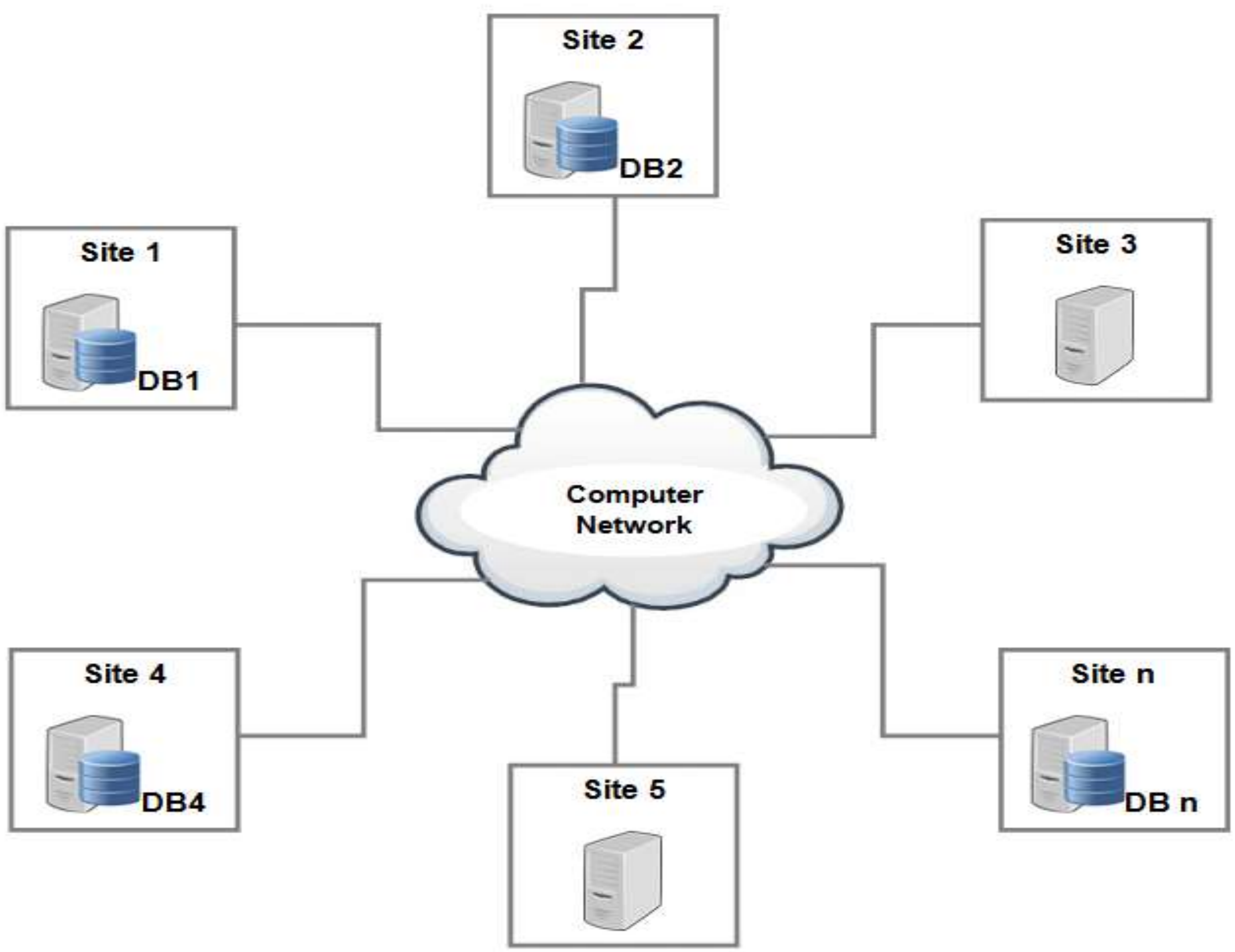
- BASE Properties, Data Consistency model, ACID Vs BASE, Comparative study of RDBMS and NoSQL.

- MongoDB (with syntax and usage): CRUD Operations, Indexing, Aggregation, MapReduce, Replication, Sharding.

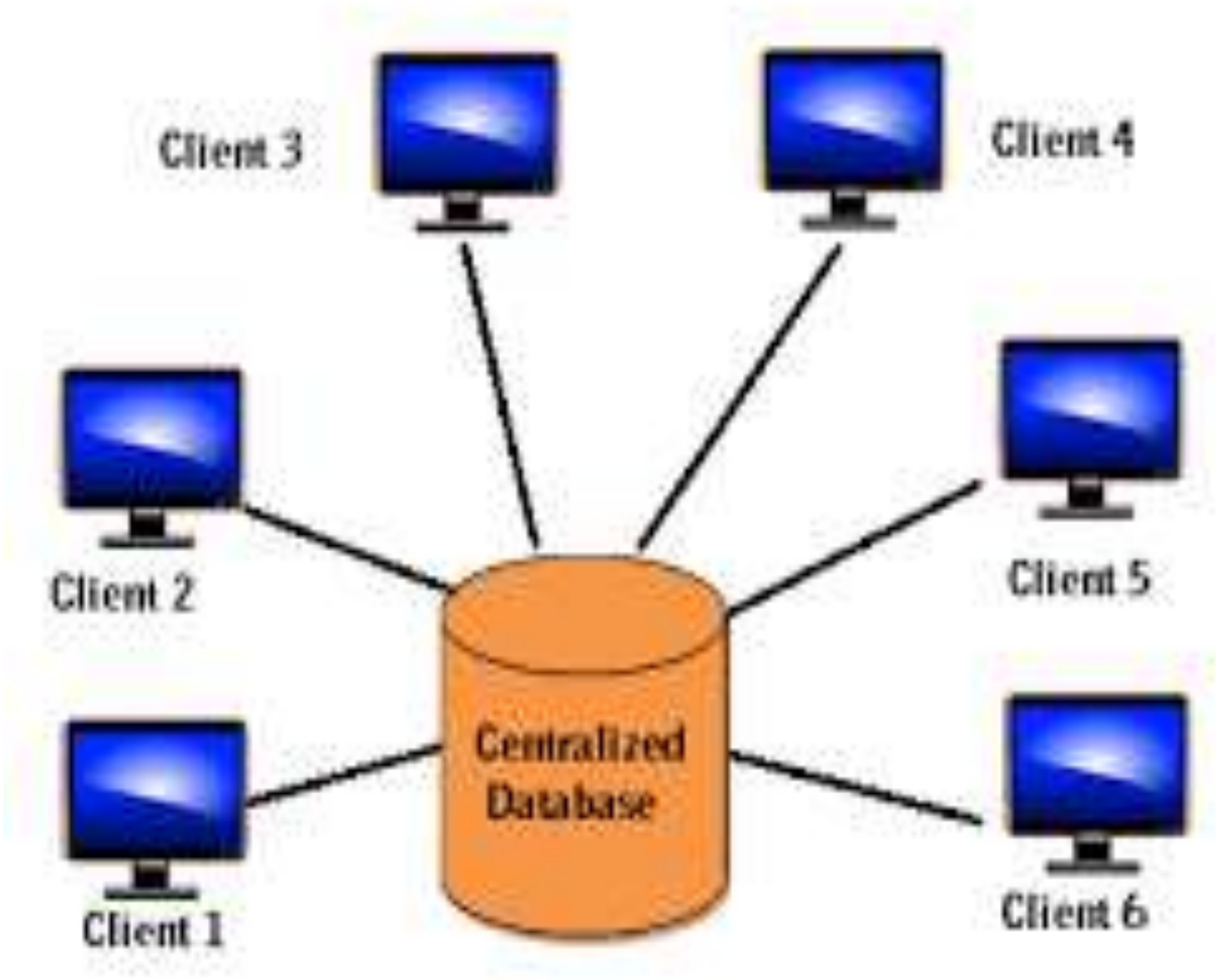
Introduction to Distributed Database System

- A **distributed database** is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.
- In other words each site is a data base system site in its own right and its own local users, its own local DBMS and its own local data communication s manager.
- Though there are many distributed databases to choose from, some examples of distributed databases include ***Apache Ignite*, *Apache Cassandra*, *Apache HBase*, *Amazon SimpleDB*, *Clusterpoint*, and *FoundationDB*.**

Introduction to Distributed Database System



Centralized Database System



Distributed Database System

Advantages of Distributed Databases:

- **Modular Development** − If the system needs to be **expanded** to new locations or new units, in centralized database systems, the action requires substantial efforts and disruption in the existing functioning. However, in distributed databases, the work simply requires adding new computers and local data to the new site and finally connecting them to the distributed system, **with no interruption** in current functions.

- **More Reliable** − In case of **database failures**, the total system of centralized databases comes to a **halt**. However, in distributed systems, when a component fails, the functioning of the system continues may be at a reduced performance. Hence **DDBMS is more reliable**.

Distributed Database System

Advantages of Distributed Databases:

- **Better Response** − If data is distributed in an efficient manner, then user requests can be met from local data itself, thus providing faster response. On the other hand, in centralized systems, all queries have to pass through the central computer for processing, which increases the response time.

- **Lower Communication Cost** − In distributed database systems, if data is located locally where it is mostly used, then the communication costs for data manipulation can be minimized. This is not feasible in centralized systems.

Distributed Database System

Disadvantages of Distributed Databases:

- **Complex nature :**

  The nature of distributed databases is much more **complex** as compared to centralized databases. This seeks the requirement of complex software that can handle the distributed database. Due to the absence of data replication, the complexity increases. Many computers which are present at different locations are connected via a network resulting in providing high–accuracy performance, availability, and reliability.

- **Overall cost :**

  Due to their complex nature, distributed DBMS take **high maintenance** which makes them costlier. It is more costly than normal dbms because of various other things like procurement costs, hardware costs, network/communication costs, labor costs, etc.

Distributed Database System

Disadvantages of Distributed Databases:

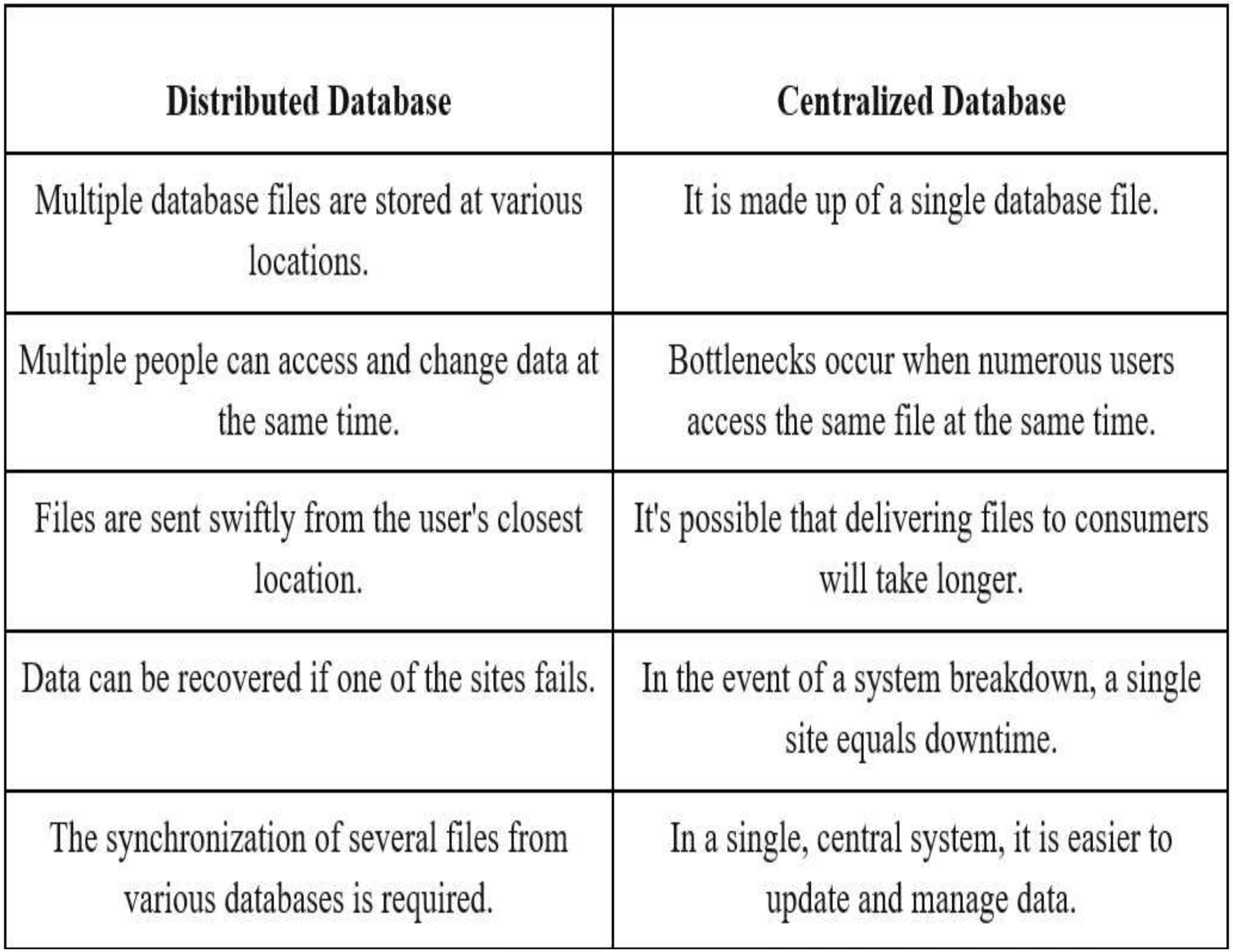
- **Security issues :**

  The security of data in the database is a major task. Distributed databases maintain no data redundancy and the security of data and network. Providing security is difficult because the data is stored in multiple databases with different locations, so security is maintained at every location along with the connecting networks. This task is shared along several sites with different people.

- **Integrity control :**

  Distributed databases consist of different sites, so it is important to provide data consistency. To ensure the integrity of data, distributed databases will **cost higher**, as the **communication** and **processing** cost is much **high**. The changes done in the single server must occur at other servers to maintain the integrity.

Comparison



Distributed Database	Centralized Database
Multiple database files are stored at various locations.	It is made up of a single database file.
Multiple people can access and change data at the same time.	Bottlenecks occur when numerous users access the same file at the same time.
Files are sent swiftly from the user's closest location.	It's possible that delivering files to consumers will take longer.
Data can be recovered if one of the sites fails.	In the event of a system breakdown, a single site equals downtime.
The synchronization of several files from various databases is required.	In a single, central system, it is easier to update and manage data.

Types of Distributed Database System

There are two types of distributed databases:

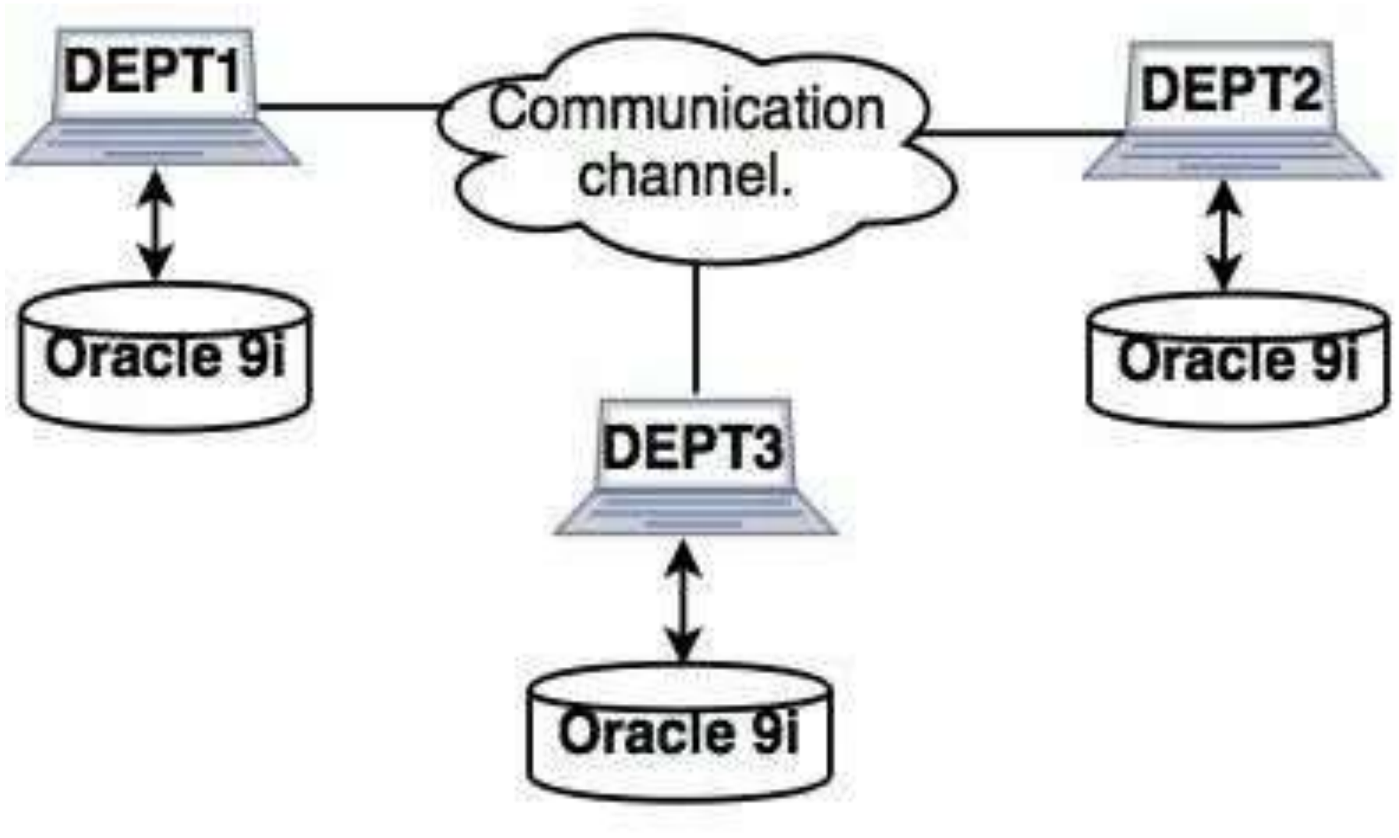
- **Homogenous** distributed database.
- **Heterogeneous** distributed database.

Homogenous Distributed Database :

- A **Homogenous** distributed database is a network of **identical databases** stored on multiple sites. All databases stores data identically, the ***operating system, DDBMS*** **and the** ***data structures used*** – all are same at all sites, making them **easy to manage**.

