Module 3: NoSQL

- Introduction to NoSQL
- NoSQL Business Drivers
- NoSQL Data Architecture Patterns: Key-value stores, Graph stores, Column family Bigtable stores, Document stores,
- Variations of NoSQL architectural patterns,
- NoSQL Case Studies
- NoSQL solution for big data, Understanding the types of big data problems;
- Analyzing big data with a shared-nothing architecture;
- Choosing distribution models: master-slave versus peer-to-peer;
- NoSQL systems to handle big data problems.

Database Components

A database: big container wherein all the information about a website, or an application is stored in a structured format like tables, hierarchy.

The major components of the Database are:

Hardware :This consists of a set of physical electronic devices such as I/O devices, storage devices and many more. It also provides an interface between computers and real-world systems.

Software:The set of programs that are used to control and manage the overall Database. It also includes the DBMS software itself.The OS, the network software being used to share the data among the users, the application programs used to access data in the DBMS.

Data: DBMS collects, stores, processes, and accesses data.The Database holds both the actual or operational data and the metadata.

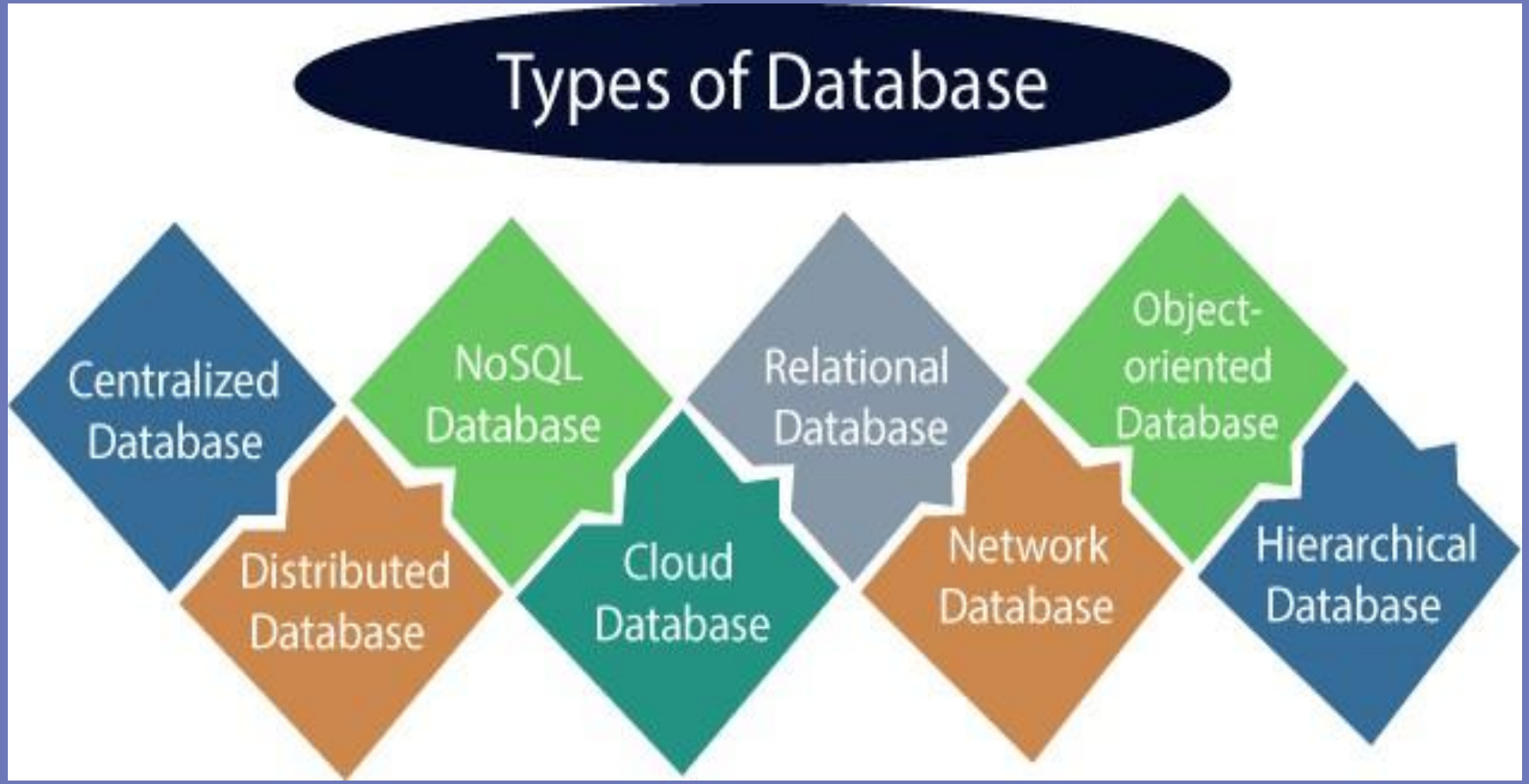
A DBMS consists of a group of commands to manipulate the database and acts as an interface between the end- users and the database

