3. Big Data Processing

1. Big Data Ecosystem:

- The **Big Data Ecosystem** refers to the broad collection of technologies and tools that work together to capture, store, process, analyze, and visualize large volumes of diverse data.
- This ecosystem is essential for extracting valuable insights from massive datasets that traditional data-processing tools cannot handle efficiently.

🔁 Key Components of Big Data Ecosystem

Data Sources – Data comes in structured (RDBMS), semi-structured (XML, JSON), and unstructured (images, videos, social media) formats.

Data Ingestion – Tools like Kafka and Flume bring data from sources into storage systems.

Storage Layer – Data is stored in distributed systems like HDFS or NoSQL databases like MongoDB.

Data Processing – Batch (Spark, MapReduce) and stream (Flink, Storm) tools process the data.

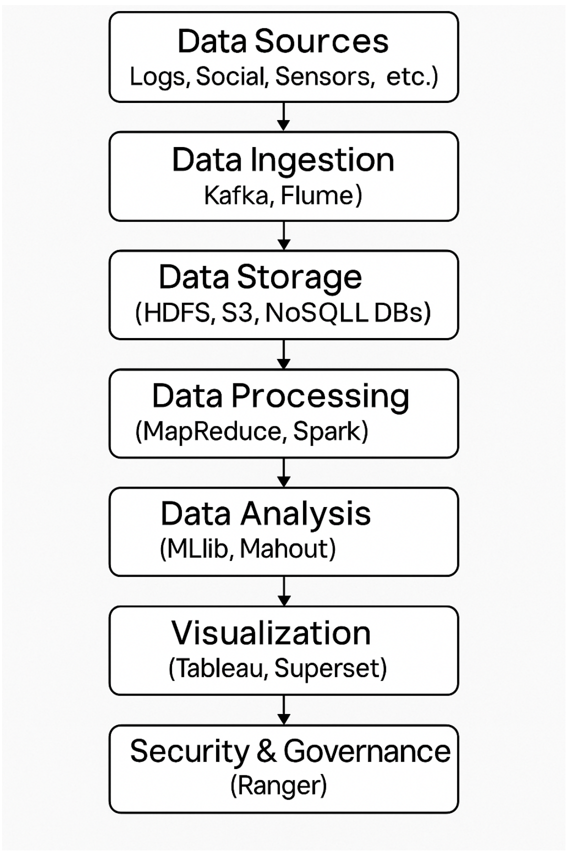
Data Management – Hive and Pig help organize and prepare data for querying.

Data Analysis & ML – Tools like MLlib and TensorFlow extract insights and build models.

Orchestration & Workflow – Airflow and Oozie automate and manage data pipelines.

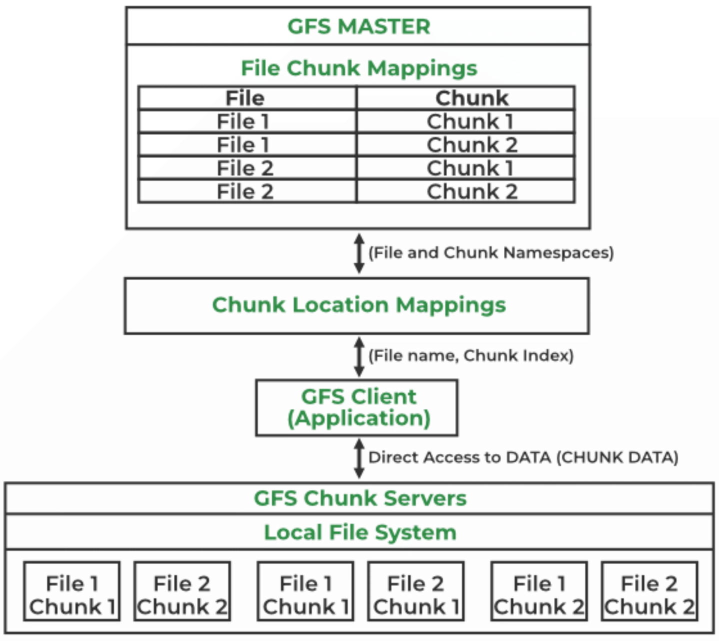
Visualization – Tableau and Power BI turn data into interactive graphs and reports.

Security & Governance – Ranger and Kerberos ensure data is secure and properly managed.