Types of Distributed Database System

Homogenous Distributed Database



Homogeneous distributed system

Types of Distributed Database System

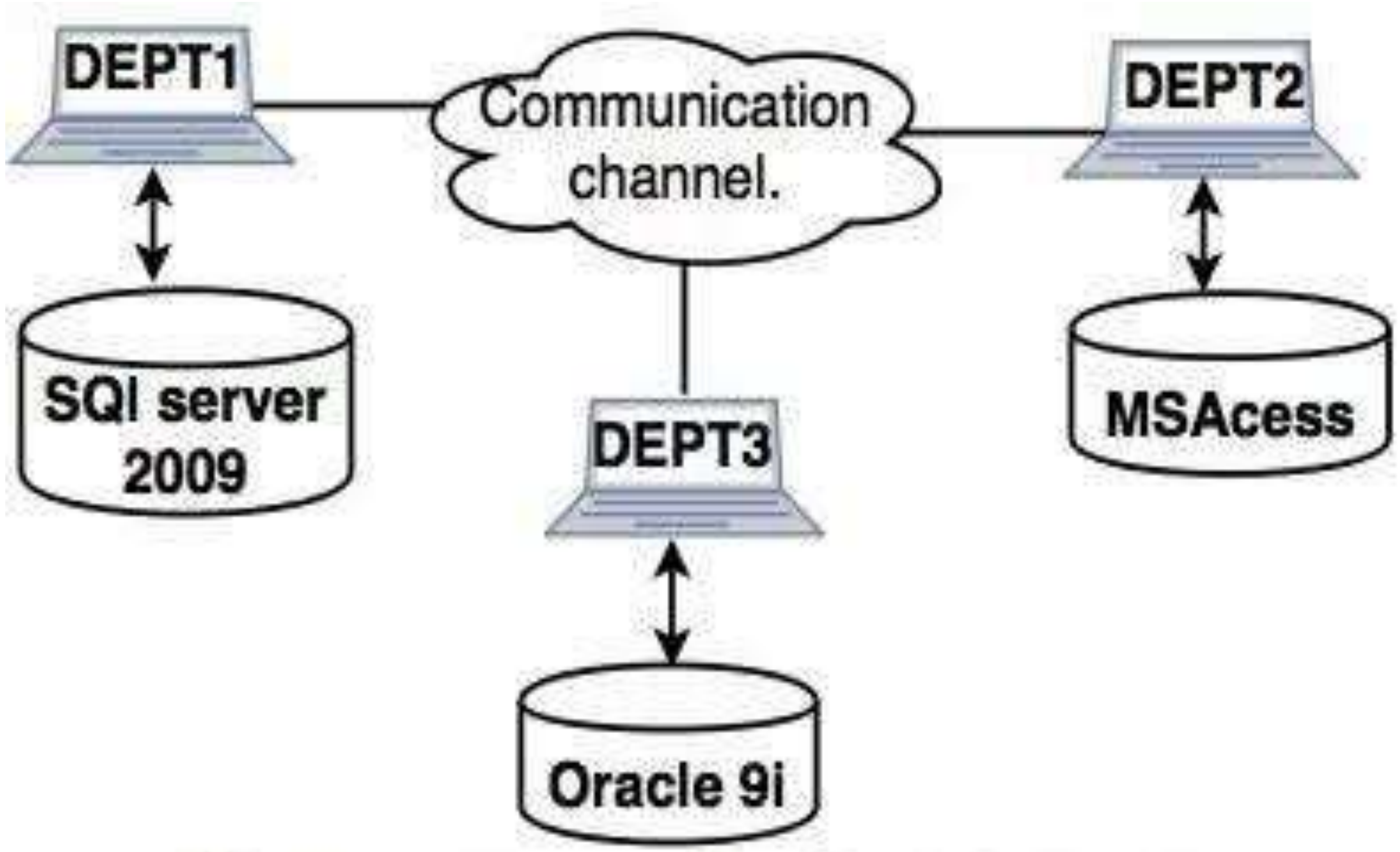
Heterogeneous distributed database :

- It is the opposite of a Homogenous distributed database. It uses *different schemas*, *operating systems*, *DDBMS*, and *different data models* causing it **difficult to manage**.

- In the case of a Heterogeneous distributed database, a particular site can be **completely unaware** of other sites. This causes **limited cooperation** in processing *user requests*, this is why **translations** are required to establish communication between sites.

Types of Distributed Database System

Heterogeneous distributed database :



Heterogeneous distributed system

Architecture of Distributed Database

Parameters of Distributed Database Systems:

- **Distribution:.**

  It describes **how data is physically distributed** among the several sites.

- **Autonomy:**

  It reveals the division of power inside the Database System and the **degree of autonomy** enjoyed by each individual DBMS.

- **Heterogeneity:**

  It speaks of the **similarity** or **differences** between the databases, system parts, and data models.

Architecture of Distributed Database

Architectural Model :

- Client-Server Architecture of DDBMS
- Peer-to-peer Architecture of DDBMS
- Multi DBMS Architecture of DDBMS

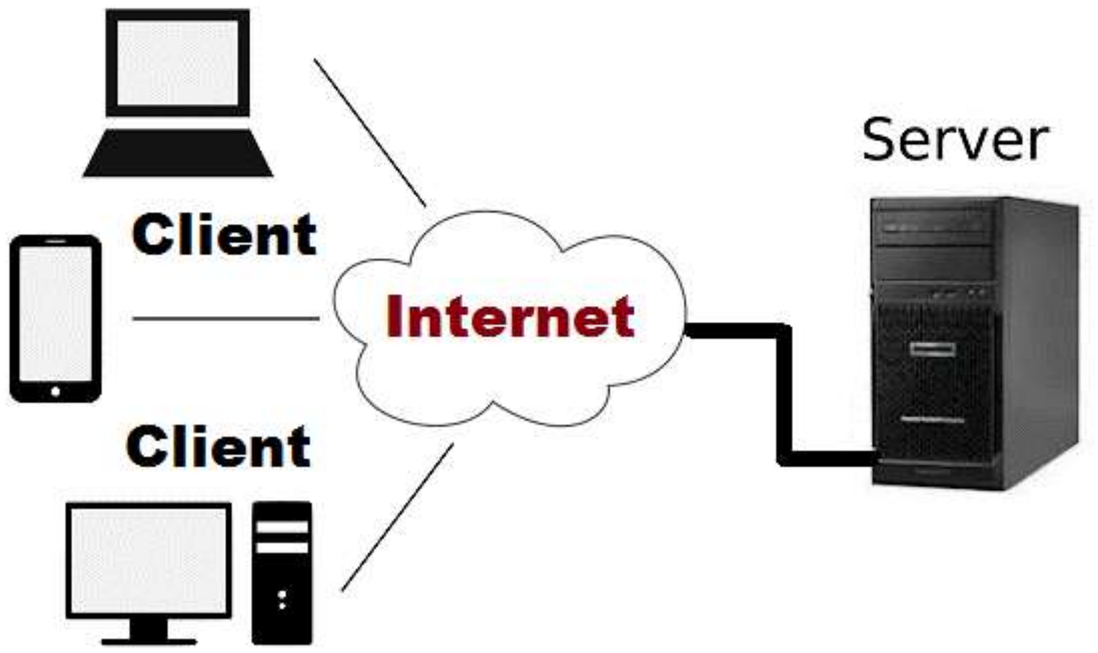
Client-Server Architecture of DDBMS :

- A client server architecture has a number of clients and a few servers connected in a network.
- A client sends a query to one of the servers. The earliest available server solves it and replies.
- A Client-server architecture is simple to implement and execute due to centralized server system.

Architecture of Distributed Database

Two types of Client-Server Architecture of DDBMS :

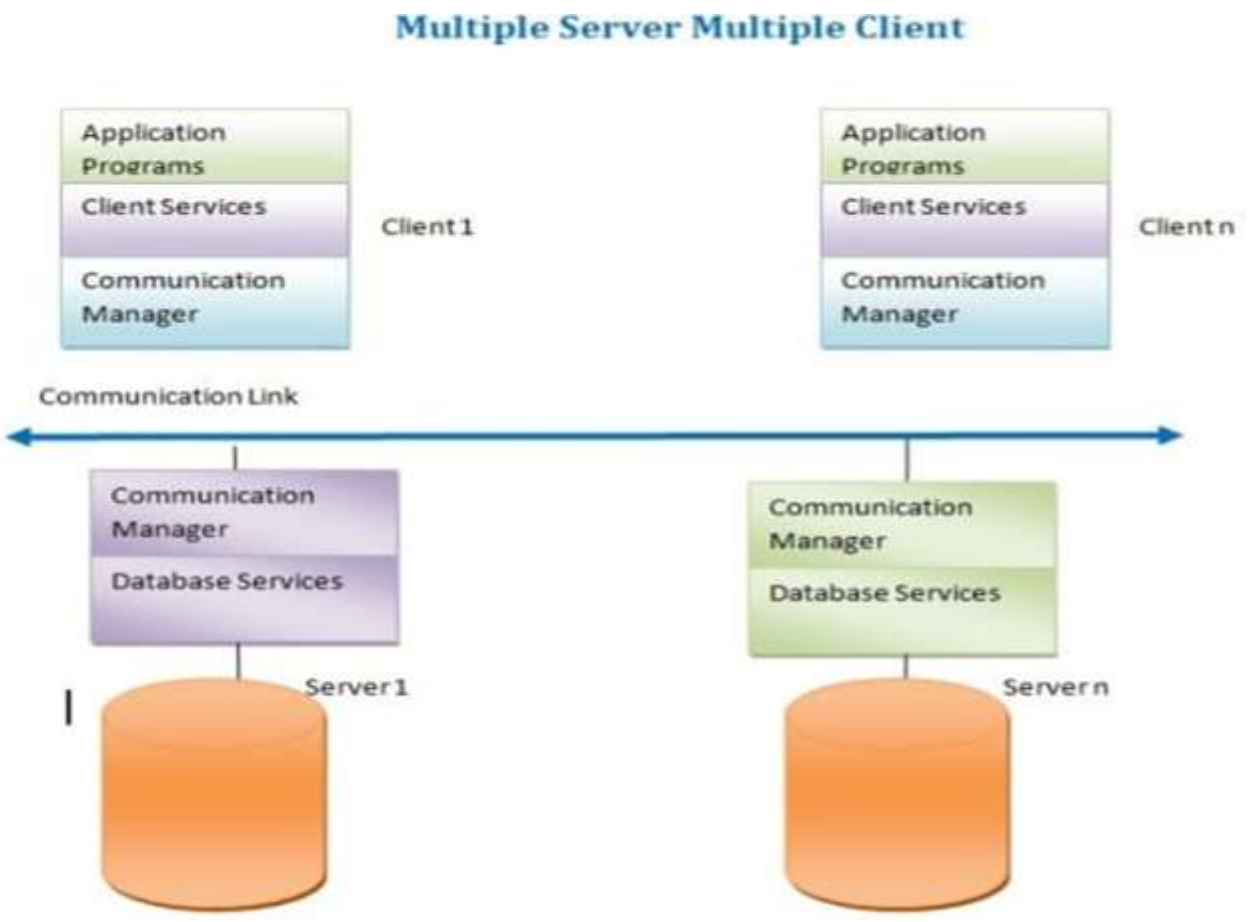
- Single Server Multiple Client :



Architecture of Distributed Database

Two types of Client-Server Architecture of DDBMS :

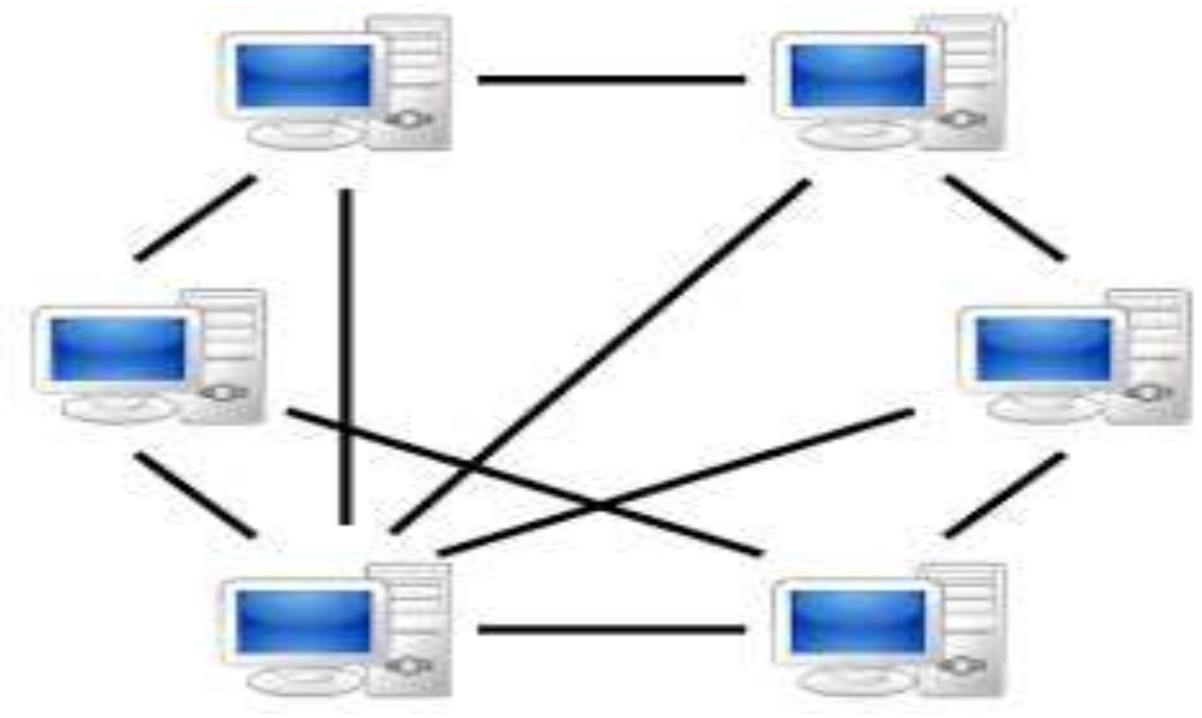
- Multiple Server Multiple Client :



Architecture of Distributed Database

Peer-to-peer Architecture of DDBMS:

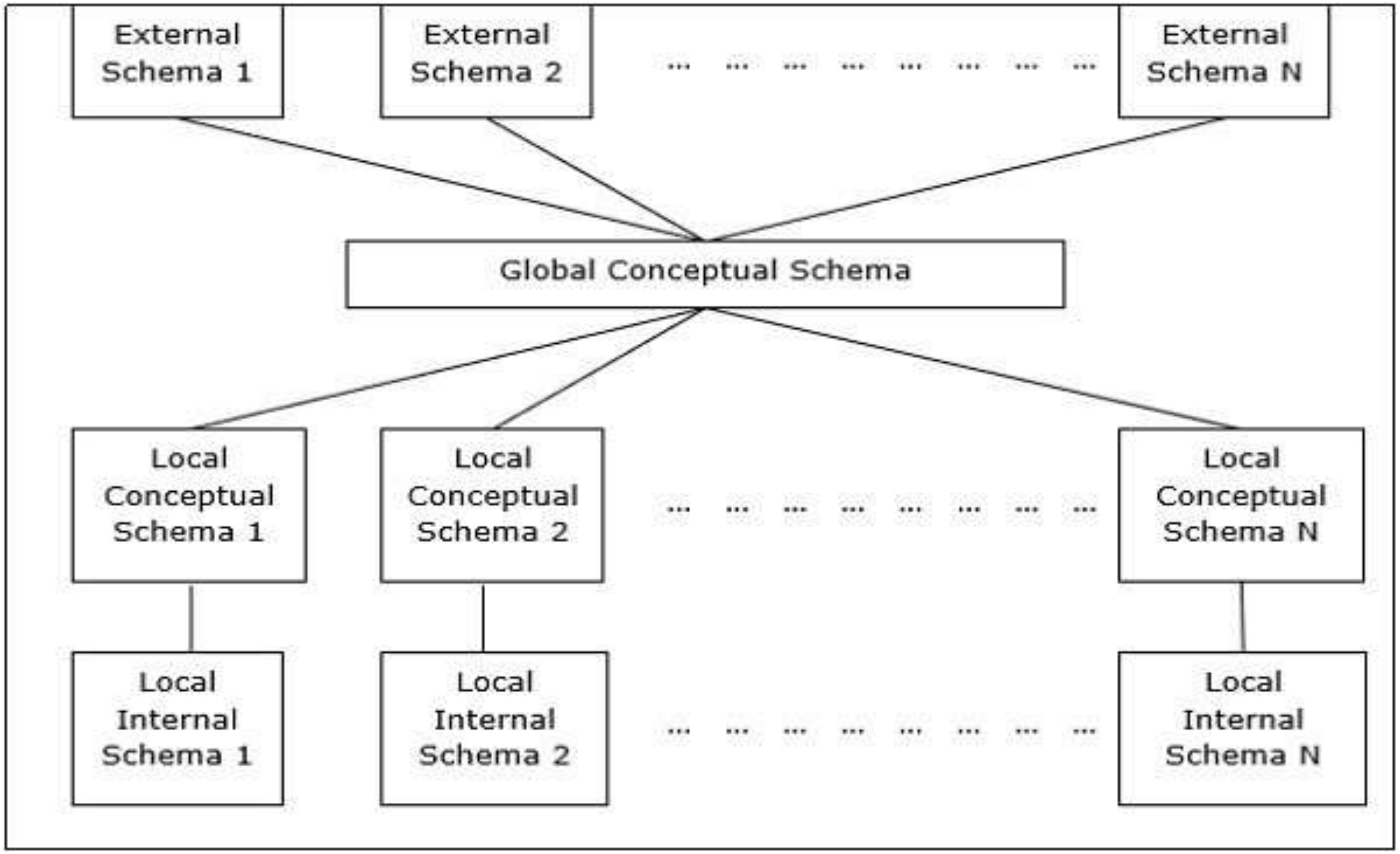
- In this architecture, each node or peer is considered as a server as well as a client, and it performs its database services as both **(server and client).**
- The peers **coordinate their efforts** and **share their resources** with one another.



Architecture of Distributed Database

Peer-to-peer Architecture of DDBMS:

This architecture generally has four levels of schemas.



Architecture of Distributed Database

Peer-to-peer Architecture of DDBMS:

This architecture generally has four levels of schemas.

- **Global Conceptual Schema:** Depicts the global logical view of data.
- **Local Conceptual Schema:** Depicts logical data organization at each site.
- **Local Internal Schema:** Depicts physical data organization at each site.
- **External Schema:** Depicts user view of data.

Architecture of Distributed Database

Multi - DBMS Architectures :

This is an integrated database system formed by a collection of **two or more autonomous database systems**. Multi-DBMS can be expressed through **six levels of schemas :**

- **Multi-database View Level:** Depicts **multiple user views** comprising of subsets of the integrated distributed database.
- **Multi-database Conceptual Level:** Depicts integrated multi-database that comprises of global **logical multi-database structure definitions.**
- **Multi-database Internal Level:** Depicts the data distribution across different sites and multi-database to **local data mapping.**

Architecture of Distributed Database

Multi - DBMS Architectures :

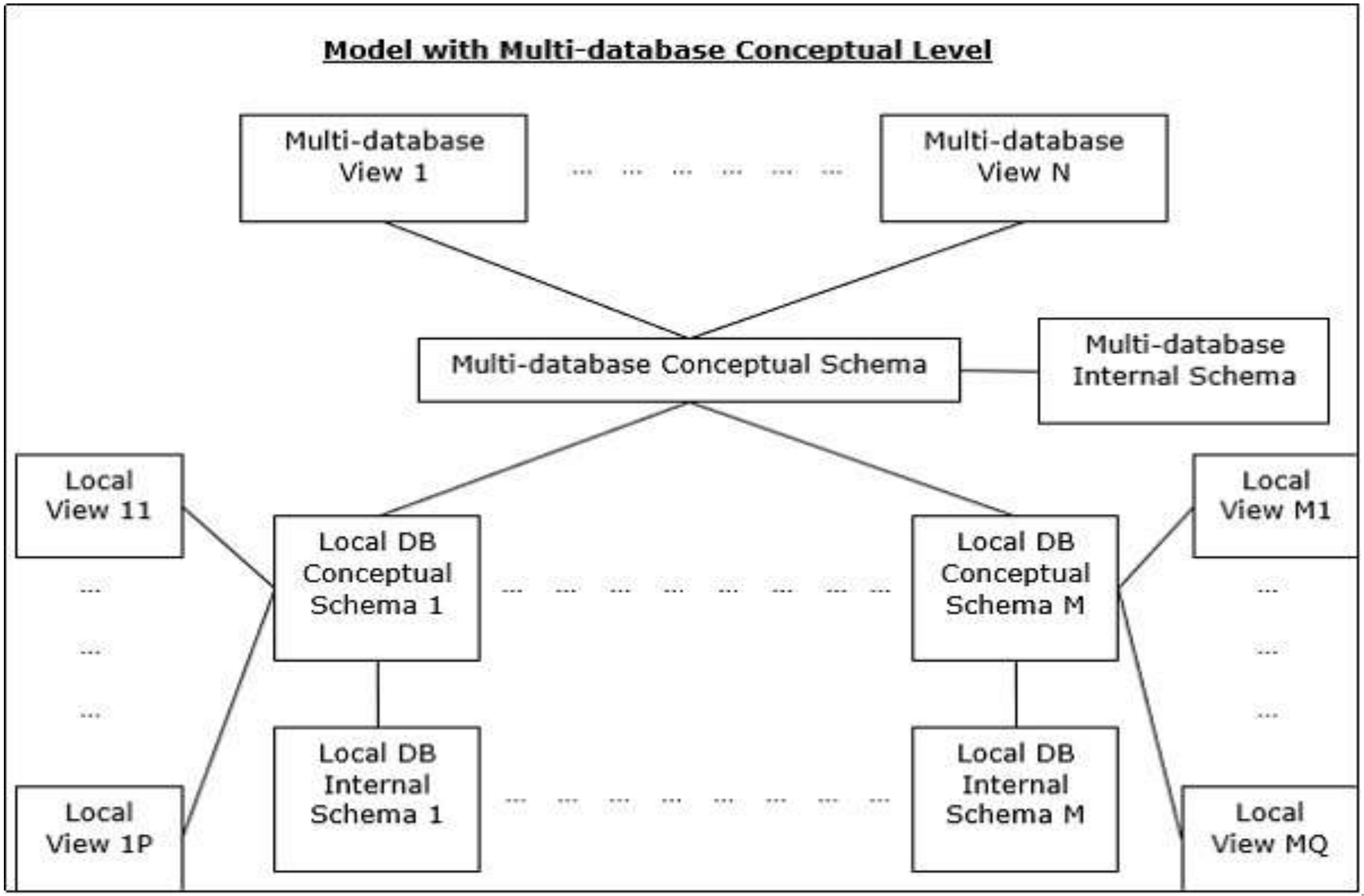
- **Local database View Level:** Depicts public view of local data.
- **Local database Conceptual Level:** Depicts local data organization at each site.
- **Local database Internal Level:** Depicts physical data organization at each site.