DBMS allows users to do the following:
Define Data – Allows the users to create, modify and delete the definitions which define the organization of the database.

Update Data – Provides access to the users to insert, modify and delete data from the database.

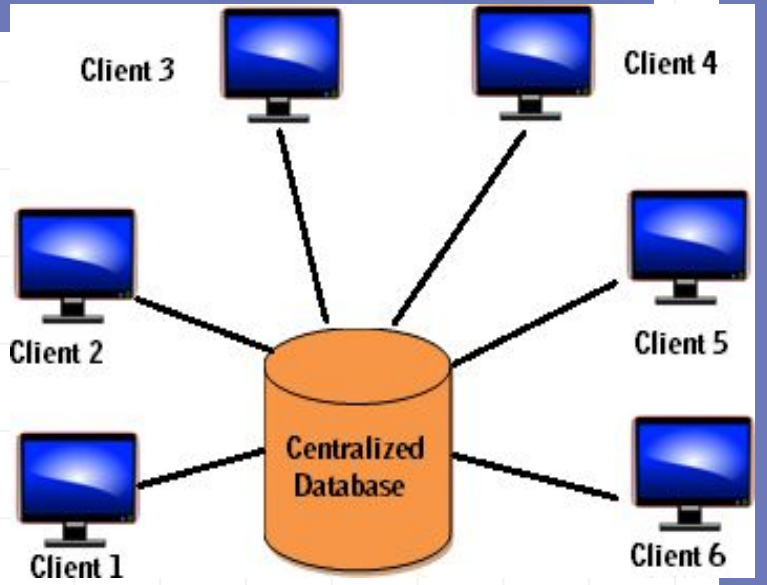
Retrieve Data – Allows the users to retrieve data from the database based on the requirement.

Administration of users – Registers the users and monitors their action, enforces data security, maintains data integrity, monitors performance and deals with concurrency control.


Types of Database
Centralized Database
Distributed Database
NoSQL Database
Cloud Database
Relational Database
Network Database
Object-oriented Database
Hierarchical Database

Centralized Database

- It is the type of database that stores data at a centralized database system.
- It comforts the users to access the stored data from different locations through several applications.
- These applications contain the authentication process to let users access data securely.
- An example: Central Library that carries a central database of each library in a college/university.

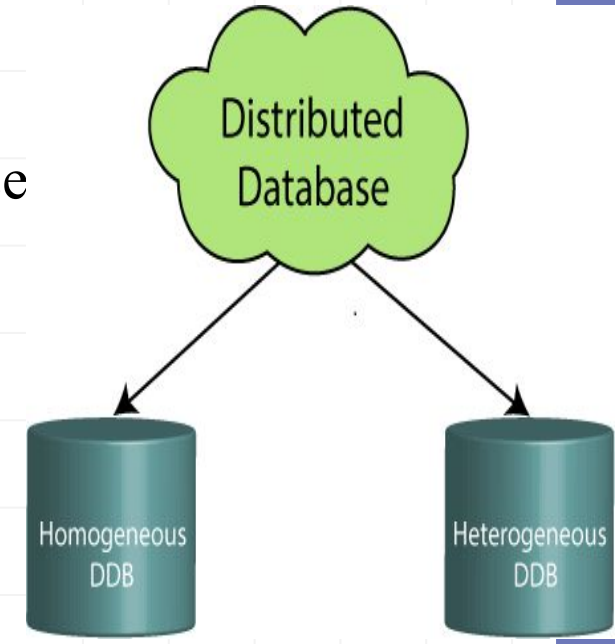


- Not easy to update,
- Response time for fetching
- Data loss in case of server failure!