2. Google File System:-

- Google File System (GFS), also called GoogleFS, is a special system created by Google to store and manage very large amounts of data across many computers.
- It is designed to be fast, reliable, and able to keep working even if some parts fail.
- GFS uses cheap hardware and splits large files into big chunks (64 MB each), storing them on several computers (called chunk servers). Each chunk is saved in three copies to prevent data loss.
- One main computer (called the master) keeps track of where all the pieces of data are stored and manages things like file names and permissions.
- The master checks in with the chunk servers regularly to make sure everything is working.
- GFS supports thousands of computers and hundreds of users at the same time, storing huge amounts of data (like 300 TB) and allowing easy and quick access to it.



Components of GFS

A group of computers makes up GFS. A cluster is just a group of connected computers. There could be hundreds or even thousands of computers in each cluster. There are three basic entities included in any GFS cluster as follows:

1. GFS Clients: These are programs or apps that ask for files – they can read, change, or add new files in the system.

2. GFS Master Server: This is the main controller. It keeps track of what's happening in the system and stores important info (called metadata) about where each piece of a file is and which file it belongs to.

3. GFS Chunk Servers: These are the machines that store the actual pieces of files, each 64 MB in size. They don't send data to the master but send it straight to the client. Each file piece is saved in three copies on different chunk servers to make sure nothing is lost if one server fails.

Features of GFS

- Namespace management and locking.
- Fault tolerance.
- High availability.
- Critical data replication.
- Automatic and efficient data recovery.
- High aggregate throughput.

Advantages of GFS

1. High accessibility Data is still accessible even if a few nodes fail.
2. Excessive throughput. many nodes operating concurrently.
3. Dependable storing.

Disadvantages of GFS

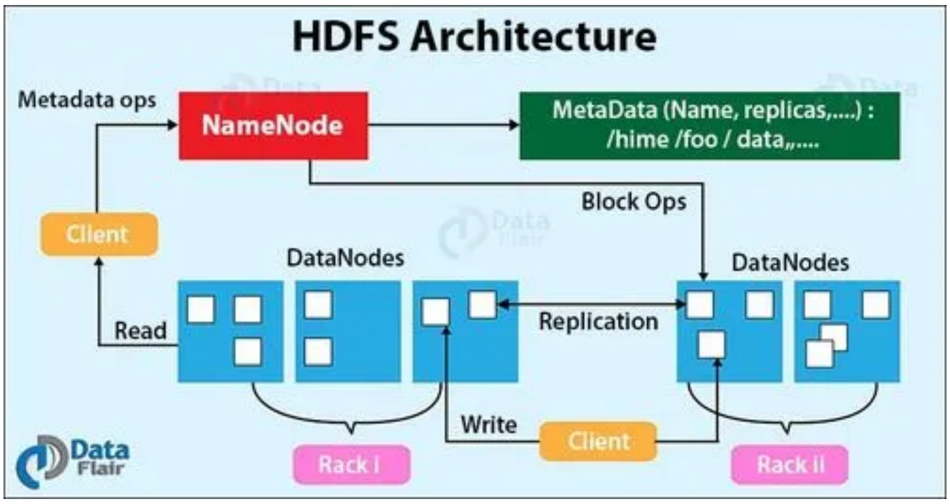
1. Not the best fit for small files.
2. Master may act as a bottleneck.
3. unable to type at random.
4. Suitable for procedures or data that are written once and only read (appended) later.

3.Hadoop Distributed File System(HDFS):-

- With growing data velocity the data size easily outgrows the storage limit of a machine. A solution would be to store the data across a network of machines. Such filesystems are called **distributed filesystems**.

- Hadoop is provides one of the most reliable filesystems. **HDFS (Hadoop Distributed File System)** is a unique design that provides storage for extremely large files with streaming data access pattern, and it runs on commodity hardware.

  - **Extremely large files**: Here, we are talking about the data in a range of petabytes (1000 TB).
  - **Streaming Data Access Pattern**: HDFS is designed on principle of write-once and read-many-times. Once data is written large portions of dataset can be processed any number times.
  - **Commodity hardware:** Hardware that is inexpensive and easily available in the market. This is one of the features that especially distinguishes HDFS from other file systems.