There are two design alternatives for multi-DBMS

- Model with multi-database conceptual level.
- Model without multi-database conceptual level.

Architecture of Distributed Database

Multi - DBMS Architectures :


Model with Multi-database Conceptual Level
Multi-database View 1
Multi-database View N
Multi-database Conceptual Schema
Multi-database Internal Schema
Local View 11
Local View 1P
Local DB Conceptual Schema 1
Local DB Internal Schema 1
Local DB Conceptual Schema M
Local DB Internal Schema M
Local View M1
Local View MQ

Distributed Database Design

- The design of a distributed computer system involves making decisions on the placement of data and programs across the sites of a computer network.
- Organization of distributed systems can be investigated along three orthogonal dimensions (Figure 3.1) :

1. **Level of sharing**
2. **Behavior of access patterns**
3. **Level of knowledge on access pattern behavior**

Distributed Database Design

72 3 Distributed Database Design

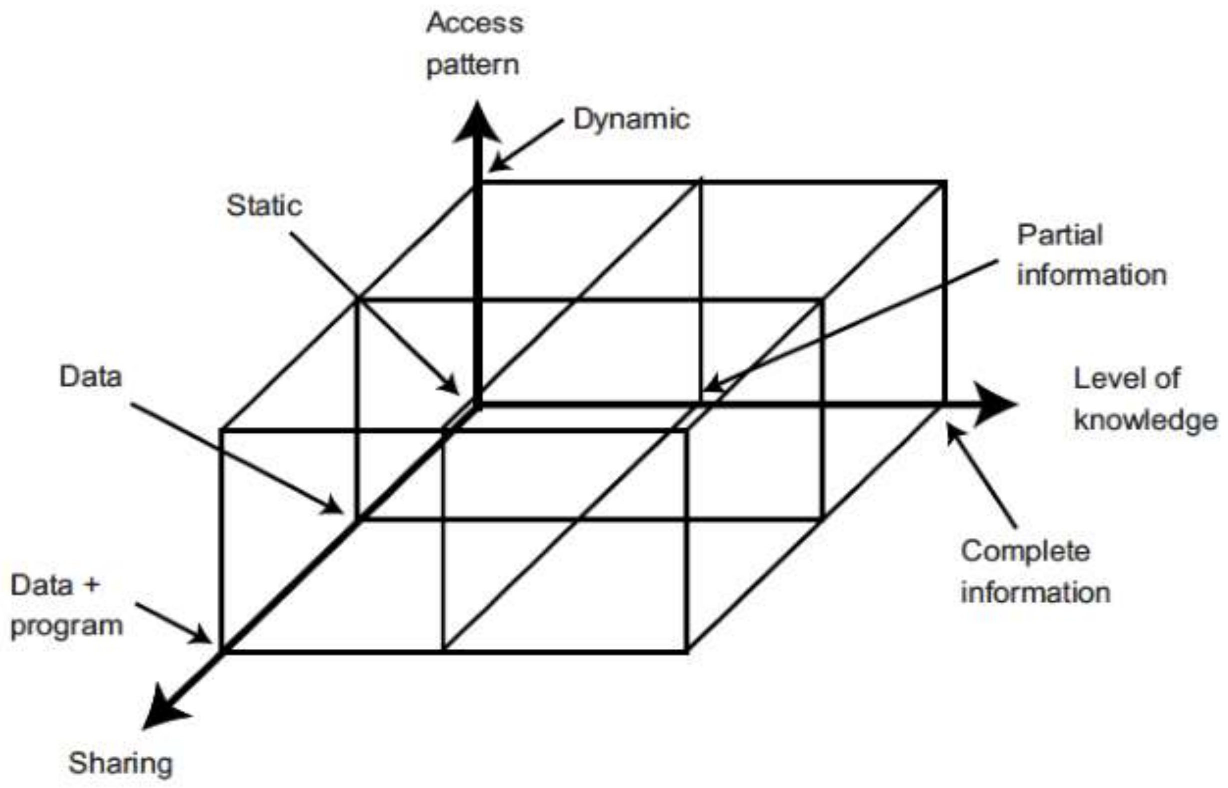


Fig. 3.1 Framework of Distribution

Distributed Database Design

Level of sharing: In terms of the level of sharing, there are three possibilities.

- **No sharing:** Each application and its data execute at one site, and there is no communication with any other program or access to any data file at other sites.
- **Data sharing:** All the **programs** are **replicated** at all the sites, but **data files are not**. Accordingly, user requests are handled at the site where they originate and the necessary data files are moved around the network.
- **Data-plus-program sharing:** both data and programs may be shared.

Design Strategies Distributed Database

There are two classical approaches to database design:

Top-down :

- Top down approach is more suitable for tightly integrated, homogeneous distributed DBMSs.
- The activity begins with a requirements analysis that defines the environment of the system and “elicits both the data and processing needs of all potential database users”.
- The requirements document is input to two parallel activities: view design and conceptual design.
- The view design activity deals with defining the interfaces for end users.

Design Strategies Distributed Database

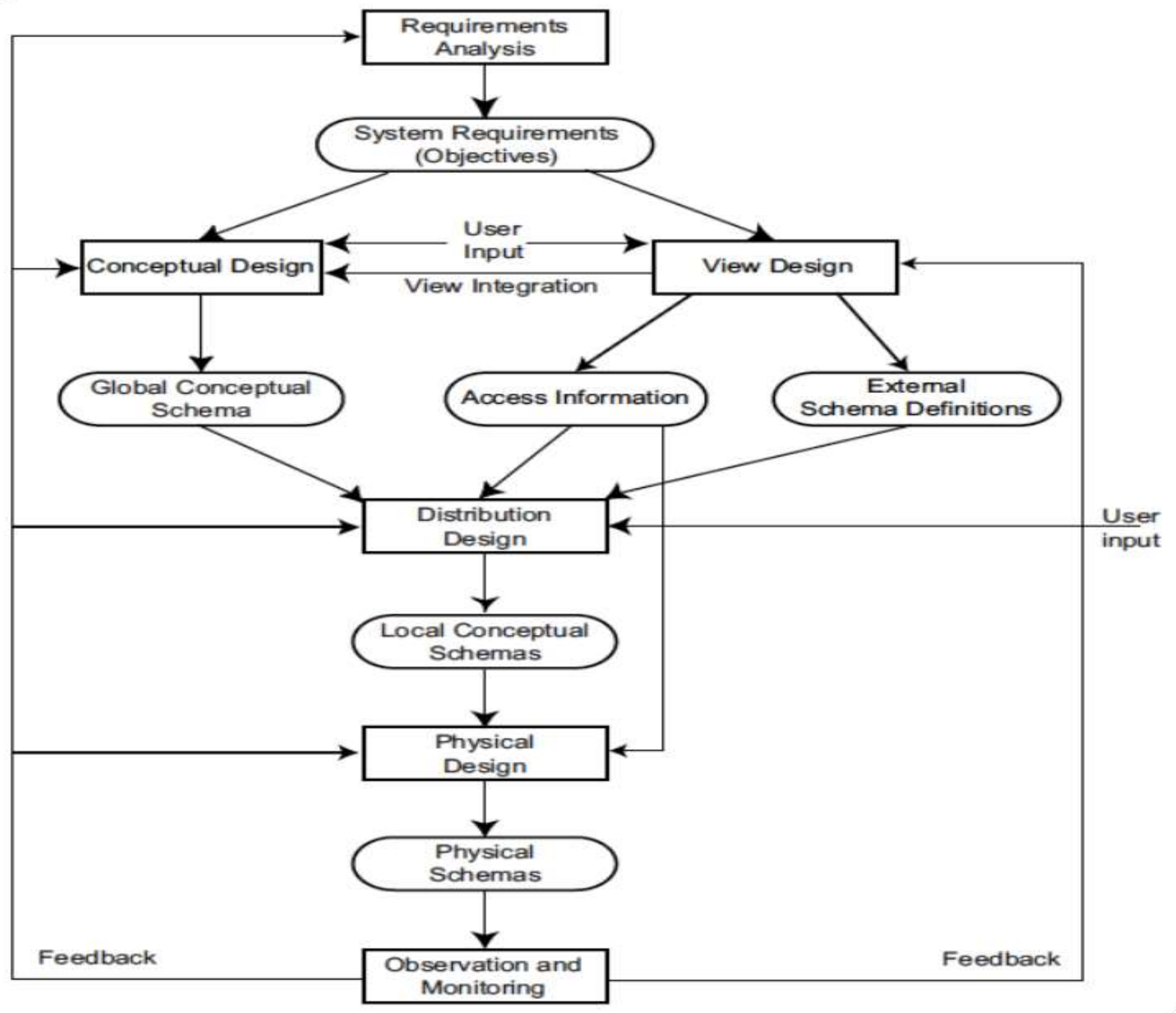


Fig. 3.2 Top-Down Design Process

Design Strategies Distributed Database

Bottom – Up :

- The database is already exiting at a number of sites already.

- To solve any common task database should be connected.

Distributed Data Storage

There are three methods by which we can store the data on different sites:

- **Replication**
- **Fragmentation**
- **Allocation**

Replication :

- This method involves redundantly storing the full **relationship** at two or more locations. Since a complete database can be accessed from each site, it becomes a redundant database. Systems preserve copies of the data as a result of replication.
  - **Fully Replicated**
  - **Partially Replicated**
  - **No Replication**

Distributed Data Storage

Fragmentation :

- According to this method, the **relationships** are divided (i.e., broken up into smaller pieces), and each fragment is stored at the many locations where it is needed.
- To ensure there is no data loss, the pieces must be created in a way that allows for the reconstruction of the original relation.

Ways of fragmentation:

Horizontal Fragmentation:

- In Horizontal Fragmentation, the relational table or schema is broken down into a group of one and more rows, and each row gets one fragment of the schema.
- It is also called **splitting by rows**.

Distributed Data Storage

Fragmentation :

Vertical Fragmentation:

- In this fragmentation, a relational table or schema is divided into some more schemas of smaller sizes.
- A common candidate key must be present in each fragment in order to guarantee a lossless join.
- This is also called **splitting by columns**.

Distributed Data Storage

Allocation :

Decides locations of different data for storage purpose.

Allocation can be

- **Centralized :** Data stored on particular site. No distribution.
- **Partitioned :** Database divided and stored on different sites.
- **Replicated :** Copies of database stored at several places.

CAP Theorem

- The **CAP theorem** (also called Brewer's theorem) states that a distributed database system can only guarantee two out of these three characteristics:
- **C**onsistency
- **A**vailability, and
- **P**artition Tolerance.

CAP Theorem

Consistency :

- A system is said to be consistent if all nodes see the same data at the same time.
- Simply, if we perform a read operation on a consistent system, it should return the value of the most recent write operation. This means that, the read should cause all nodes to return the same data, i.e., the value of the most recent write.

CAP Theorem

Availability :

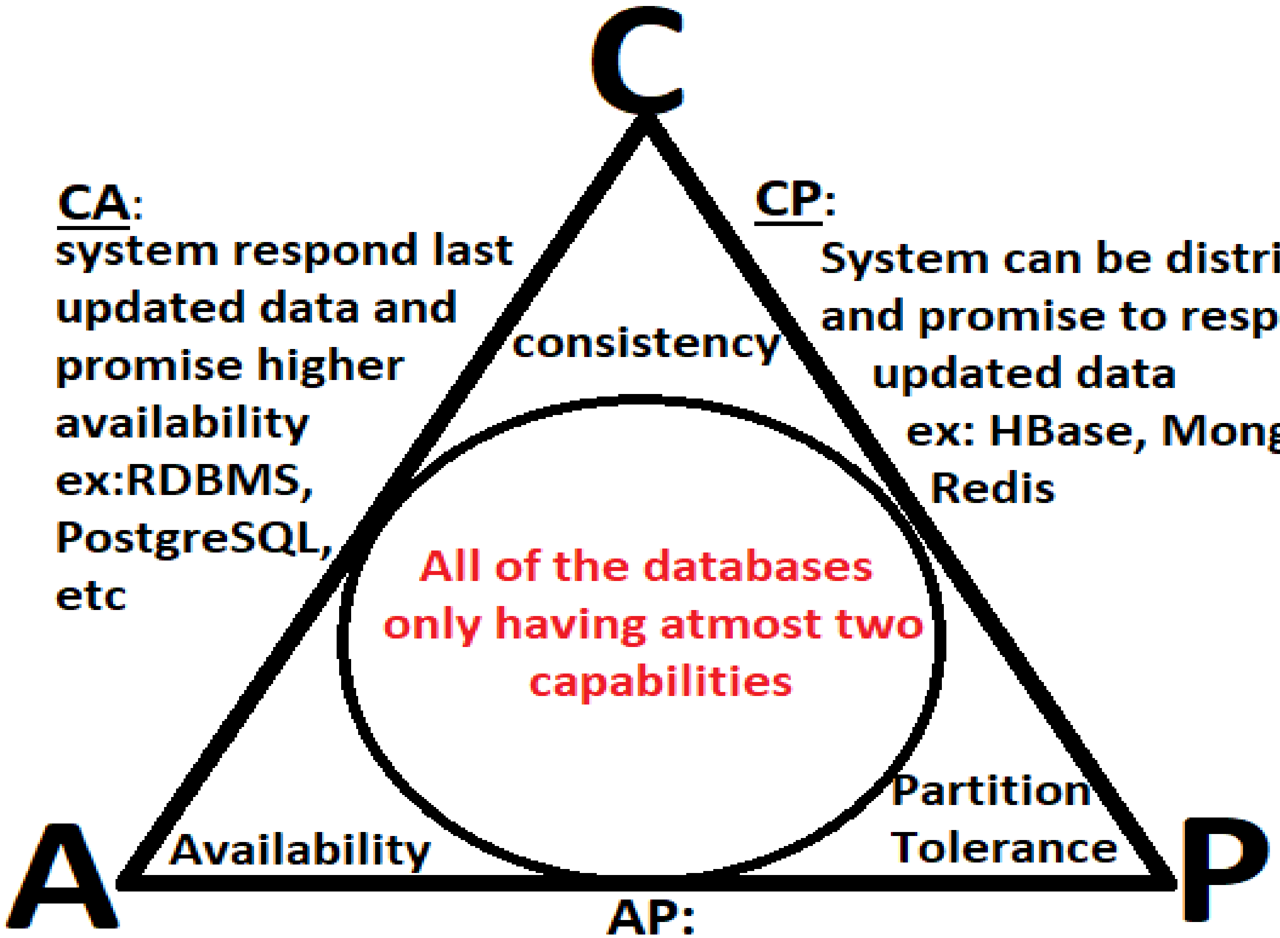
- Availability in a distributed system ensures that the system remains operational 100% of the time. Every request gets a (non-error) response regardless of the individual state of a node.
- Note: this does not guarantee that the response contains the most recent write.

CAP Theorem

Partition Tolerance :

- This condition states that the system does not fail, regardless of if messages are dropped or delayed between nodes in a system.
- Partition tolerance has become more of a necessity than an option in distributed systems. It is made possible by sufficiently replicating records across combinations of nodes and networks.

CAP Theorem



CA:
system respond last updated data and promise higher availability
ex:RDBMS, PostgreSQL, etc

CP:
System can be distributed and promise to respond last updated data
ex: HBase, MongoDB, Redis

AP:
System can be ditributed and promise to has high availability
ex: Cassandra, CouchDB, Riak, Voldemort, DynamoDB, etc

CAP Theorem

CAP theorem NoSQL database types :

NoSQL databases are ideal for distributed network applications.

Today, NoSQL databases are classified based on the two CAP characteristics they support:

CP database:

- A CP database delivers **consistency** and **partition tolerance** at the expense of **availability**. When a partition occurs between any two nodes, the system has to shut down the non-consistent node **(i.e., make it unavailable)** until the partition is resolved.

CAP Theorem

CAP theorem NoSQL database types :

AP database:

- An AP database delivers **availability** and **partition tolerance** at the expense of **consistency**. When a partition occurs, all nodes remain available but those at the wrong end of a partition might return an older version of data than others. (When the partition is resolved, the AP databases typically resync the nodes to repair all **inconsistencies** in the system.)

CA database:

- A CA database delivers **consistency** and **availability** across all nodes. It can't do this if there is a partition between any two nodes in the system, however, and therefore can't deliver fault tolerance.

Types of Big Data

Big Data :

"Big data" is high-volume, velocity, and variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making."

Types of Big Data

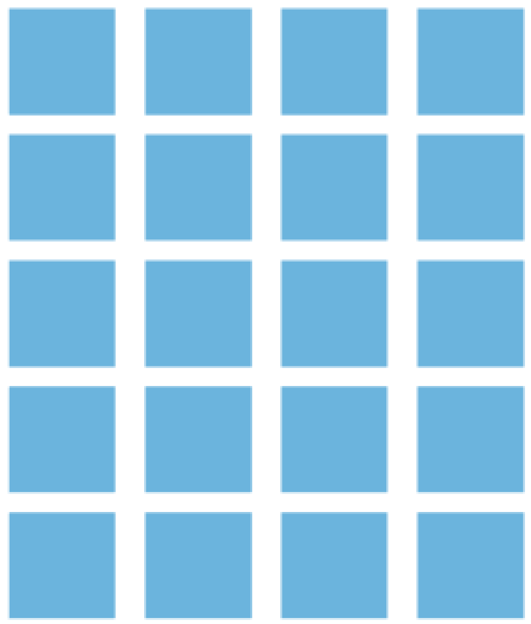
1. **Structured Data**
2. **Unstructured Data**
3. **Semi-structured Data**

Types of Big Data

1. Structured Data :

- The data which is to the point, factual, and highly organized is referred to as structured data. It is quantitative in nature, i.e., it is related to quantities that means it contains measurable numerical values like numbers, dates, and times.

- It is easy to search and analyze structured data. Structured data exists in **a predefined format**.
- **Relational database** consisting of tables with rows and columns is one of the best examples of structured data.
- Structured data generally exist in tables like excel files and Google Docs spreadsheets.



Structured Data

Types of Big Data

2. Unstructured Data :

- All the unstructured files, log files, audio files, and image files are included in the unstructured data. Some organizations have much data available, but they did not know how to derive data value since the data is raw.

Types of Big Data

2. Unstructured Data :

- Unstructured data is the data that **lacks** any **predefined model** or format. It **requires** a **lot of storage space**, and it is **hard** to **maintain security**. **Managing, analyzing, or searching** for unstructured data is **hard**. It resides in various **different formats** like text, images, audio and video files, etc.
- It is **qualitative** in nature and sometimes stored in a non-relational database or NO-SQL.
- It is **not stored in relational databases**, so it is hard for computers and humans to interpret it. The **amount** of unstructured data is **much more than** the **structured** or **semi-structured** data.
- Examples of human-generated unstructured data are Text files, Email, social media, media, mobile data, business applications, and others. The machine-generated unstructured data includes satellite images, scientific data, sensor data, digital surveillance, and many more.

Types of Big Data

3. Semi-structured :

- Semi-structured data (e.g., JSON, CSV, XML) is the "**bridge**" between structured and unstructured data. It does not have a **predefined data model and is more complex than structured data, yet easier to store than unstructured data.**

- Semi-structured data uses "metadata" (e.g., tags and semantic markers) to identify specific data characteristics and scale data into records and preset fields. Metadata ultimately enables semi-structured data to be better cataloged, searched and analyzed than unstructured data.