Distributed Database

- Data is distributed among different database systems of an organization.
- These database systems are connected via communication links. Such links help the end-use to access the data easily.

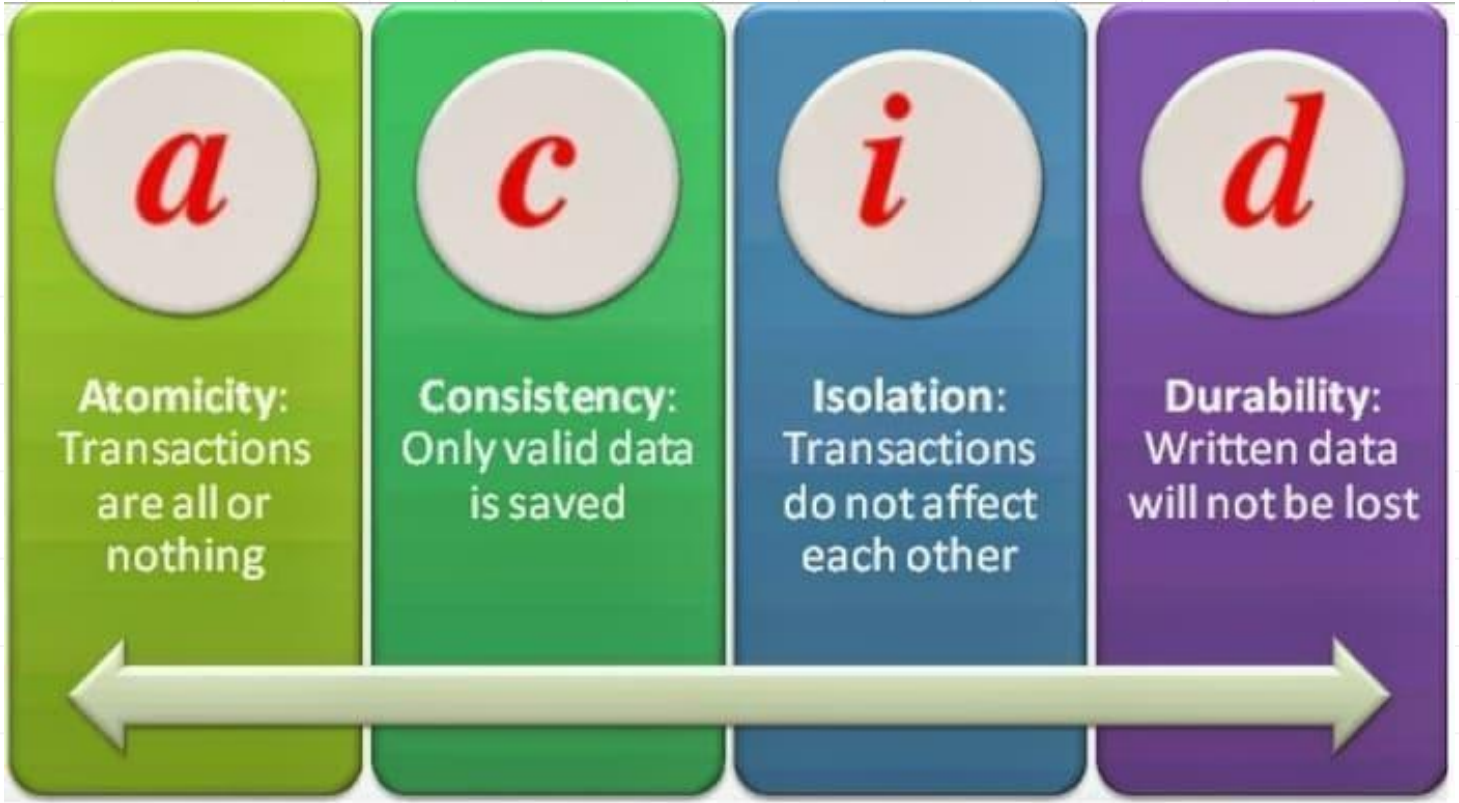
- Modular development is possible in a distributed database, i.e., the system can be expanded by including new computers and connecting them to the distributed system.
- One server failure will not affect the entire data set.
  - **Examples** Apache Cassandra, HBase, Ignite, etc.