Nodes: Master-slave nodes typically form the **HDFS cluster.**

1. **NameNode(MasterNode):**
   - Manages all the slave nodes and assigns work to them.
   - It executes filesystem namespace operations like opening, closing, and renaming files and directories.
   - It should be deployed on reliable hardware that has a high configuration. not on commodity hardware.
2. **DataNode(SlaveNode):**
   - Actual worker nodes do the actual work like reading, writing, processing, etc.
   - They also perform creation, deletion, and replication upon instruction from the master.
   - They can be deployed on commodity hardware.

HDFS daemons: Daemons are the processes running in the background.

- **Namenodes:**
  - Run on the master node.
  - Store metadata (data about data) like file path, the number of blocks, block Ids. etc.
  - Requires a high amount of RAM.
  - Store meta-data in RAM for fast retrieval i.e to reduce seek time. Though a persistent copy of it is kept on disk.
- **DataNodes:**
  - Run on slave nodes.
  - Require high memory as data is actually stored here.

Features:

- Distributed data storage.
- Blocks reduce seek time.
- The data is highly available as the same block is present at multiple datanodes.
- High fault tolerance.

Limitations:

- Low latency data access:
- Small file problem:

4. Hadoop Shell commands:-

1. hdfs dfs -ls /path

Description: Lists all the files and directories in the given HDFS path.
Use Case: To check the contents of a folder in HDFS.
Example: hdfs dfs -ls /user/hadoop

This lists all files and subdirectories under /user/hadoop.

2. hdfs dfs -mkdir /path

Description: Creates a new directory in the Hadoop Distributed File System.
Use Case: To organize files by creating folders in HDFS.
Example: hdfs dfs -mkdir /user/hadoop/newfolder

3. hdfs dfs -put localfile /path

Description: Copies a file from the local file system to the HDFS.
Use Case: To upload data files to HDFS for processing.
Example: hdfs dfs -put data.csv /user/hadoop/input

4. hdfs dfs -get /path localdir

Description: Downloads a file from HDFS to your local system.
Use Case: To retrieve processed data from HDFS.
Example: hdfs dfs -get /user/hadoop/output/result.txt /home/user/

5. hdfs dfs -rm /path

Description: Deletes a file from the HDFS.
Use Case: To remove unwanted or old files from the HDFS.
Example: hdfs dfs -rm /user/hadoop/oldfile.txt

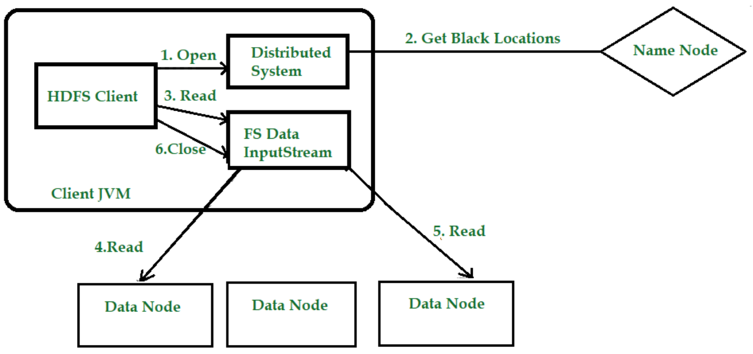
5. Anatomy of File Read and Write in HDFS:-

- Big data is nothing but a collection of data sets that are large, complex, and which are difficult to store and process using available data management tools or traditional data processing applications.
- Hadoop is a framework (open source) for writing, running, storing, and processing large datasets in a parallel and distributed manner.

Hadoop has two components:

- HDFS (Hadoop Distributed File System)
- YARN (Yet Another Resource Negotiator)

A. Anatomy of File Read in HDFS



Step 1: Open File
The client starts reading a file by calling the open() function on the Hadoop file system. This prepares the system to access the file data.

Step 2: Contact NameNode
The file system asks the NameNode where the first parts (blocks) of the file are stored. The NameNode replies with the addresses of DataNodes holding those blocks and gives the client a stream to read.