Difference

On the basis of	Structured data	Unstructured data
Technology	It is based on a relational database.	It is based on character and binary data.
Flexibility	Structured data is less flexible and schema-dependent.	There is an absence of schema, so it is more flexible.
Scalability	It is hard to scale database schema.	It is more scalable.
Robustness	It is very robust.	It is less robust.
Format	It has a predefined format.	It has a variety of formats, i.e., it comes in a variety of shapes and sizes.

Difference

On the basis of	Structured data	Unstructured data
Performance	Here, we can perform a structured query that allows complex joining, so the performance is higher.	While in unstructured data, textual queries are possible, the performance is lower than semi-structured and structured data.
Nature	Structured data is quantitative, i.e., it consists of hard numbers or things that can be counted.	It is qualitative, as it cannot be processed and analyzed using conventional tools.
Analysis	It is easy to search.	Searching for unstructured data is more difficult.

Introduction to NoSQL Database

- NoSQL Database is used to refer a Non-SQL or non relational database or Not Only SQL.
- It provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases.
- NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications.
- NoSQL is an approach to database management that can accommodate a wide variety of data models, including **key-value, document, columnar and graph** formats. A NoSQL database generally means that it **is non-relational, distributed, flexible and scalable.**

History of NoSQL Database

- The term *NoSQL* was used by **Carlo Strozzi** in 1998 to name his lightweight Strozzi NoSQL open-source relational database that did not expose the standard Structured Query Language (SQL) interface, but was still relational.
- His NoSQL RDBMS is distinct from the around-2009 general concept of NoSQL databases. Strozzi suggests that, because the current NoSQL movement "departs from the relational model altogether, it should therefore have been called more appropriately **'NoREL'**",referring to **“Not Relational**".
- **Johan Oskarsson**, then a developer at Last.fm reintroduced the term ***NoSQL*** in early 2009 when he organized an event to discuss "open-source distributed, non-relational databases".
- The name attempted to label the emergence of an increasing number of non-relational, distributed data stores, including open source clones of Google's Bigtable/MapReduce and Amazon's DynamoDB.

Features of NoSQL Database

Each NoSQL database has its own unique features. At a high level, many NoSQL databases have the following features:

- Flexible schemas
- Horizontal scaling
- Fast queries due to the data model
- Ease of use for developers

Why NoSQL?

NoSQL databases are used in nearly every industry. Use cases range from the highly critical (e.g., storing financial data and healthcare records) to the more fun abrnd frivolous (e.g., storing IoT readings from a smart kitty litter box).

When should NoSQL be used?

When deciding which database to use, decision-makers typically find one or more of the following factors lead them to selecting 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.

Why NoSQL?

NoSQL databases are used in nearly every industry. Use cases range from the highly critical (e.g., storing financial data and healthcare records) to the more fun abrnd frivolous (e.g., storing IoT readings from a smart kitty litter box).

When should NoSQL be used?

When deciding which database to use, decision-makers typically find one or more of the following factors lead them to selecting 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.

Advantages of NoSQL

There are several advantages to using NoSQL databases, including:

- NoSQL databases **simplify application development**, particularly for interactive real-time web applications, such as those using a REST API and web services.
- These databases provide **flexibility** for data that has not been normalized, which requires a flexible data model, or has different properties for different data entities.
- They offer **scalability** for larger data sets, which are common in analytics and artificial intelligence (AI) applications.
- NoSQL databases are **better suited** for cloud, mobile, social media and big data requirements.
- They are designed for specific use cases and are easier to use than general-purpose relational or SQL databases for those types of applications.

Disadvantages of NoSQL

- Because NoSQL databases are newer, there are **no** comprehensive **industry standards** as with relational and SQL DBMS offerings.
- **Lack of a rigid database schema** and constraints removes the data integrity safeguards that are built into relational and SQL database systems.
- Because most NoSQL databases use the eventual consistency model, they do not provide the same level of data consistency as SQL databases. At times the **data will not be consistent**, which means they are not well-suited for transactions that require immediate integrity, such as banking and ATM transactions.

Types of NoSQL databases

There are four popular types of NoSQL database systems. Each uses a different type of data model, resulting in significant differences between each NoSQL type.

- Key-value store
- Document store
- Wide-column store
- Graph databases

Types of NoSQL databases

Key-value store :

- Also known as **key-value databases**, these systems implement a simple data model that pairs a unique key with an associated value.
- Because this model is simple, it can be used to develop highly scalable and performant applications.
- Key-value databases are ideal for session management and caching in web applications, such as those needed when managing shopping cart details for online buyers or for managing session details for multiplayer gaming.

Types of NoSQL databases

Key-value store :

- Implementations differ in the way they are oriented to work with RAM, solid-state drives or disk drives.

- Examples of popular key-value databases include Aerospike, DynamoDB, Redis and Riak.

Types of NoSQL databases

Document databases :

- Also called document stores, these databases store semi-structured data and descriptions of that data in **document** format.
- They enable developers to create and update programs without needing to reference master schema.
- Use of document databases has increased along with the use of JavaScript and the JavaScript Object Notation (JSON), a data interchange format that has gained wide currency among web application developers.

Types of NoSQL databases

Document databases :

- Document databases are used for content management and mobile application data handling, such as blogging platforms, web analytics and e-commerce applications.

- Couchbase Server, CouchDB, MarkLogic and MongoDB are examples of document databases.

Types of NoSQL databases

Wide-column store :

- These databases use familiar tables, columns and rows like relational database tables, but column names and formatting can differ from row to row in a single table.
- Each column is also stored separately on disk.
- As opposed to traditional row-orientated storage, a wide-column store is optimal when querying data by columns. Typical applications where wide-column stores can excel include recommendation engines, catalogs, fraud detection and event-logging.
- Accumulo, Amazon SimpleDB, Cassandra, HBase and Hypertable are examples of wide-column stores.

Types of NoSQL databases

Graph databases :

- Graph data stores organize data as **nodes**, which are similar to rows in a relational database, and **edges**, which represent connections between nodes.
- Because the graph system stores the relationship between nodes, it can support richer representations of data relationships.
- Also, unlike relational models that rely on strict schemas, the graph data model can evolve over time and use.

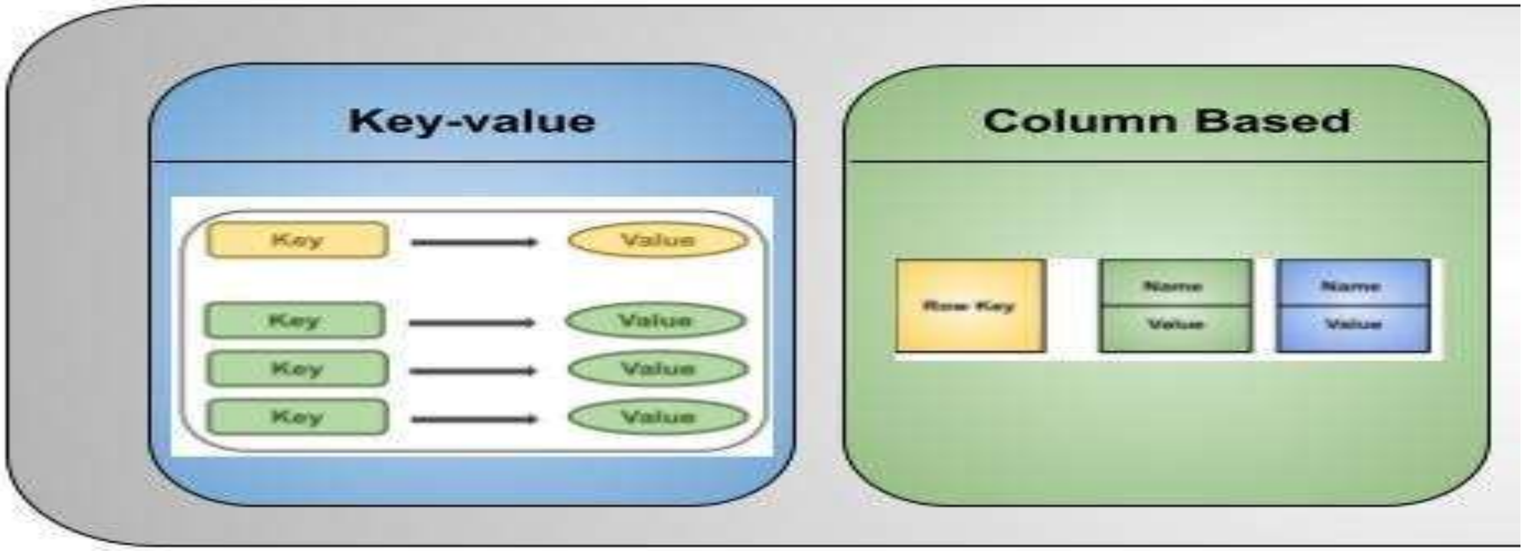
Types of NoSQL databases

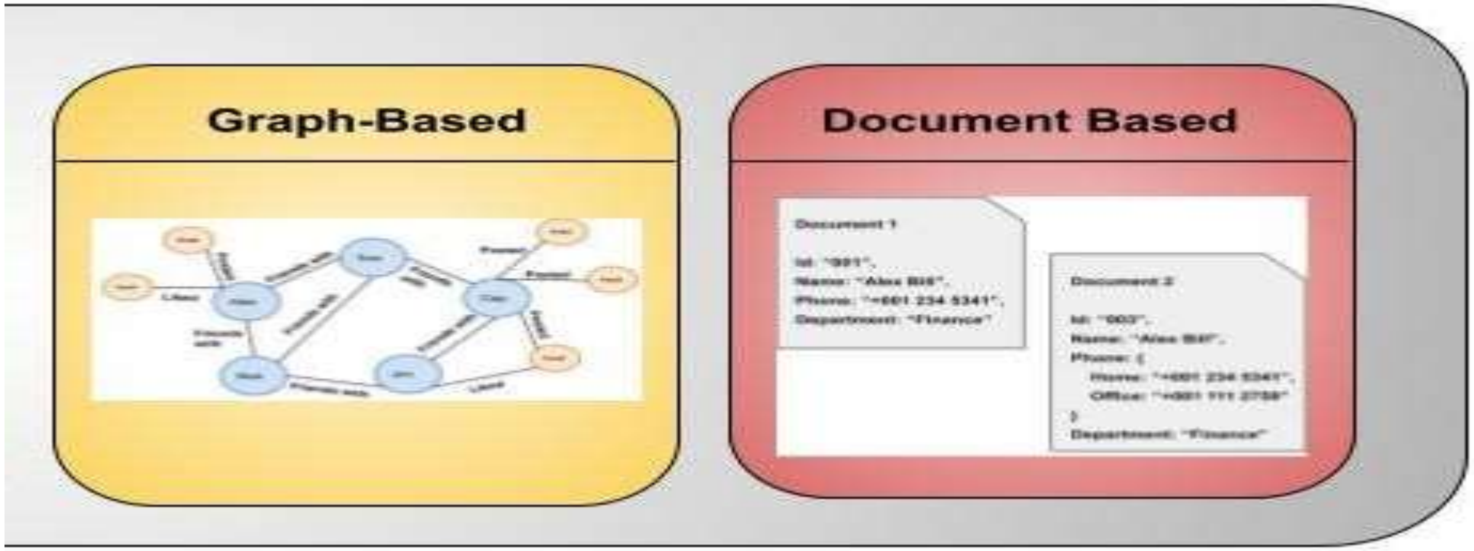
Graph databases :

- Graph databases are applied in systems that must map relationships, such as social media platforms, reservation systems or customer relationship management.

- Examples of graph databases include AllegroGraph, IBM Graph and Neo4j.

Types of NoSQL databases





Document 1

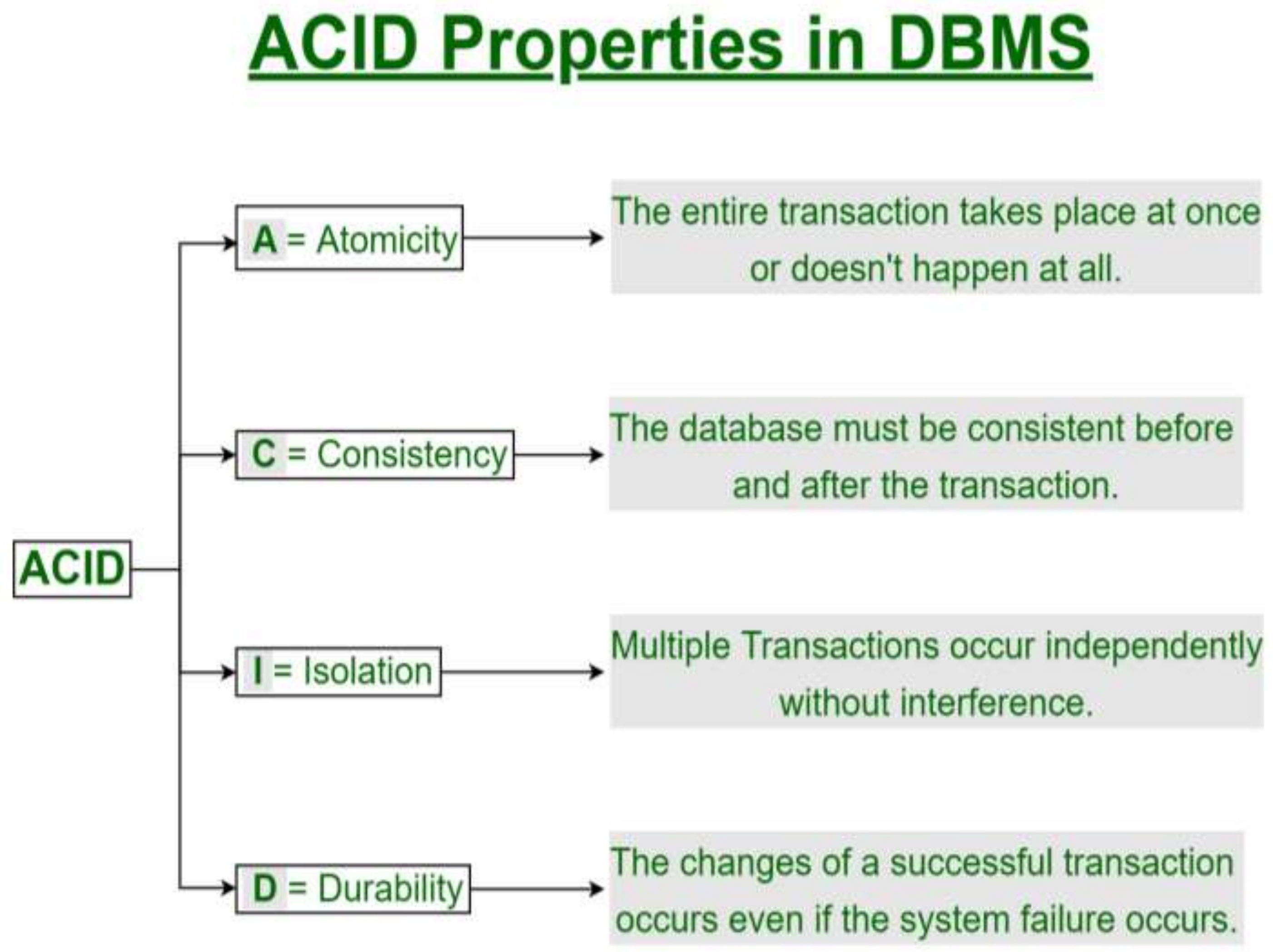
Id: "001",
Name: "Alex Bill",
Phone: "+001 234 5341",
Department: "Finance"

Document 2

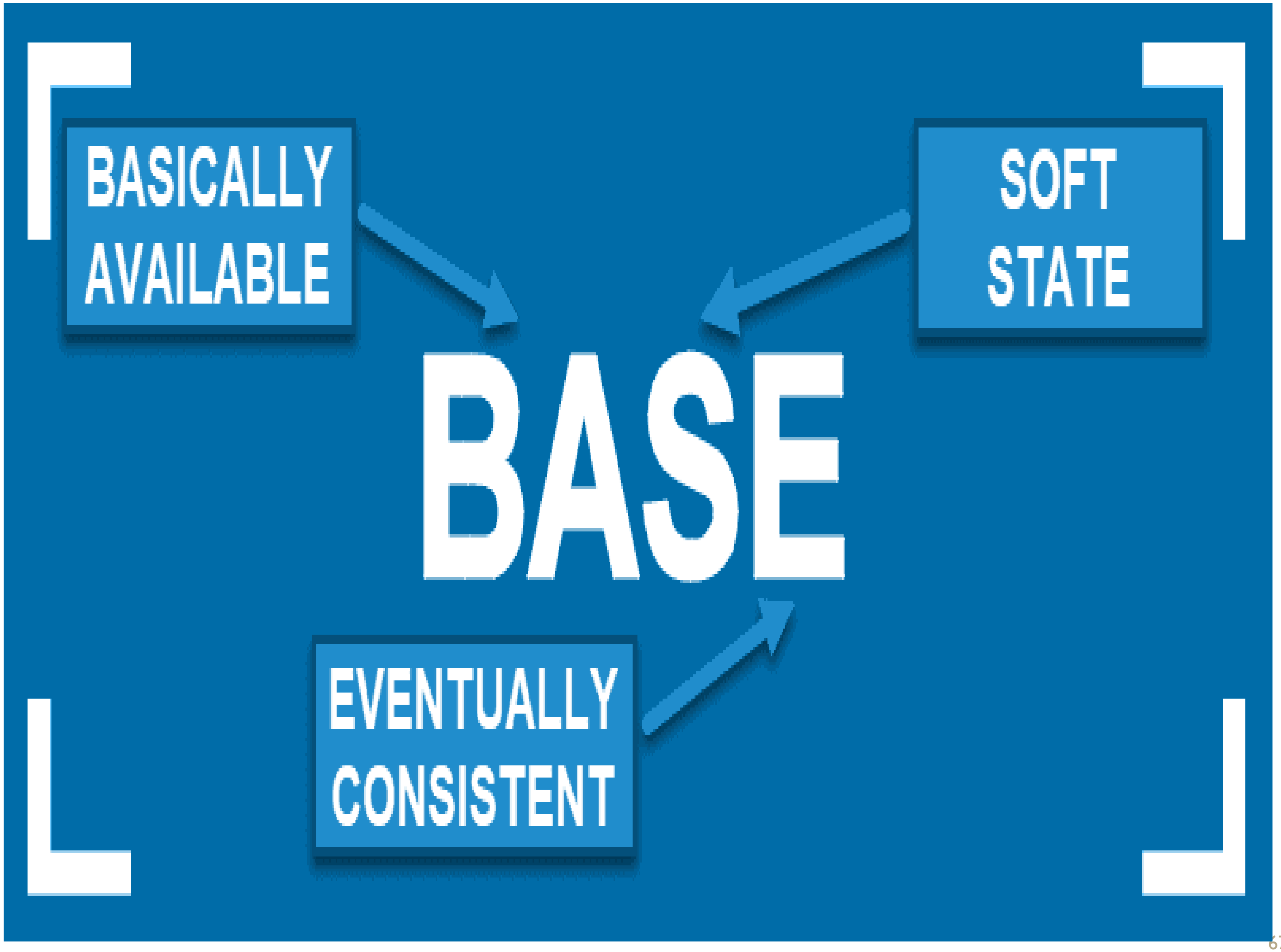
Id: "002",
Name: "Alex Bill",
Phone: {
Home: "+001 234 5341",
Office: "+001 111 2738"
}
Department: "Finance"

BASE Properties

Before going to BASE Properties, revise ACID Properties :


ACID Properties in DBMS
ACID
A = Atomicity
C = Consistency
I = Isolation
D = Durability
The entire transaction takes place at once or doesn't happen at all.
The database must be consistent before and after the transaction.
Multiple Transactions occur independently without interference.
The changes of a successful transaction occurs even if the system failure occurs.

BASE Properties



BASE Properties

NoSQL relies upon a softer model known, appropriately, as the **BASE model**. This model accommodates the **flexibility** offered by NoSQL and similar approaches to the management and curation of unstructured data.

BASE consists of three principles:

- **Basically Available:** The NoSQL database approach focuses on the **availability** of data even in the presence of **multiple failures**. It achieves this by using a highly distributed approach to database management.

BASE Properties

- **Soft state:** The state of the data could change without application interactions due to eventual consistency.
- One of the basic concepts behind BASE is that data consistency is the developer's problem and should not be handled by the database.