Relational Database

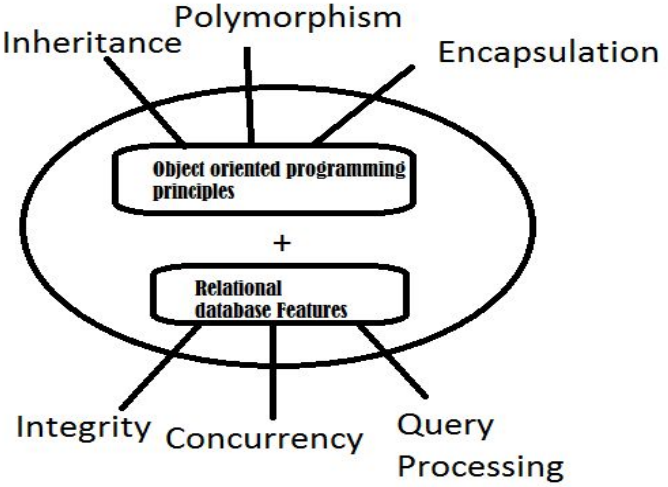
- This database is based on the relational data model, which stores data in the form of rows (tuple) and columns (attributes), and together forms a table(relation).
- A relational database uses SQL for storing, manipulating, as well as maintaining the data. E.F. Codd invented the database in 1970.
- Each table in the database carries a key that makes the data unique from others.
- **Examples** of Relational databases are MySQL, Microsoft SQL Server, Oracle, IBM DB2 etc.

Properties of Relational Database



Object-oriented Databases

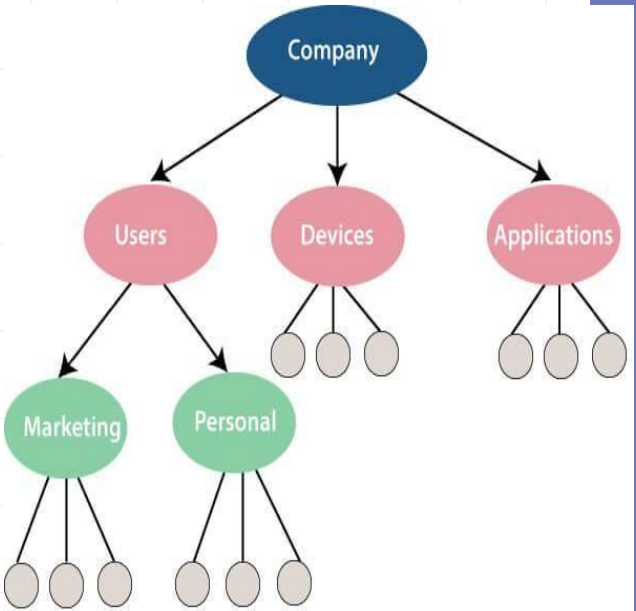
- The type of database that uses the object-based data model approach for storing data in the database system.
- The data is represented and stored as objects which are similar to the objects used in the object-oriented programming language.



  - Objects are the real-world entity, and types are the collection of objects.
  - It is an alternative implementation to that of the relational model.
  - **properties:** Objects, Classes, Inheritance, Polymorphism, Encapsulation
    - Eg. GemStone, Gbase, Vbase.