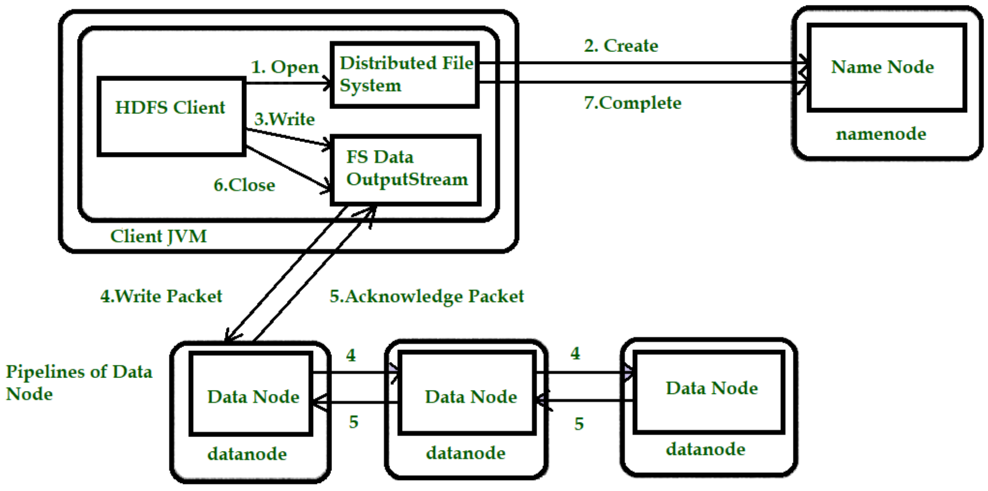
Step 3: Connect to DataNode
The client calls read() on the stream, which connects to the closest DataNode that has the first block. This DataNode starts sending the data to the client.

Step 4: Stream Data
Data flows continuously from the DataNode to the client as it reads the stream. The client keeps reading until the current block is fully read.

Step 5: Switch Blocks
When the client finishes reading one block, the connection to that DataNode closes. The system then finds the next DataNode with the next block, so the client can keep reading smoothly.

]Step 6: Close File
After finishing reading the entire file, the client closes the input stream. This ends the reading process.

B. Anatomy of File Write in HDFS



Step 1: File Creation
The client starts the process of writing a new file by calling the create() method through the DistributedFileSystem (DFS). This initiates communication with the HDFS system.

Step 2: Contact with NameNode
DFS sends a request to the NameNode to create the file. The NameNode checks for permissions and ensures the file doesn't already exist. If valid, it creates a record and returns a stream (FSDataOutputStream) to the client for writing.

Step 3: Splitting and Packet Queueing
As the client writes data, it is split into smaller packets and placed in a queue. The DataStreamer handles these packets, contacts the NameNode to allocate blocks, and sets up a pipeline of three DataNodes for replication.

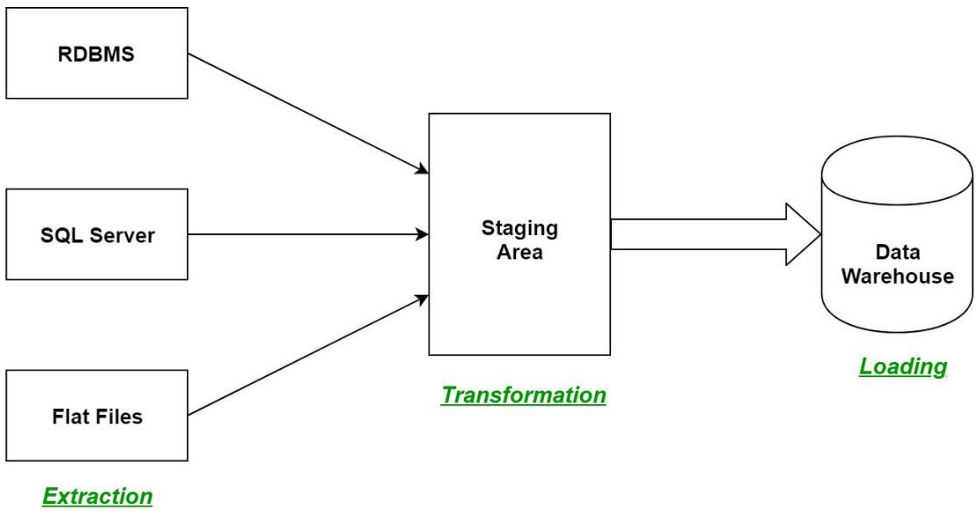
Step 4: Packet Transmission
The DataStreamer sends the packets to the first DataNode in the pipeline. Each DataNode stores the packet and forwards it to the next one, ensuring that all three store a copy of the same data block.

Step 5: Acknowledgment
An acknowledgment queue keeps track of packets successfully stored. Each DataNode sends an acknowledgment back to the client, confirming safe storage.

Step 6: Completion
After all packets are acknowledged, the client notifies the NameNode that the file writing is complete. The file is then finalized and closed in the HDFS system.