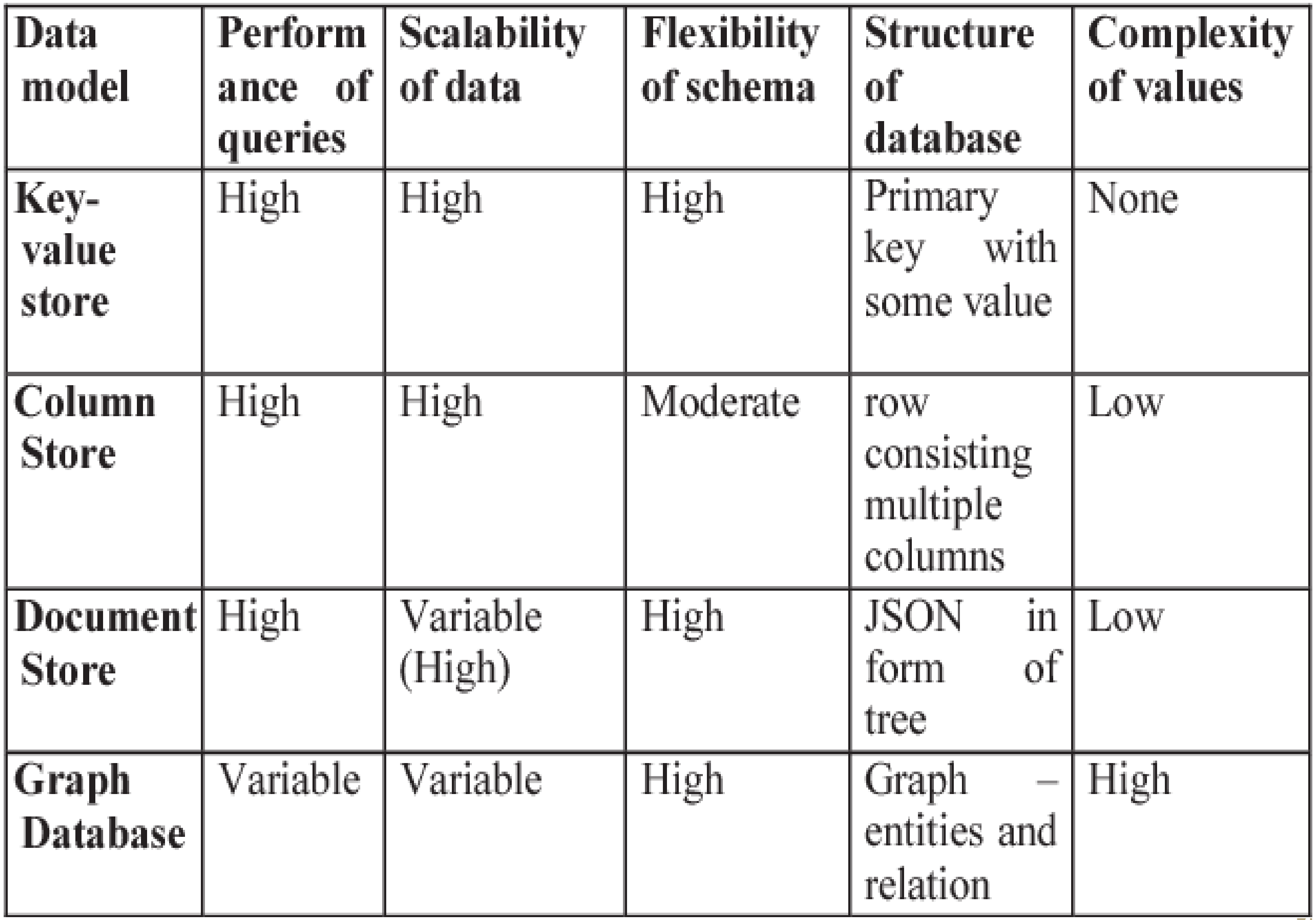
- **Eventual Consistency:** The only requirement that NoSQL databases have regarding consistency is to require that at some point in the future, data will converge to a consistent state. No guarantees are made, however, about when this will occur.

Types of NoSQL Data Models

In general, there are four different types of data models in NoSQL. They are as follows :

- Key-value Store
- Document-based store NoSQL
- Column based store
- Graph-based store

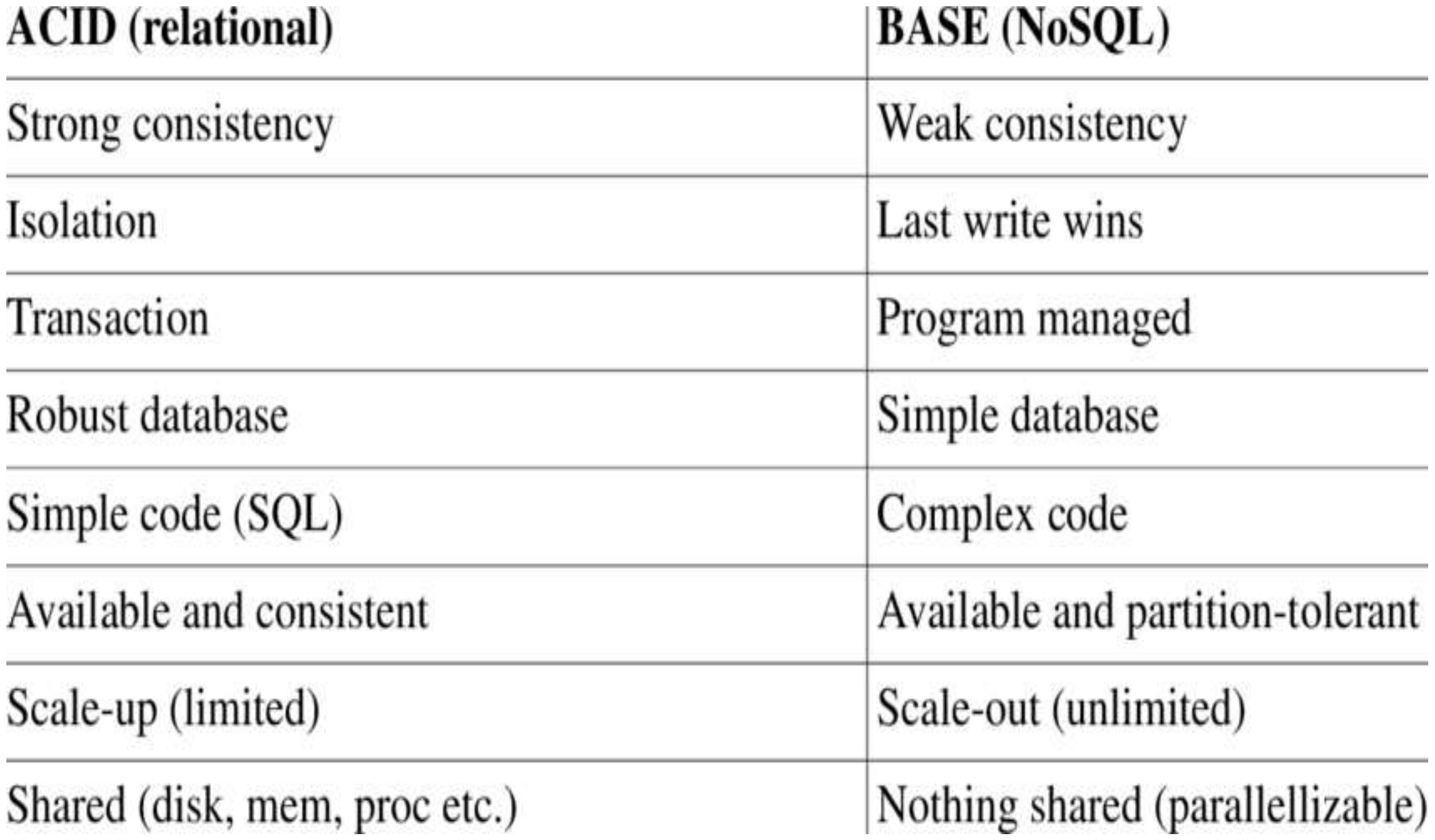
Comparison of NoSQL Data Models



Data model	Performance of queries	Scalability of data	Flexibility of schema	Structure of database	Complexity of values
Key-value store	High	High	High	Primary key with some value	None
Column Store	High	High	Moderate	row consisting multiple columns	Low
Document Store	High	Variable (High)	High	JSON in form of tree	Low
Graph Database	Variable	Variable	High	Graph – entities and relation	High

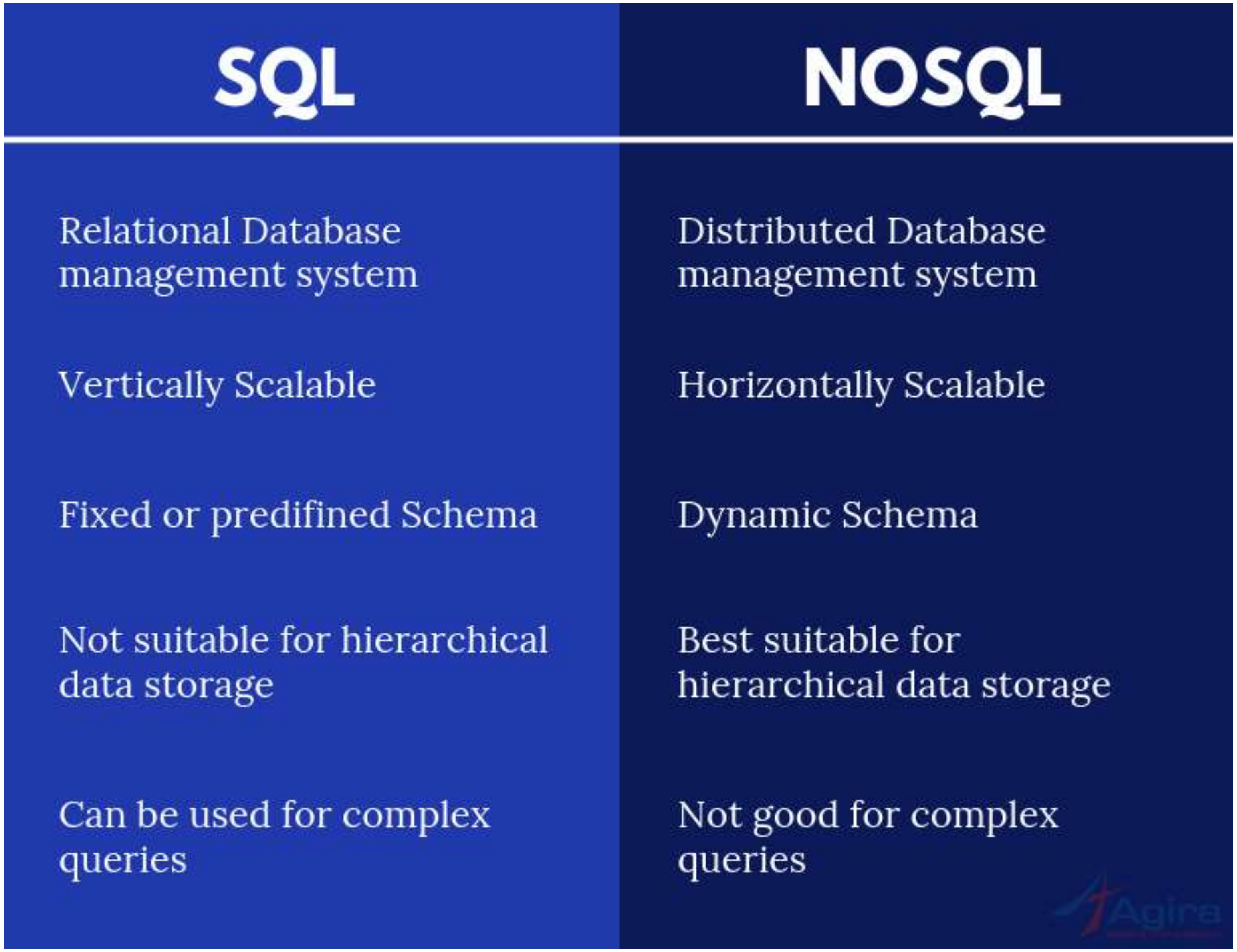
Data Consistency Models: ACID vs. BASE

The two most common consistency models are known by the acronyms ACID and BASE.



ACID (relational)	BASE (NoSQL)
Strong consistency	Weak consistency
Isolation	Last write wins
Transaction	Program managed
Robust database	Simple database
Simple code (SQL)	Complex code
Available and consistent	Available and partition-tolerant
Scale-up (limited)	Scale-out (unlimited)
Shared (disk, mem, proc etc.)	Nothing shared (parallellizable)

Difference between SQL and No SQL



SQL	NOSQL
Relational Database management system	Distributed Database management system
Vertically Scalable	Horizontally Scalable
Fixed or predifined Schema	Dynamic Schema
Not suitable for hierarchical data storage	Best suitable for hierarchical data storage
Can be used for complex queries	Not good for complex queries

NoSQL databases vs RDBMS

NoSQL Database	Relational Database
NoSQL Database supports a very simple query language.	Relational Database supports a powerful query language.
NoSQL Database has no fixed schema.	Relational Database has a fixed schema.
NoSQL Database is only eventually consistent.	Relational Database follows acid properties. (Atomicity, Consistency, Isolation, and Durability)
NoSQL databases don't support transactions (support only simple transactions).	Relational Database supports transactions (also complex transactions with joins).
NoSQL Database is used to handle data coming in high velocity.	Relational Database is used to handle data coming in low velocity.

NoSQL databases vs RDBMS

NoSQL Database	Relational Database
NoSQL database can manage structured, unstructured and semi-structured data.	Relational database manages only structured data.
NoSQL databases have no single point of failure.	Relational databases have a single point of failure with failover.
NoSQL databases can handle big data or data in a very high volume .	NoSQL databases are used to handle moderate volume of data.
NoSQL has decentralized structure.	Relational database has centralized structure.
NoSQL database gives both read and write scalability.	Relational database gives read scalability only.
NoSQL database is deployed in horizontal fashion.	Relation database is deployed in vertical fashion.

Mongo DB

- MongoDB is an open source NoSQL database. NoSQL means the database does not use relational tables like a traditional SQL database.

- There is a range of NoSQL database types, but MongoDB stores data in **JavaScript-like objects** known as **documents.**

- This not only simplifies database management for developers but also creates a **highly scalable** environment for cross-platform applications and services.

Features of Mongo DB

1. **Ad Hoc Queries Support :**
   MongoDB supports Ad Hoc Queries, it means that the MongoDB supports queries that were not known while establishing a structure for the database. This is an amazing feature that makes MongoDB stand out. Ad hoc queries function in a way to better the performance and are real-time.

2. **Indexing :**
   One of the most important features for a Database is Indexing, which results in improvements. We can index any field in MongoDB. Creating indexes helps in **faster search** results. Indexing is possible with any **field** or **key**, in a document.

Features of Mongo DB

3. Replication :

Simply speaking, replication is a process of, distributing the data on multiple servers and keeping everything synchronized. **Primary Node** and **Secondary Nodes** are introduced here.

Working: Whenever a primary node, with the data, is down or experiences failure, the secondary node will be the working primary node, making it possible for the data to be available.

This feature is important regarding Data Storage and Backup, as it allows us to recover and restore, in case of failure in hardware or services.

Features of Mongo DB

4. Load Balancing :

Proper load balancing is fundamental to database management for expansion in large-scale organizations. As client traffic and requests come in thousands and millions, they must be well distributed across the different servers to maximize performance and reduce over **congestion**. MongoDB efficiently handles read and write requests, **balancing** the incoming load across its **multiple servers** and ensuring **data consistency**. This means that with MongoDB, there is no need for additional load balancers.

Features of Mongo DB

5. Schema-less database :

MongoDB is a schema-less database meaning that it can consist of different types of documents in a single collection.

In MongoDB, one collection can have documents with different sizes, numbers of fields, and different content. This brings about a very **flexible database schema.**

Features of Mongo DB

6. Sharding :

When we are encountered with large datasets, we can implement the Sharding of Data. **Sharding** is the process of **database partitioning** and **spreading across multiple machines**, while the replication is the process of making multiple copies of the database. The data is distributed over multiple collections, and these collections are known as "**Shards**".

In a sharding environment, queries are sent to different shards using a **shard key**. This handling process is called **Mongos**. When done correctly, sharding leads to better **load balancing.**

Features of Mongo DB

7. GridFS :

When a **file** exceeds the BSON document, it is **divided into multiple small chunks** and stored separately in various documents. These chunks have a size of 255KB, excluding the last chunk.

GridFS, which stands for Grid File System, use **two separate collections**. One collection is used to store the larger **file's chunks**, while the second collection is used to store the **metadata**. When we execute a query for this file, the GridFS will collect and return all the chunks together. GridFS also implements the **Indexing**, which allows the query execution for returning the file easier.

Features of Mongo DB

8. **Aggregation Pipeline :**
   Aggregation Framework is one of the most efficient features offered by MongoDB. Simply speaking it is a process of creating a pipeline of multiple operations and getting a final result, filtered.

9. **High Performance :**
   MongoDB is open-source, is one the highest performing database. With the implementation of **replication and indexing**, query execution and data fetching are faster.

Advantages of MongoDB

- Schema less − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.

- Structure of a single object is clear.

- No complex joins.

- Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.

Advantages of MongoDB

- Ease of scale-out − MongoDB is easy to scale.
- Conversion/mapping of application objects to database objects not needed.
- Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Why Use MongoDB?

- Document Oriented Storage − Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication and high availability
- Auto-Sharding
- Rich queries
- Fast in-place updates
- Professional support by MongoDB

Where to Use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

MongoDB - Datatypes

MongoDB supports many datatypes. Some of them are −

- **String** − This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer** − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** − This type is used to store a boolean (true/ false) value.
- **Double** − This type is used to store floating point values.
- **Min/ Max keys** − This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays** − This type is used to store arrays or list or multiple values into one key.

MongoDB - Datatypes

MongoDB supports many datatypes. Some of them are −

- **Timestamp** − ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object** − This datatype is used for embedded documents.
- **Null** − This type is used to store a Null value.
- **Symbol** − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- **Date** − This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.

MongoDB - Datatypes

MongoDB supports many datatypes. Some of them are −

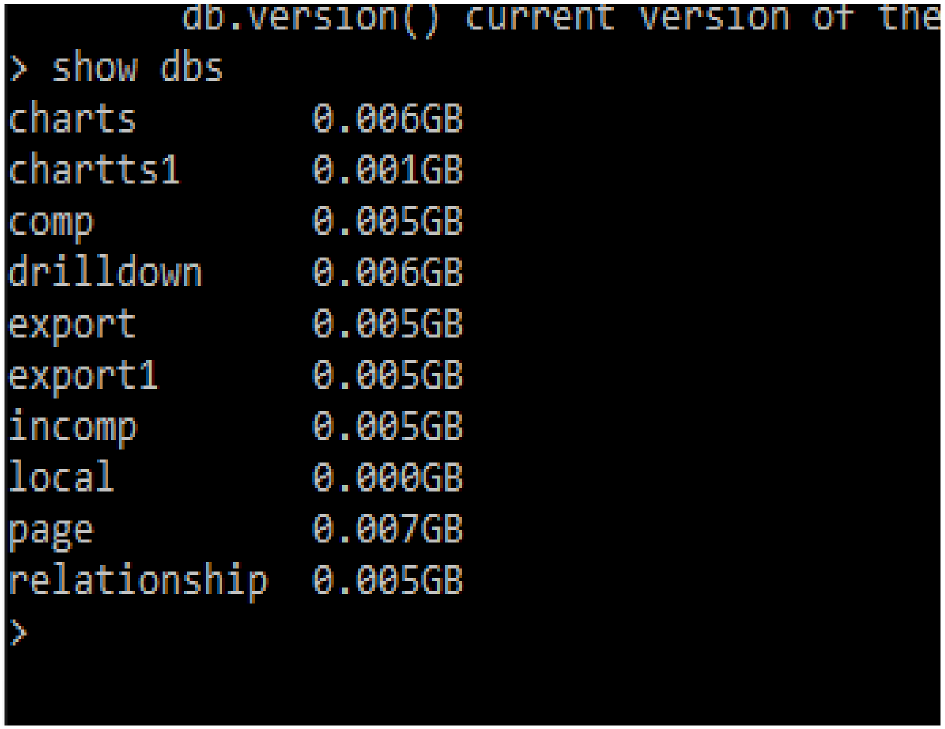
- **Object ID** − This datatype is used to store the document's ID.
- **Binary data** − This datatype is used to store binary data.
- **Code** − This datatype is used to store JavaScript code into the document.
- **Regular expression** − This datatype is used to store regular expression.