Hierarchical Databases

- It is the type of database that stores data in the form of parent- children relationship nodes.
- Here, it organizes data in a tree- like structure.
- Data gets stored in the form of records that are connected via links.
- Each child record in the tree will contain only one parent. On the other hand, each parent record can have multiple child records.
- IBM's Information Management System (IMS), XML



Network Databases

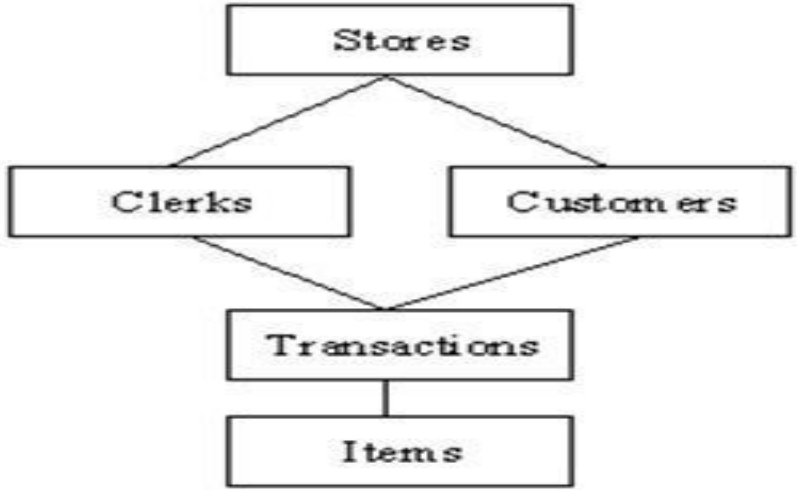
- It is the database that typically follows the network data model.
- Here, the representation of data is in the form of nodes connected via links between them.
- Unlike the hierarchical database, it allows each record to have multiple children and parent nodes to form a generalized graph structure.

Integrated Data Store (IDS)

Raima Database Manager

Univac DMS-1100

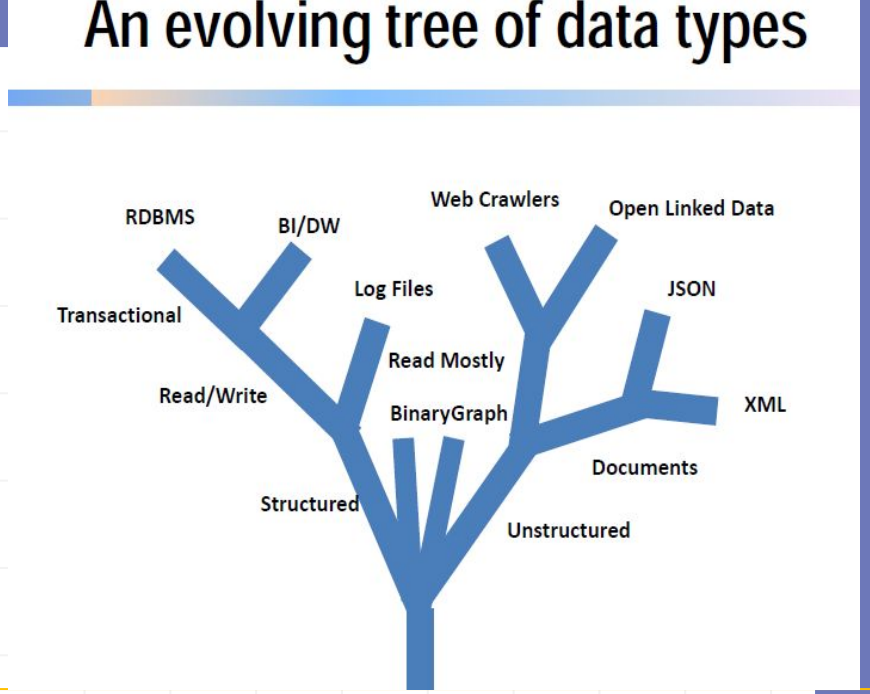
IDMS (Integrated Database Mngt System)



- **Personal Database:** Collecting and storing data on the user's system defines a Personal Database.This database is basically designed for a single user.
- **Operational Database:** The type of database which creates and updates the database in real-time. It is basically designed for executing and handling the daily data operations in several businesses. For example, An organization uses operational databases for managing per day transactions.
- **Enterprise Database:** Large organizations or enterprises use this database for managing a massive amount of data. It helps organizations to increase and improve their efficiency. Such a database allows simultaneous access to users.