6. ETL (Extract, Transform, Load) Process:-

- The ETL process, which stands for Extract, Transform, and Load, is a critical methodology used to prepare data for storage, analysis, and reporting in a data warehouse.
- It involves three distinct stages that help to streamline raw data from multiple sources into a clean, structured, and usable form.
- It provides a structured foundation for data analytics, improving the quality, security, and accessibility of enterprise data.



ETL

1. Extraction

The Extract phase is the first step in the ETL process, where raw data is collected from various data sources. These sources can be diverse, ranging from structured sources like databases (SQL, NoSQL), to semi-structured data like JSON, XML, or unstructured data such as emails or flat files. The main goal of extraction is to gather data without altering its format, enabling it to be further processed in the next stage.

Types of data sources can include:

- **Structured:** SQL databases, ERPs, CRMs
- **Semi-structured:** JSON, XML
- **Unstructured:** Emails, web pages, flat files

2. Transformation

The Transform phase is where the magic happens. Data extracted in the previous phase is often raw and inconsistent. During transformation, the data is cleaned, aggregated, and formatted according to business rules. This is a crucial step because it ensures that the data meets the quality standards required for accurate analysis.

Common transformations include:

- **Data Filtering:** Removing irrelevant or incorrect data.
- **Data Sorting:** Organizing data into a required order for easier analysis.
- **Data Aggregating:** Summarizing data to provide meaningful insights (e.g., averaging sales data).

The transformation stage can also involve more complex operations such as currency conversions, text normalization, or applying domain-specific rules to ensure the data aligns with organizational needs.

3. Loading

Once data has been cleaned and transformed, it is ready for the final step: Loading. This phase involves transferring the transformed data into a data warehouse, data lake, or another target system for storage. Depending on the use case, there are two types of loading methods:

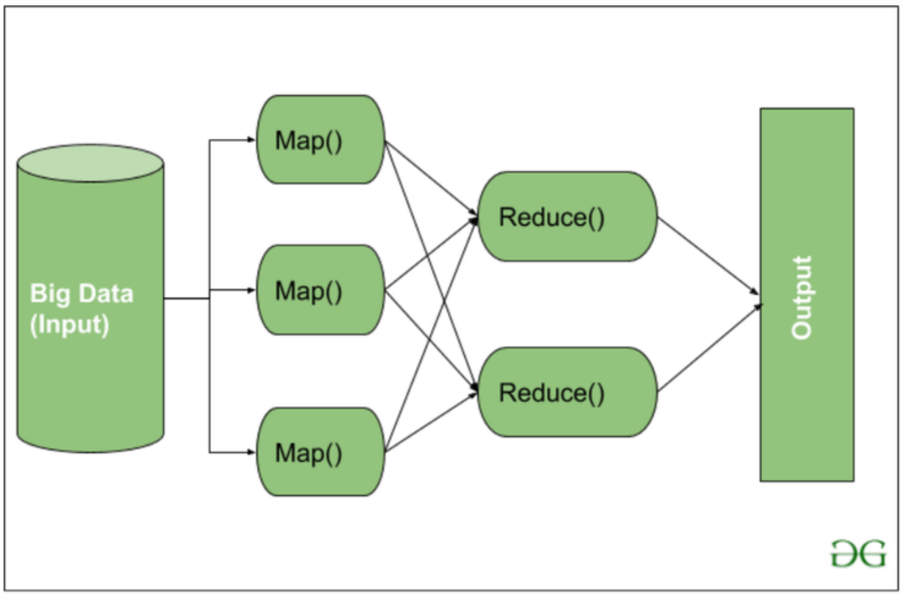
- **Full Load:** All data is loaded into the target system, often used during the initial population of the warehouse.
- **Incremental Load:** Only new or updated data is loaded, making this method more efficient for ongoing data updates.

Challenges in ETL Process:

- Data Quality Issues:
- Performance Bottlenecks:
- Scalability Issues:

7. Map Reduce in Hadoop:-

- One of the three components of Hadoop is Map Reduce. The first component of Hadoop that is, Hadoop Distributed File System (HDFS) is responsible for storing the file. The second component that is, Map Reduce is responsible for processing the file.
- MapReduce has mainly 2 tasks which are divided phase-wise.In first phase, **Map** is utilised and in next phase **Reduce** is utilised.
- **MapReduce** is required because it provides a simple and efficient way to process this big data in a **distributed** manner.
- It breaks the job into smaller tasks that run in parallel on different nodes, which speeds up processing and makes handling huge data practical. It also handles failures and ensures that the process is reliable and scalable.