Basic commands of MongoDB

Show All Databases

Use below command to get list of all databases.

show dbs



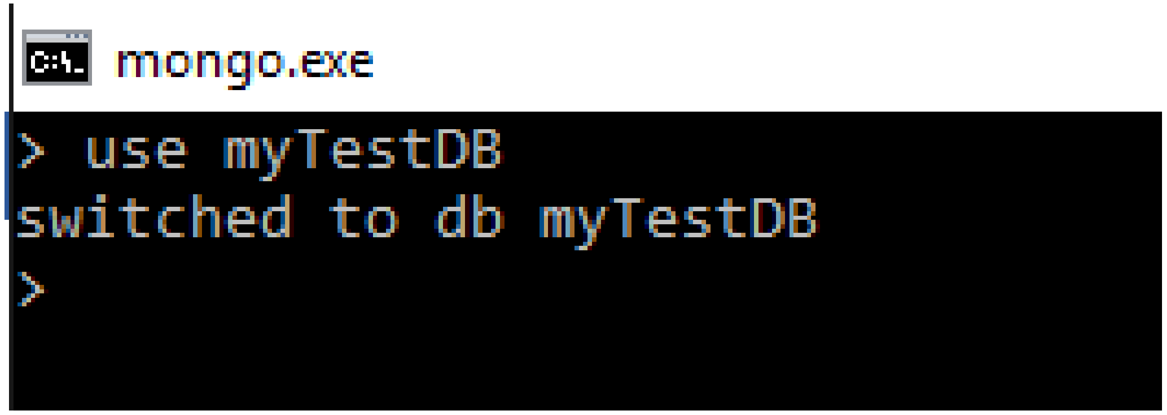
```
        db.version() current version of the
> show dbs
charts        0.006GB
charts1       0.001GB
comp          0.005GB
drilldown     0.006GB
export        0.005GB
export1       0.005GB
incomp        0.005GB
local         0.000GB
page          0.007GB
relationship  0.005GB
>
```

Basic commands of MongoDB

Create new database

To create a new database execute the following command.

use DATABASE_NAME



mongo.exe

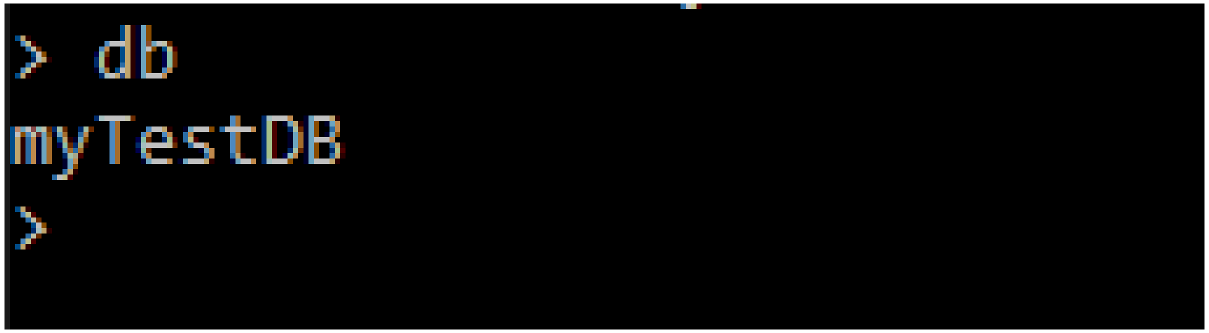
```
> use myTestDB
switched to db myTestDB
>
```

Basic commands of MongoDB

Know your current selected database

To know your current working/selected database execute the following command

db



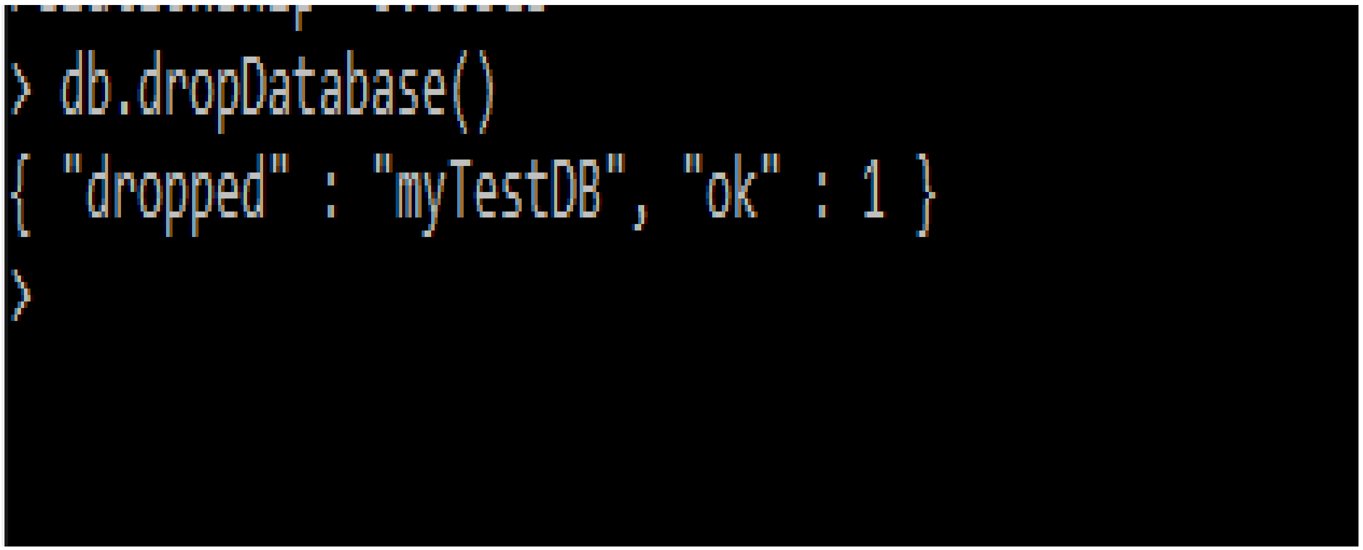
```
> db
myTestDB
>
```

Basic commands of MongoDB

Drop database :

To drop the database execute following command, this will drop the selected database

db.dropDatabase()



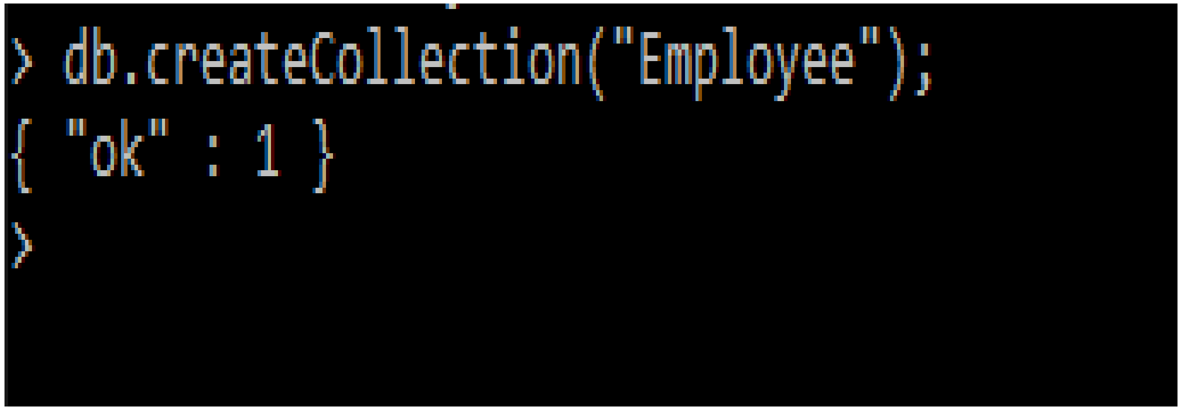
```
> db.dropDatabase()
{ "dropped" : "myTestDB", "ok" : 1 }
>
```

Basic commands of MongoDB

Create collection

To create the new collection execute the following commands

db.createCollection(name)



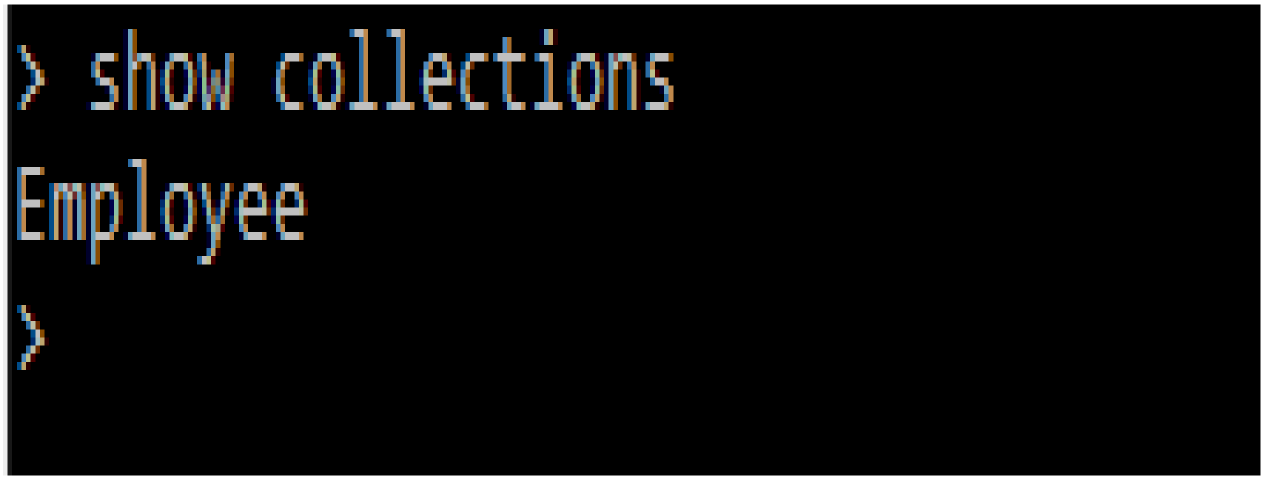
```
> db.createCollection("Employee");
{ "ok" : 1 }
>
```

Basic commands of MongoDB

To check collections list

To get the list of collections created execute the following command

Show collections



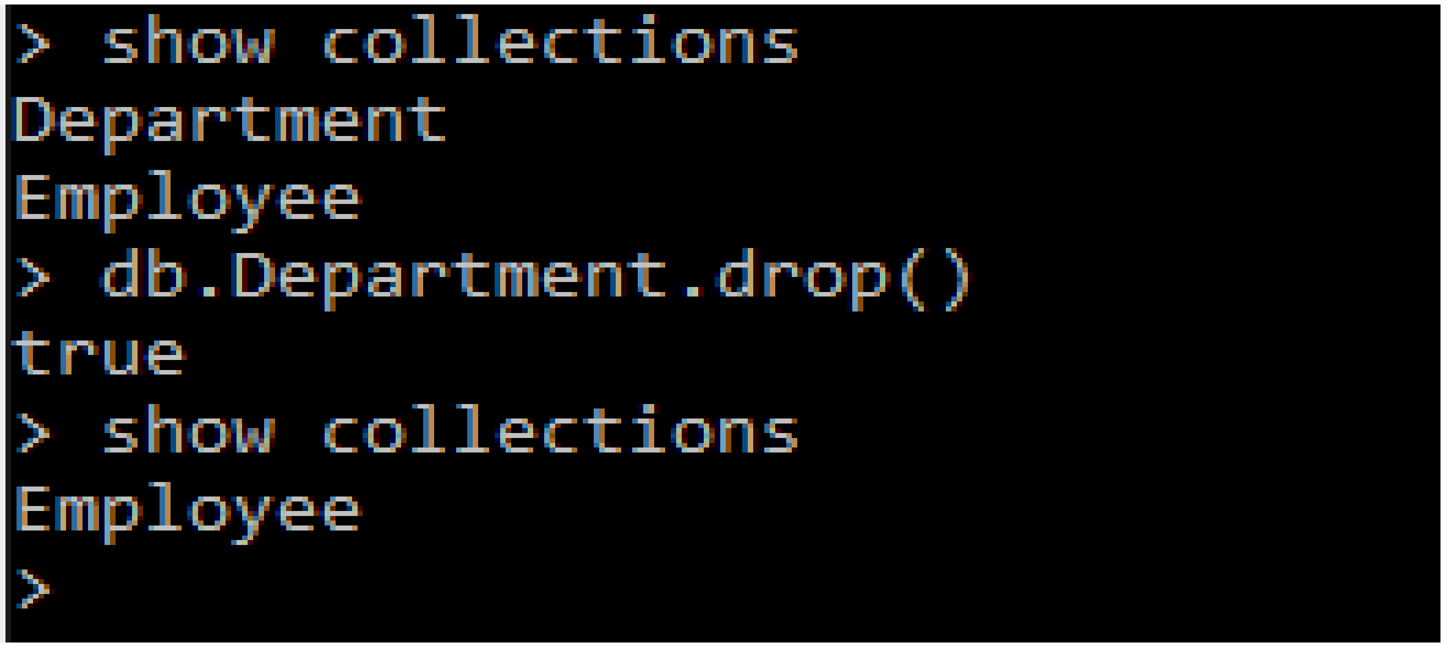
```
> show collections
Employee
>
```

Basic commands of MongoDB

Drop collection

To drop the selected collection execute the following command

db.COLLECTION_NAME.drop()



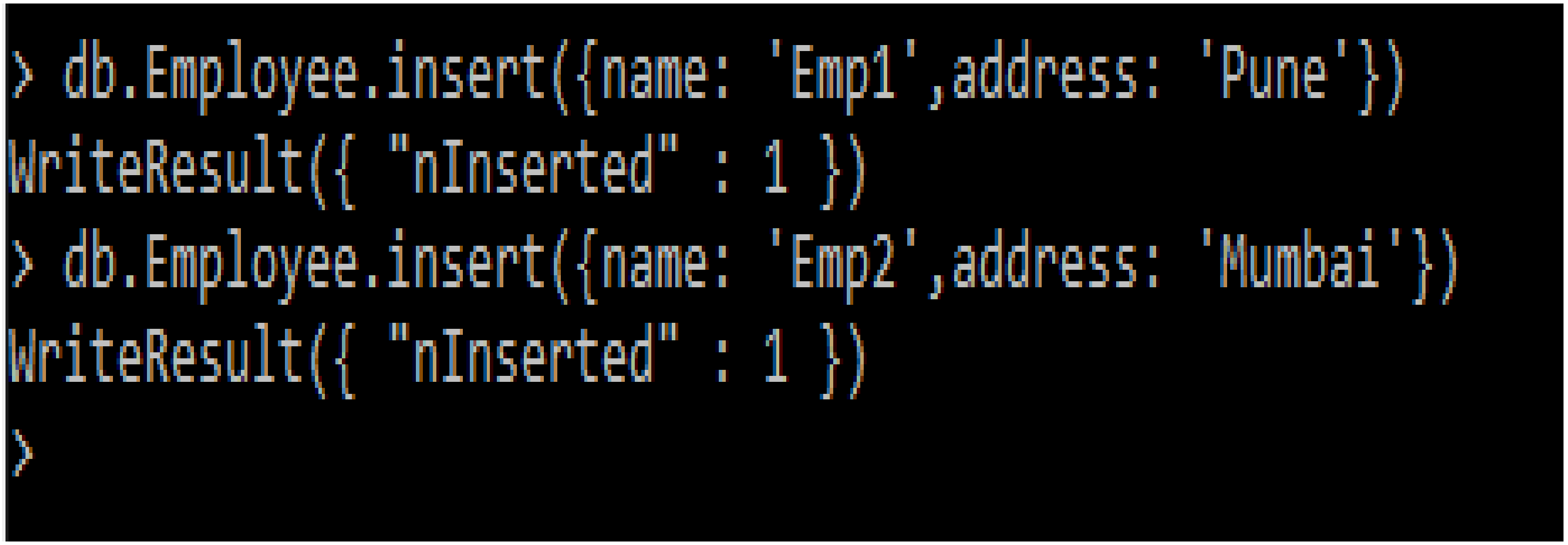
```
> show collections
Department
Employee
> db.Department.drop()
true
> show collections
Employee
>
```

Basic commands of MongoDB

Insert document in collection

To insert single document in selected collection execute the following command

>db.COLLECTION_NAME.insert(document)

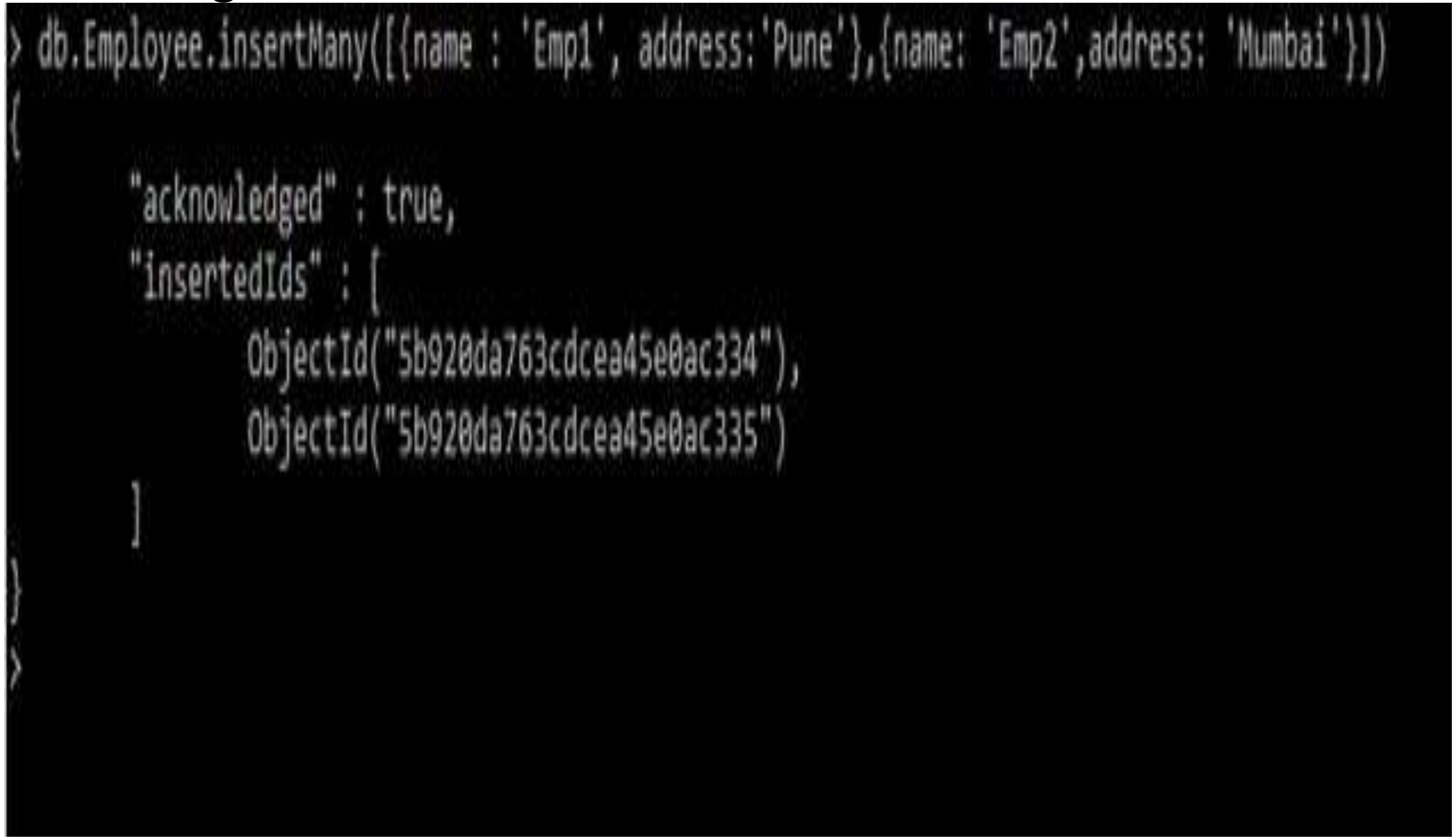


```
> db.Employee.insert({name: 'Emp1',address: 'Pune'})
WriteResult({ "nInserted" : 1 })
> db.Employee.insert({name: 'Emp2',address: 'Mumbai'})
WriteResult({ "nInserted" : 1 })
>
```