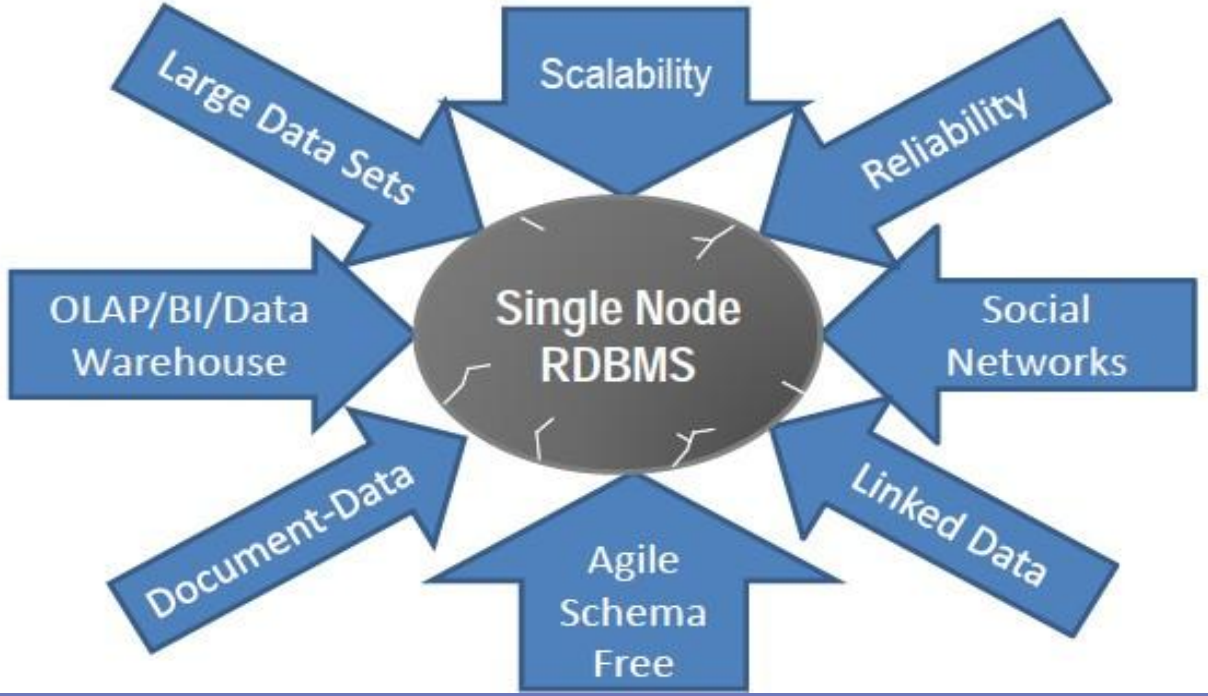
SQL

- SQL stands for Structured Query Language.
- It is used for storing and managing data in relational database management system (RDBMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

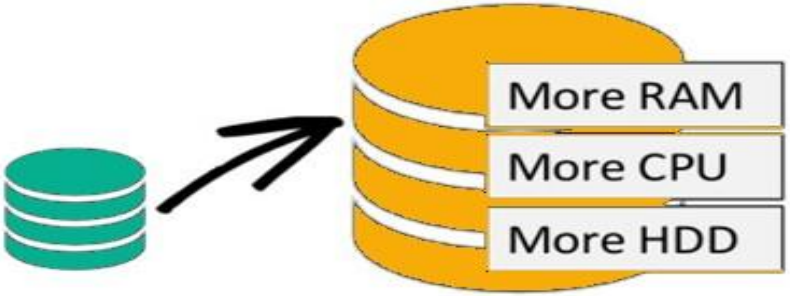


Humongous data storage is not possible with RDBMS, which stores data in fixed schema and big data is having variety of data. The system response time becomes slow when you use RDBMS for massive volumes of data.