Example:
Input:

```
Hello I am GeeksforGeeks
Hello I am an Intern
```

Output:

```
GeeksforGeeks  1
Hello    2
I        2
Intern   1
am       2
an       1
```

Step 1: File Splitting in HDFS

The large file sample.txt is split into four parts: first.txt, second.txt, third.txt, and fourth.txt. Each part contains two lines and is stored on different DataNodes in the Hadoop cluster.

Step 2: User Submits Job

The user submits a MapReduce job using a command like:
hadoop jar wordcount.jar DriverCode sample.txt result.output
Here, wordcount.jar is the program, sample.txt is the input file, and result.output is where the output will be saved.

Step 3: Job Tracker Coordinates the Job

The Job Tracker receives the request and contacts the NameNode to get metadata about the file splits. It knows where the file parts are stored and assigns tasks to Task Trackers (workers) near the data locations.

Step 4: Map Phase

Each Task Tracker runs a Mapper on one input split. The Mapper converts each line of text into key-value pairs like (word, 1). So, words from the file get converted into pairs like (Hello, 1), (I, 1), (am, 1), etc.

Step 5: Shuffle and Sort

The intermediate data from all Mappers is shuffled and sorted by keys. All counts for the same word are grouped together, for example, (Are, [1, 1, 1]) if "Are" appeared three times.

Step 6: Reduce Phase

Reducers take the shuffled and sorted data and sum the counts for each word. For example, the reducer adds all the 1's for "Are" and outputs (Are, 3).

Step 7: Output Written

The final word count results are written to the output directory result.output on HDFS.

8. Differentiate between SQL and NoSQL database:-

Aspect	SQL (Relational)	NoSQL (Non-relational)
Data Structure	Tables with rows and columns	Document-based, key-value, column-family, or graph-based
Schema	Fixed schema (predefined structure)	Flexible schema (dynamic and adaptable)
Scalability	Vertically scalable (upgrading hardware)	Horizontally scalable (adding more servers)
Data Integrity	ACID-compliant (strong consistency)	BASE-compliant (more available, less consistent)
Query Language	SQL (Structured Query Language)	Varies (e.g., MongoDB uses its own query language)
Performance	Efficient for complex queries and transactions	Better for large-scale data and fast read/write operations
Examples	MySQL, PostgreSQL, Oracle, MS SQL Server	MongoDB, Cassandra, CouchDB, Neo4j

NoSQL Database:

NoSQL databases are a category of **non-relational databases** designed to handle **large-scale, unstructured, and semi-structured data** efficiently.

1. Handling Huge Volumes of Data

- NoSQL databases are designed to **store and process massive amounts of data** (structured, semi-structured, and unstructured), which is essential in big data environments.

2. Flexible Schema

- They offer **schema-less or flexible schema designs**, making it easy to handle dynamic and evolving data formats — perfect for unstructured big data like logs, social media, or IoT data.

3. High Scalability

- NoSQL systems are **horizontally scalable**, meaning they can handle more data by adding more servers to the system — a necessity for big data growth.

4. High Performance

- NoSQL databases are optimized for **fast read/write operations**, making them ideal for real-time applications like recommendation engines, dashboards, and analytics.

5. Distributed Architecture

- Most NoSQL databases are **distributed by default**, which means data is automatically partitioned and stored across multiple servers, increasing fault tolerance and data availability.

6. Cost-Effective

- By using **commodity hardware** and open-source technologies, NoSQL databases provide a cost-efficient solution to manage and process large-scale data.

Types of NoSQL Database:

Feature	**Document-Based**	**Key-Value Store**	**Column-Oriented**	**Graph-Based**
Data Model	JSON-like documents	Key-Value pairs	Columns instead of rows	Nodes & Relationships
Best Use Case	Semi-structured data	Fast lookups & caching	Analytics & big data	Relationship-heavy data
Query Performance	Moderate	Fast	High for analytics	Optimized for relationships
Schema	Flexible	Dynamic	Semi-structured	Schema-less
Scalability	Horizontal	High horizontal	Highly scalable	Scales with relationships
Examples	MongoDB, CouchDB	Redis, DynamoDB	Cassandra, HBase	Neo4j, Amazon Neptune