Basic commands of MongoDB

Insert Multiple document in collection

To insert multiple documents in selected collection execute following command



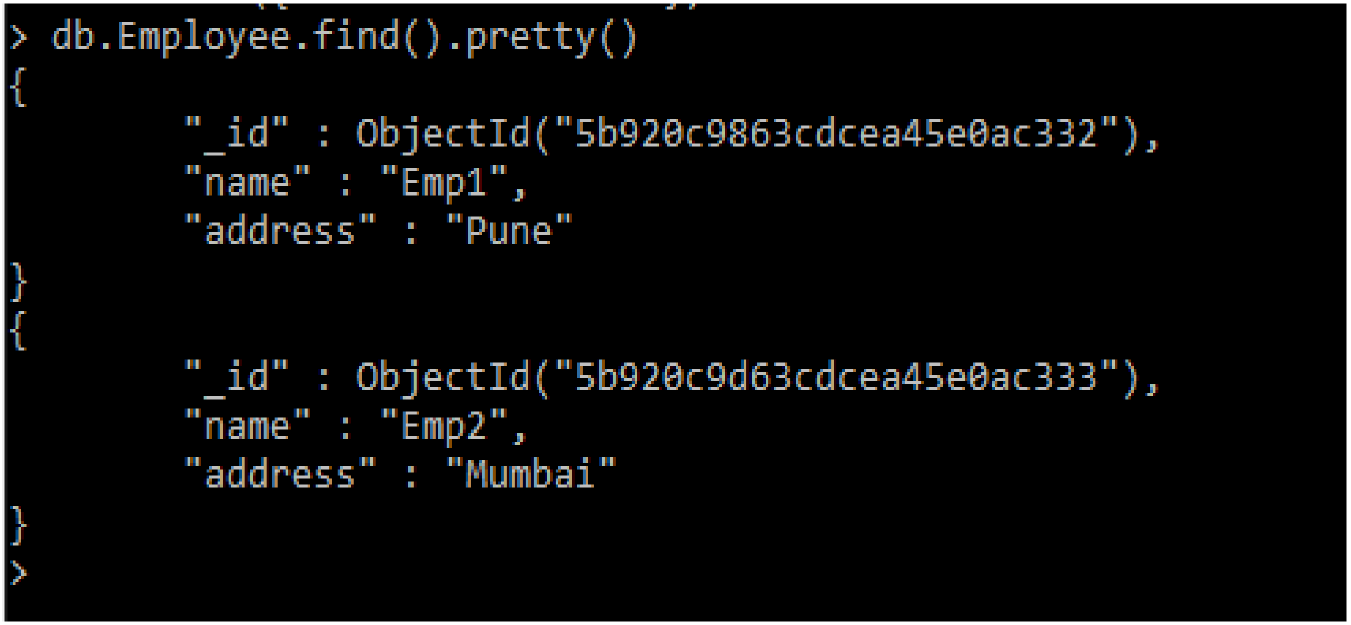
```
> db.Employee.insertMany([{name : 'Emp1', address:'Pune'},{name: 'Emp2',address: 'Mumbai'}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5b920da763cdcea45e0ac334"),
                ObjectId("5b920da763cdcea45e0ac335")
        ]
}
>
```

Basic commands of MongoDB

Get collection document

To get the list documents in collection execute the following command

>db.COLLECTION_NAME.find()



```
> db.Employee.find().pretty()
{
        "_id" : ObjectId("5b920c9863cdcea45e0ac332"),
        "name" : "Emp1",
        "address" : "Pune"
}
{
        "_id" : ObjectId("5b920c9d63cdcea45e0ac333"),
        "name" : "Emp2",
        "address" : "Mumbai"
}
>
```

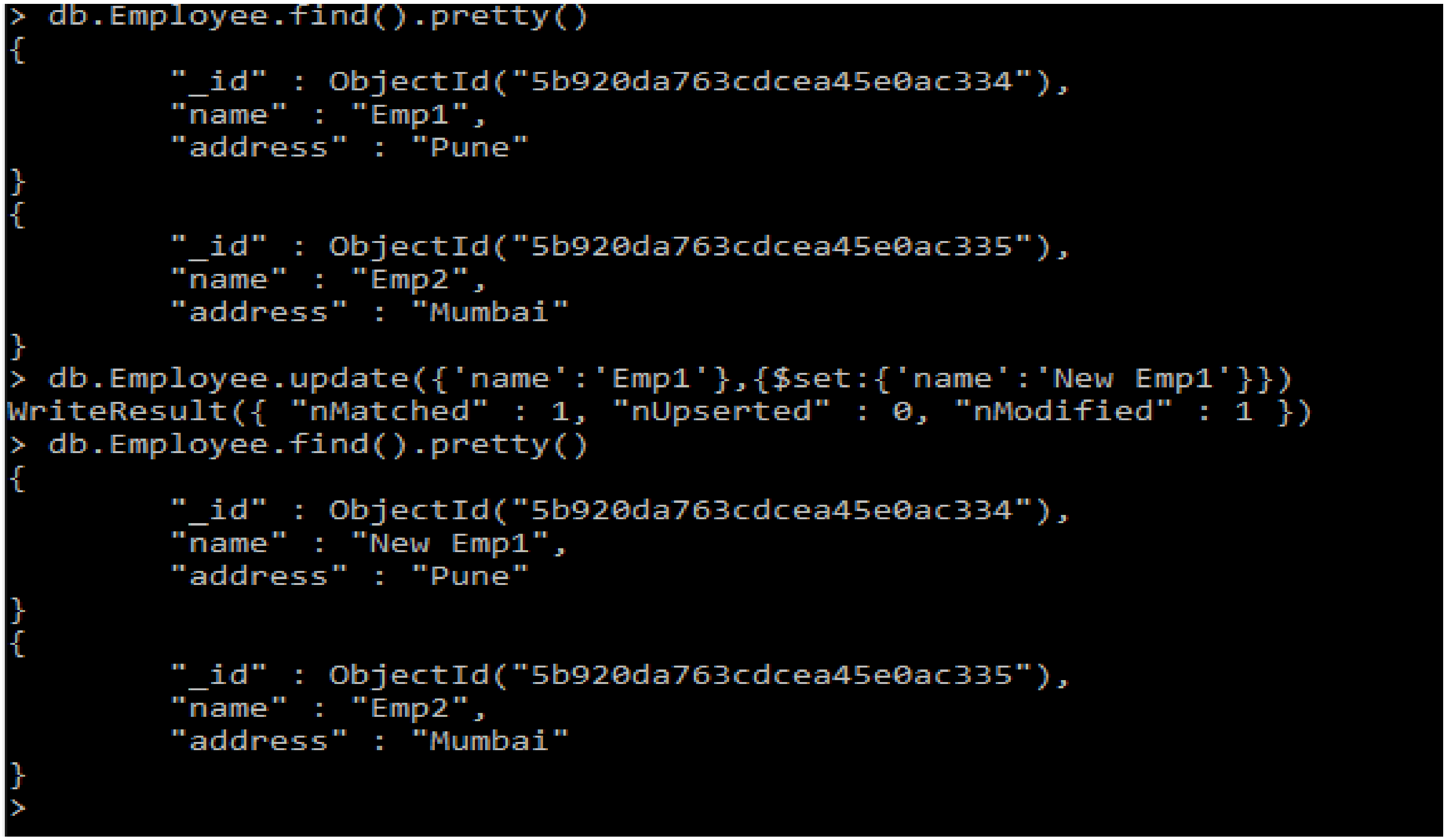
Notice that we use pretty() to format the output to JSON so that it is easier to read.

Basic commands of MongoDB

Update document

To update the document in collection execute the following command

>db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)



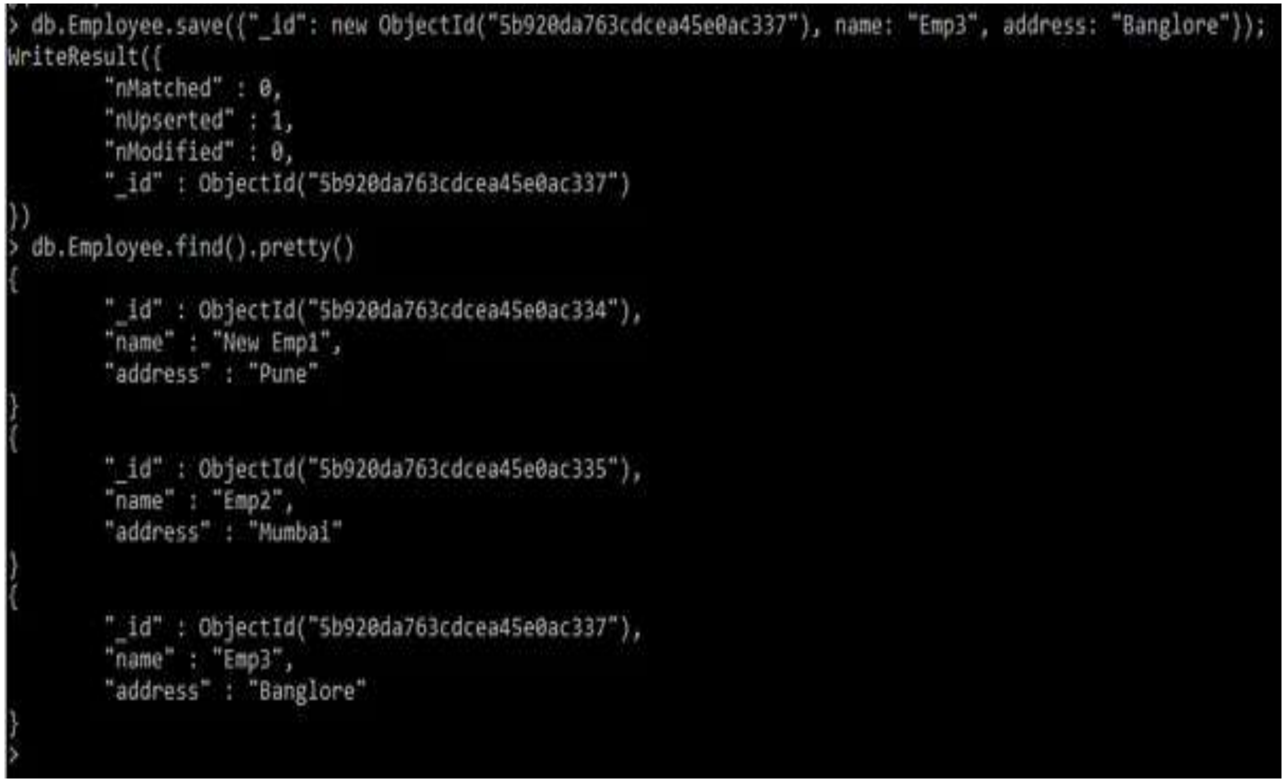
```
> db.Employee.find().pretty()
{
        "_id" : ObjectId("5b920da763cdcea45e0ac334"),
        "name" : "Emp1",
        "address" : "Pune"
}
{
        "_id" : ObjectId("5b920da763cdcea45e0ac335"),
        "name" : "Emp2",
        "address" : "Mumbai"
}
> db.Employee.update({'name':'Emp1'},{$set:{'name':'New Emp1'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.Employee.find().pretty()
{
        "_id" : ObjectId("5b920da763cdcea45e0ac334"),
        "name" : "New Emp1",
        "address" : "Pune"
}
{
        "_id" : ObjectId("5b920da763cdcea45e0ac335"),
        "name" : "Emp2",
        "address" : "Mumbai"
}
>
```

Basic commands of MongoDB

Save document

To save document in collection execute the following command

>db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})



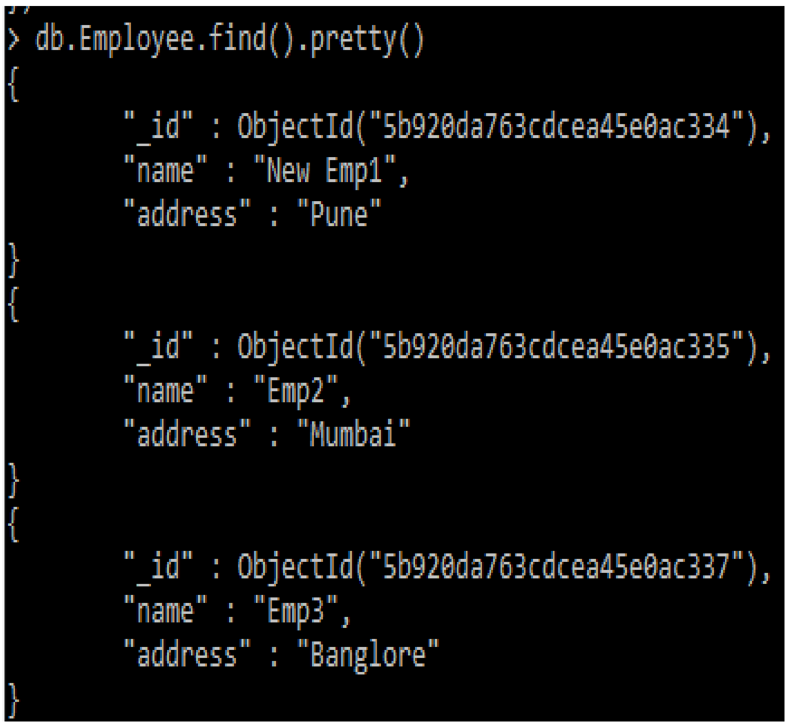
```
> db.Employee.save({"_id": new ObjectId("5b920da763cdcea45e0ac337"), name: "Emp3", address: "Banglore"});
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("5b920da763cdcea45e0ac337")
})
> db.Employee.find().pretty()
{
        "_id" : ObjectId("5b920da763cdcea45e0ac334"),
        "name" : "New Emp1",
        "address" : "Pune"
}
{
        "_id" : ObjectId("5b920da763cdcea45e0ac335"),
        "name" : "Emp2",
        "address" : "Mumbai"
}
{
        "_id" : ObjectId("5b920da763cdcea45e0ac337"),
        "name" : "Emp3",
        "address" : "Banglore"
}
>
```

Basic commands of MongoDB

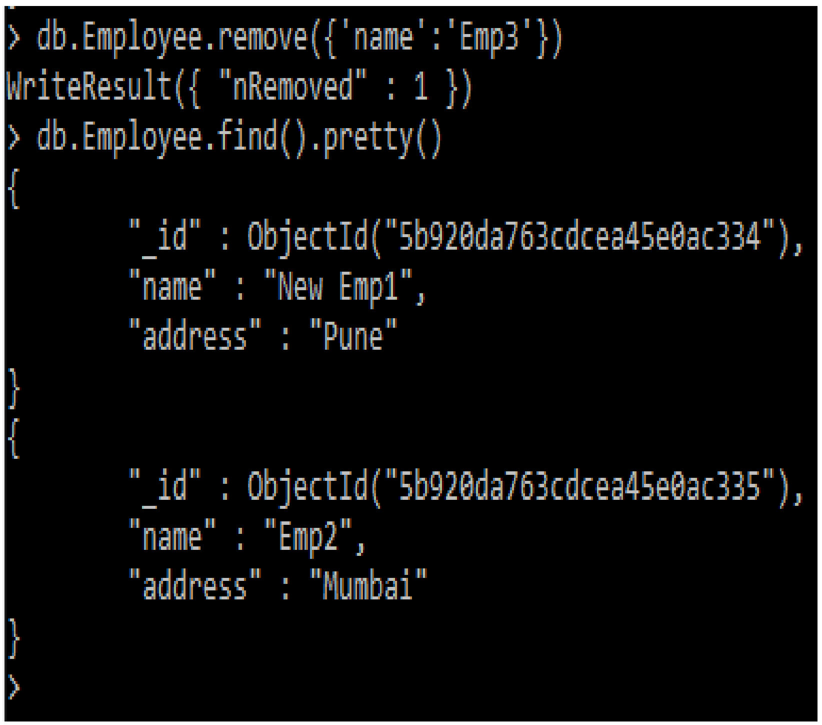
Delete document

To delete document in selected collection execute the following command

>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)



```
> db.Employee.find().pretty()
{
        "_id" : ObjectId("5b920da763cdcea45e0ac334"),
        "name" : "New Emp1",
        "address" : "Pune"
}
{
        "_id" : ObjectId("5b920da763cdcea45e0ac335"),
        "name" : "Emp2",
        "address" : "Mumbai"
}
{
        "_id" : ObjectId("5b920da763cdcea45e0ac337"),
        "name" : "Emp3",
        "address" : "Banglore"
}
```



```
> db.Employee.remove({'name':'Emp3'})
WriteResult({ "nRemoved" : 1 })
> db.Employee.find().pretty()
{
        "_id" : ObjectId("5b920da763cdcea45e0ac334"),
        "name" : "New Emp1",
        "address" : "Pune"
}
{
        "_id" : ObjectId("5b920da763cdcea45e0ac335"),
        "name" : "Emp2",
        "address" : "Mumbai"
}
>
```

Basic commands of MongoDB

The sort() Method :

To sort documents in MongoDB, you need to use sort() method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. **1** is used for **ascending order** while **-1** is used for **descending order.**

Syntax :

>db.COLLECTION_NAME.find().sort({KEY:1})

Basic commands of MongoDB

The sort() Method :

Example :

Consider the collection myycol has the following data.

```
{_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"}
{_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"}
{_id : ObjectId("507f191e810c19729de860e3"), title: "Tutorials Point Overview"}
```

Following example will display the documents sorted by title in the descending order.

```
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Tutorials Point Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>
```

MongoDB - INDEXING

Indexing :

Indexes are special data structures, that store a small portion of the data set in an easy-to-traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

The createIndex() Method

To create an index, you need to use createIndex() method of MongoDB.