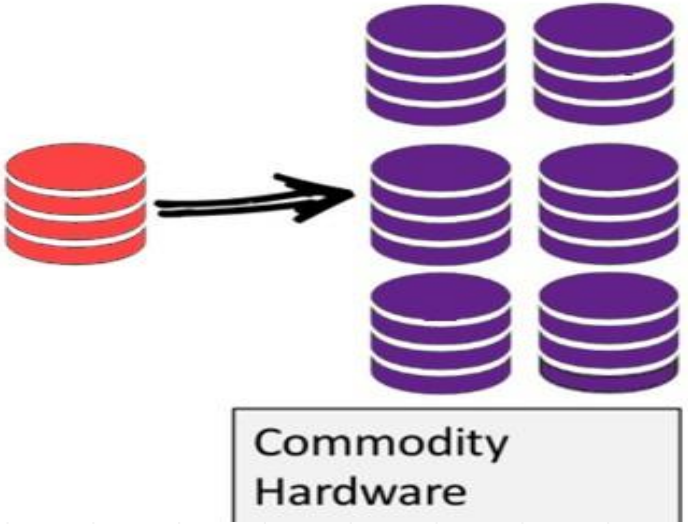
Pressures on Single Node RDBMS Architectures



Scale-Up (*vertical* scaling):



Scale-Out (*horizontal* scaling):



"Scale up" our systems by upgrading our existing hardware.This process is expensive.

Distribute database load on multiple hosts whenever the load increases.This method is known as "Scaling out."

NoSQL Name

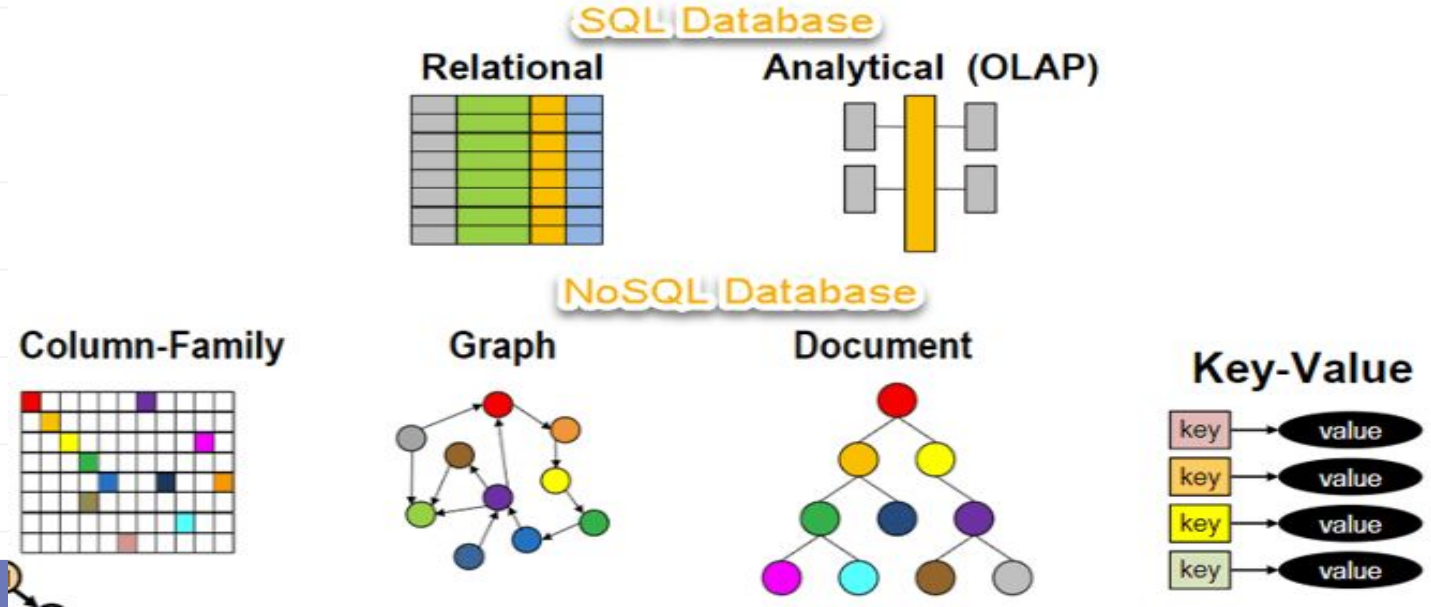
- "SQL" = Traditional Relational DBMS
- Recognition over past decade or so
- Not every data management/analysis problem is best solved using a traditional relational DBMS
- "NoSQL" = "No SQL" =Not using traditional relational DBMS
- "No SQL" ≠ Don't use SQL language
- "NoSQL" = ? "Not Only SQL" => NOSQL
- A common misconception is that NoSQL databases or non-relational databases don't store relationship data well.
    - NoSQL databases can store relationship data—they just store it differently than relational databases do.