Syntax

>db.COLLECTION_NAME.createIndex({KEY:1})

Here key is the name of the field on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

MongoDB - INDEXING

Indexing :

The createIndex() Method

Example :

```
>db.mycol.createIndex({"title":1})
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

MongoDB - INDEXING

Indexing :

The ensureIndex() Method :

The **ensureIndex()** method is a method in MongoDB used to create indexes. This method creates a new index on the collection and if the index already exists, no operation is performed. The method can be used for both single-field and multi-field indexes.

Syntax :

db.collection.ensureIndex(keys, options)

MongoDB - INDEXING

Indexing :

The ensureIndex() Method :

The **ensureIndex()** method is particularly useful in the following scenarios:

- When you want to ensure that a field or multiple fields are indexed.
- When you want to create an index but are unsure if the index already exists.

MongoDB - INDEXING

Indexing :

The ensureIndex() Method :

Suppose we have a collection named users that contains the following documents:

```
{ "_id" : ObjectId("61e07a8f1e4c7d67dc65a3c3"), "name":"Alice", "age": 25}
{ "_id" : ObjectId("61e07a9a1e4c7d67dc65a3c4"), "name":"Bob", "age":30 }
{"_id":ObjectId("61e07aa21e4c7d67dc65a3c5"),"name":"Charlie", "age":35}
```

We can use the ensureIndex() method to create an index on the name field as follows:

db.users.ensureIndex({ name: 1 })

This method creates an index on the name field in the users collection. If the index already exists, no operation is performed.

MongoDB - INDEXING

Indexing :

The dropIndex() method

You can drop a particular index using the dropIndex() method of MongoDB.

Syntax

>db.COLLECTION_NAME.dropIndex({KEY:1})

Here, "key" is the name of the file on which you want to remove an existing index. Instead of the index specification document (above syntax), you can also specify the name of the index directly as:

dropIndex("name_of_the_index")

Here key is the name of the field on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

MongoDB - Aggregation

Aggregation :

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In SQL count(*) and with group by is an equivalent of MongoDB aggregation.

The aggregate() Method :

For the aggregation in MongoDB, you should use aggregate() method.

Syntax

>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

MongoDB - Aggregation

Aggregation :

Example :Sample data in collection −

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
},
{
   _id: ObjectId(7df78ad8902d)
   title: 'NoSQL Overview',
   description: 'No sql database is very fast',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 10
},
{
   _id: ObjectId(7df78ad8902e)
   title: 'Neo4j Overview',
   description: 'Neo4j is no sql database',
   by_user: 'Neo4j',
   url: 'http://www.neo4j.com',
   tags: ['neo4j', 'database', 'NoSQL'],
   likes: 750
},
```

MongoDB - Aggregation

Aggregation :

Now from the above collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following aggregate() method −

```
>db.mycol.aggregate([{$group:{_id:"$by_user",
num_tutorial : {$sum : 1}}}])
```

```
{ "_id" : "tutorials point", "num_tutorial" : 2 }
{ "_id" : "Neo4j", "num_tutorial" : 1 }
```

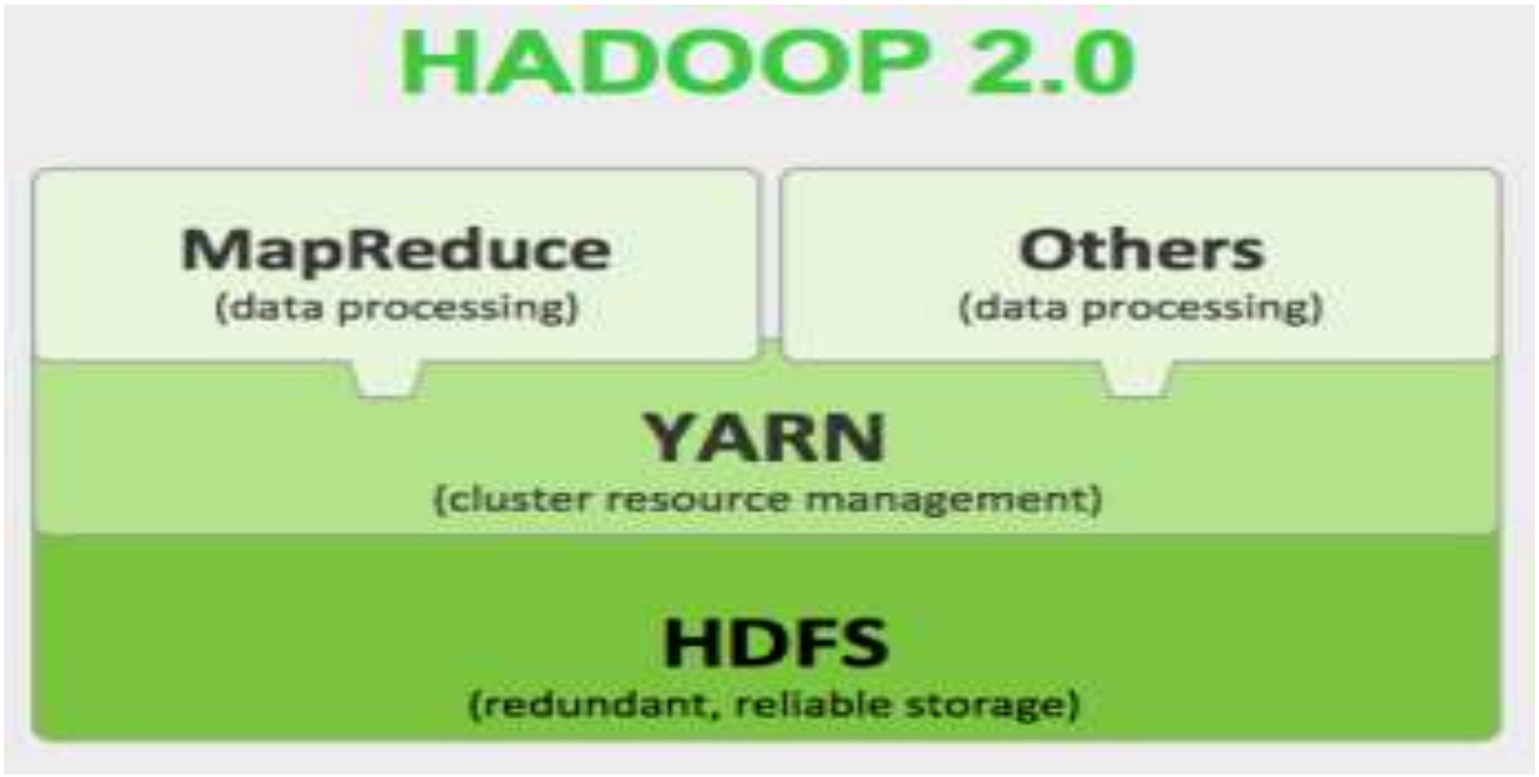
HADOOP

- Hadoop is an open source distributed processing framework that manages data processing and storage for **big data** applications in scalable clusters of computer servers using simple programming models.
- Hadoop is designed to scale up from a single computer to thousands of clustered computers, with each machine offering local computation and storage.
- Hadoop is written in **Java** and is not OLAP (online analytical processing). It is used for batch/offline processing. It is being used by Facebook, Yahoo, Google, Twitter, LinkedIn and many more. Moreover it can be scaled up just by adding nodes in the cluster.
- Hadoop systems can handle various forms **of structured, semistructured and unstructured data**, giving users more flexibility for collecting, managing and analyzing data than relational databases and data warehouses provide.

HADOOP – Modules (Components)

Hadoop includes the following four modules as its primary components:

- **Hadoop Distributed File System (HDFS)**
- **Hadoop YARN**
- **Hadoop MapReduce**
- **Hadoop Common**



HADOOP - Modules

Hadoop Distributed File System (HDFS) :

- HDFS is a distributed file system designed to run on low-cost hardware. HDFS outperforms conventional file systems in terms of data throughput. It’s a Java-based distributed file system that lets you store large amounts of data across multiple nodes (machines).

- It's the primary data storage system for Hadoop applications and also manages access to the data in clusters. HDFS is built around an architecture of **NameNodes** and **DataNodes**. Each cluster includes one NameNode, a master node that manages the file system's namespace and controls file access, plus multiple DataNodes that manage storage on the servers that make up the cluster.

HADOOP - Modules

Hadoop YARN :

- **Y**et **A**nother **R**esource **N**egotiator, but typically referred to by the acronym alone, YARN is Hadoop's cluster resource management and job scheduling technology.
- In Hadoop clusters, YARN sits between HDFS and the processing engines deployed by users. It uses a combination of containers, application coordinators and node-level monitoring agents to dynamically allocate cluster resources to applications and oversee the execution of processing jobs.
- YARN supports multiple job scheduling approaches, including a first-in-first-out queue and several methods that schedule jobs based on assigned cluster resources.

HADOOP - Modules

MapReduce :

- This is a built-in programming framework and processing engine for running batch applications that group together various transactions or processes.
- As its name indicates, MapReduce uses map and reduce functions to split processing jobs into multiple tasks that run at the cluster nodes where data is stored and then to combine what each task produces into a coherent set of results.
- It supports parallel processing of large data sets across Hadoop clusters in a fault-tolerant way.

HADOOP - Modules

Hadoop Common :

- A set of shared utilities and libraries, Hadoop Common supports the other modules by providing various cluster configuration, management and security features.
- Examples include a key management server, a secure mode for user authentication and access, a registry of application services running in a cluster and a service-level authorization mechanism to help ensure that clients have permission to access particular services.

Advantages of Hadoop :

- **Fast:** In HDFS the data distributed over the cluster and are mapped which helps in **faster retrieval**. Even the tools to process the data are often on the **same servers**, thus **reducing the processing time**. It is able to process terabytes of data in minutes and Peta bytes in hours.
- **Scalable:** Hadoop cluster can be extended by just adding nodes in the cluster.
- **Cost Effective:** Hadoop is open source and uses commodity hardware to store data so it really cost effective as compared to traditional relational database management system.
- **Resilient to failure:** HDFS has the property with which it can replicate data over the network, so if one node is down or some other network failure happens, then Hadoop takes the other copy of data and use it.

HADOOP

Disadvantages of Hadoop :

Issue With Small Files :

- Hadoop is suitable for a small number of large files but when it comes to the application which deals with a large number of small files, Hadoop fails here.

Vulnerable By Nature :

- Hadoop is written in **Java which is a widely used programming language** hence it is easily exploited by cyber criminals which makes Hadoop vulnerable to security breaches.

Processing Overhead :

- In Hadoop, Hadoop cannot do in-memory calculations hence it incurs processing overhead.

HADOOP

Disadvantages of Hadoop :

Supports Only Batch Processing :

- At the core, Hadoop has a batch processing engine which is not efficient in stream processing. It cannot produce output in real-time with low latency. It only works on data which we collect and store in a file in advance before processing.

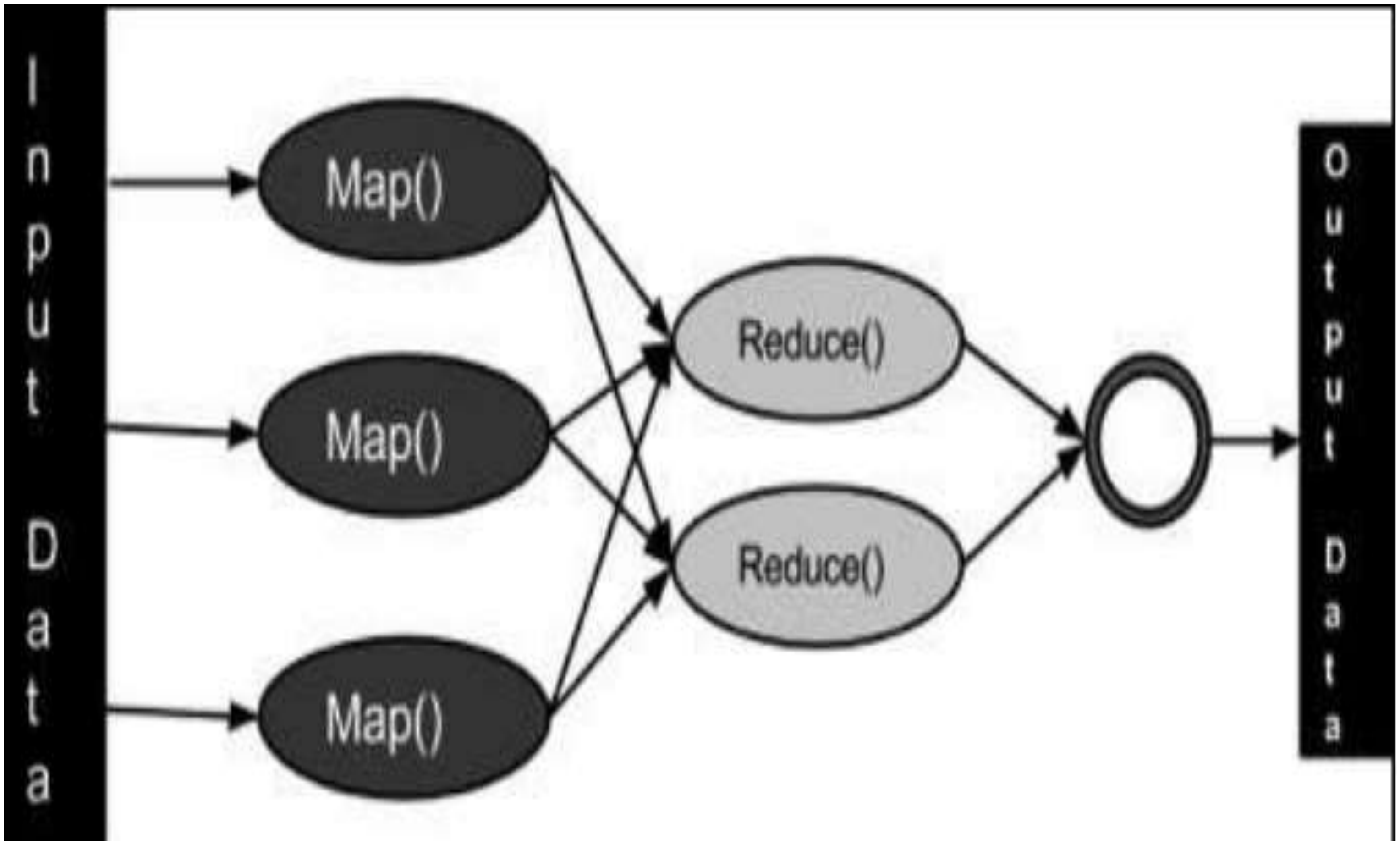
Iterative Processing :

- Hadoop cannot do iterative processing by itself. **Machine learning** or iterative processing has a cyclic data flow whereas Hadoop has data flowing in a chain of stages where output on one stage becomes the input of another stage.

HADOOP - MapReduce

- MapReduce is a processing technique and a program model for distributed computing based on java.
- The MapReduce algorithm contains two important tasks, namely **Map and Reduce**. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples **(key/value pairs).**
- Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

HADOOP - MapReduce



HADOOP - MapReduce

MapReduce program executes in three stages, namely **map stage, shuffle stage, and reduce stage.**

Map stage − The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.

Reduce stage − This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

HADOOP - MapReduce

Hadoop divides the job into tasks. There are two types of tasks:

Map tasks (Splits & Mapping)

Reduce tasks (Shuffling, Reducing)

as mentioned above.

The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called a

1. **Jobtracker:** Acts like a master (responsible for complete execution of submitted job)

2. **Multiple Task Trackers:** Acts like slaves, each of them performing the job.

For every job submitted for execution in the system, there is one **Jobtracker** that resides on **Namenode** and there are **multiple tasktrackers** which reside on **Datanode**.

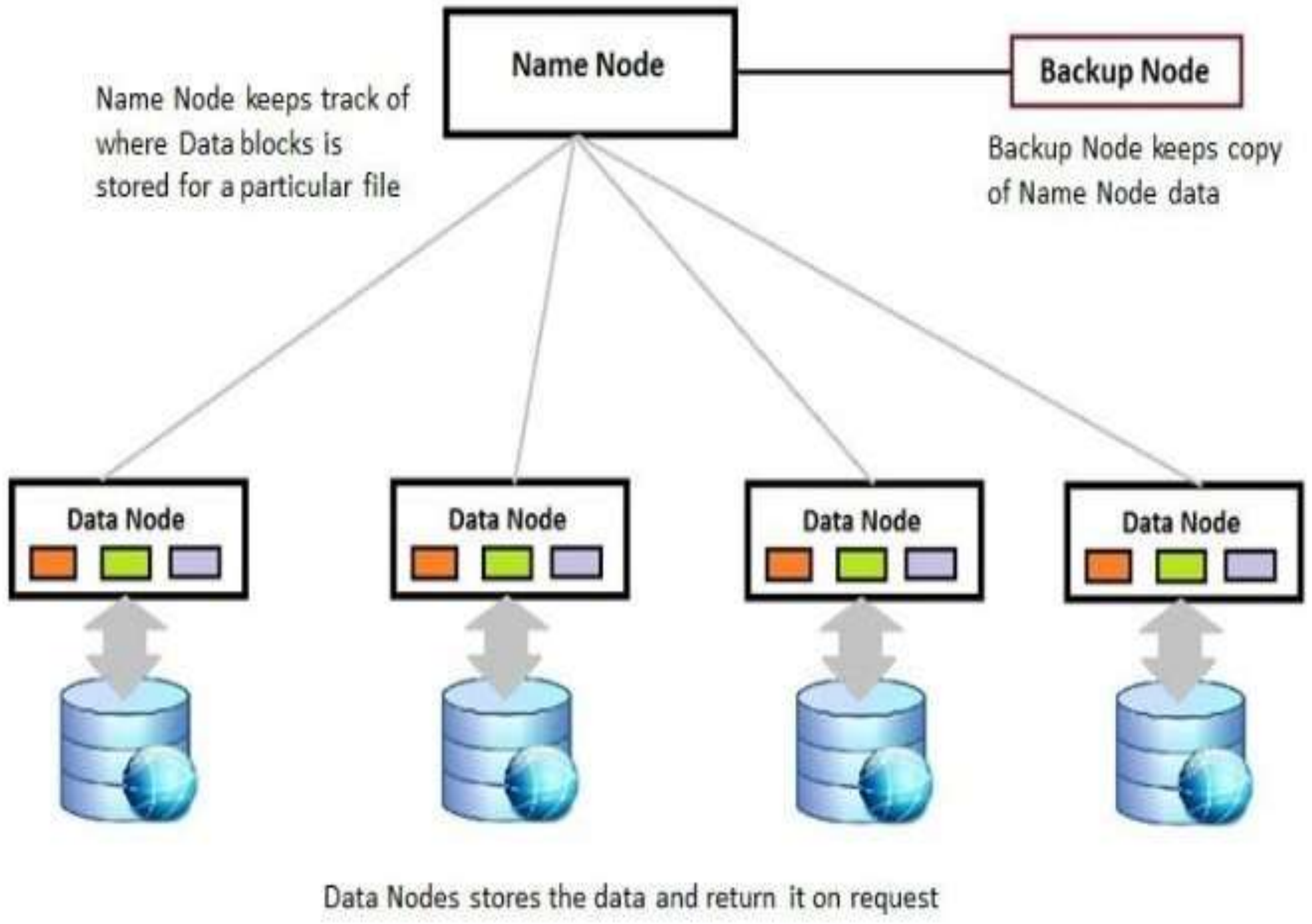
HADOOP - HDFS

- HDFS (Hadoop Distributed File System) is the primary storage system used by Hadoop applications. This open source framework works by rapidly transferring data between nodes. It's often used by companies who need to **handle and store big data.**

- HDFS is a key component of many Hadoop systems, as it provides a means for managing big data, as well as supporting **big data analytics.**

- HDFS is **fault-tolerant** and designed to be deployed on **low-cost, commodity** hardware. HDFS provides high throughput data access to application data and is suitable for applications that have large data sets and enables streaming access to file system data.

HADOOP - HDFS


Name Node
Backup Node
Name Node keeps track of where Data blocks is stored for a particular file
Backup Node keeps copy of Name Node data
Data Node
Data Node
Data Node
Data Node
Data Nodes stores the data and return it on request

HADOOP - HDFS

HDFS is composed of master-slave architecture, which includes the following elements:

NameNode :

All the blocks on DataNodes are handled by NameNode, which is known as the master node.

Secondary NameNode :

When NameNode runs out of disk space, a secondary NameNode is activated to perform a checkpoint..

DataNode

Every slave machine that contains data organsises a DataNode. DataNode stores data in **ext3 or ext4** file format on DataNodes.

Backup Node

Backup nodes are used to provide high availability of the data.

HADOOP - HDFS

Features of HDFS :

- HDFS can be configured to create **multiple replicas** for a particular file. If any one replica fails, the user can still access the data from other replicas.

- **Horizontal scalability** means that the data stored on multiple nodes can be stored in a single file system. Vertical scalability means that data can be stored on multiple nodes. Data can be replicated to ensure data integrity.

- Data is stored on HDFS, not on the local filesystem of your computer. In the event of a failure, the data is stored on a separate server, and can be accessed by the application running on your local computer. Data is replicated on multiple servers to ensure that even in the event of a server failure, your data is still accessible.

HADOOP - HDFS

Advantages of HDFS :

- It is a **highly scalable** data storage system. This makes it ideal for data-intensive applications like Hadoop and streaming analytics. Another major benefit of Hadoop is that it is **easy to set up**. This makes it ideal for non-technical users.
- It is very easy to implement, yet very robust. There is a lot of **flexibility** you get with Hadoop. It is a **fast** and **reliable** file system.
- This makes Hadoop a great fit for a wide range of data applications. The most common one is analytics. You can use Hadoop to process large amounts of data quickly, and then analyze it to find trends or make recommendations.
- You can increase the size of the cluster by adding more nodes or increase the size of the cluster by adding more nodes.

HADOOP - HDFS

Advantages of HDFS :

- Specialization reduces the overhead of data movement across the cluster and provides high availability of data.

- Automatic data replication can be accomplished with a variety of technologies, including RAID, Hadoop, and database replication. Logging data and monitoring it for anomalies can also help to detect and respond to hardware and software failures.

Assignment No : 5

1. Explain following NOSQL database types with examples and also state the scenario where it is useful
   - i) Column-oriented
   - ii) Graph
   - iii) Document -oriented
2. Explain CAP theorem and BASE properties
3. Describe distributed database. Explain System architecture of distributed transaction.
4. Explain following types of data with example
   - i) Structured
   - ii) Semi-structured
   - iii) Unstructured
5. Explain how NOSQL databases are different than relational databases? Describe in detail the key value store NOSQL data model with example.

THANK YOU…!!!