What is NoSQL?

- **NoSQL** Database is a non-relational Data Management System, that does not require a fixed schema.
- It avoids joins, and is easy to scale.
- The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs.
- NoSQL is used for Big data and real-time web apps.
- For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.
- Carlo Strozzi introduced the NoSQL concept in 1998.

Traditional RDBMS uses SQL syntax to store and retrieve data for further insights.

A NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data.



Features of NoSQL

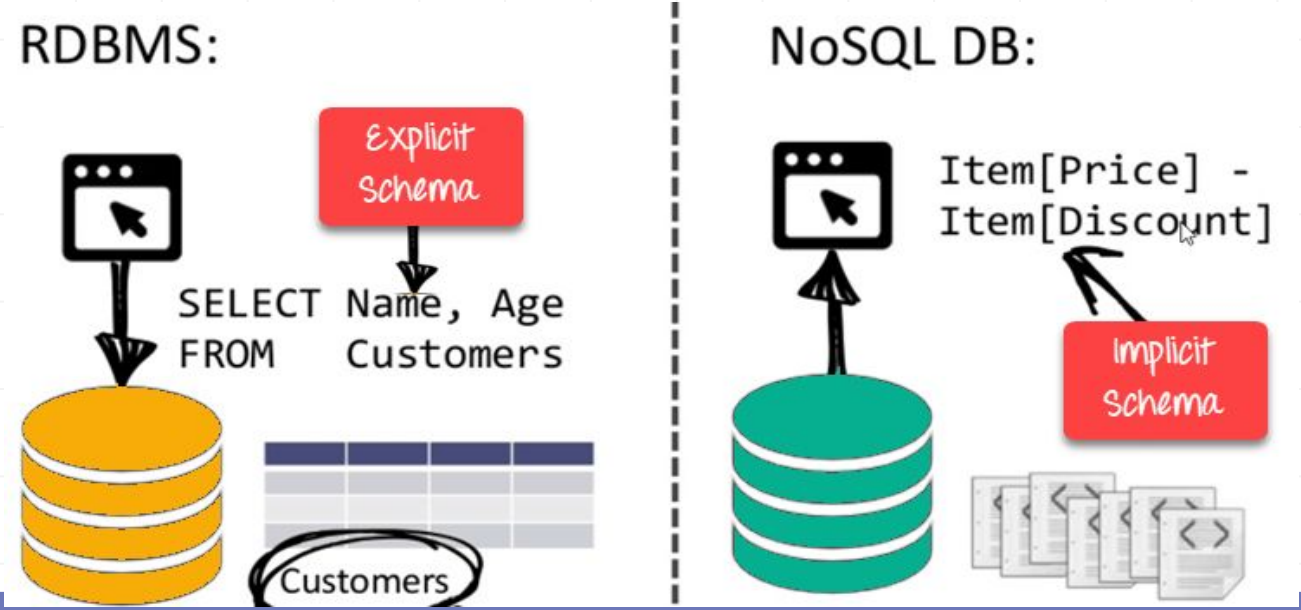
Non-relational

- NoSQL databases never follow the relational model
- Never provide tables with flat fixed-column records
- Work with self-contained aggregates or BLOBs
- Doesn't require object-relational mapping and data normalization
- No complex features like query languages, query planners,referential integrity, joins, ACID

Features of NoSQL

Schema-free

- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data
- Offers heterogeneous structures of data in the same domain



Features of NoSQL

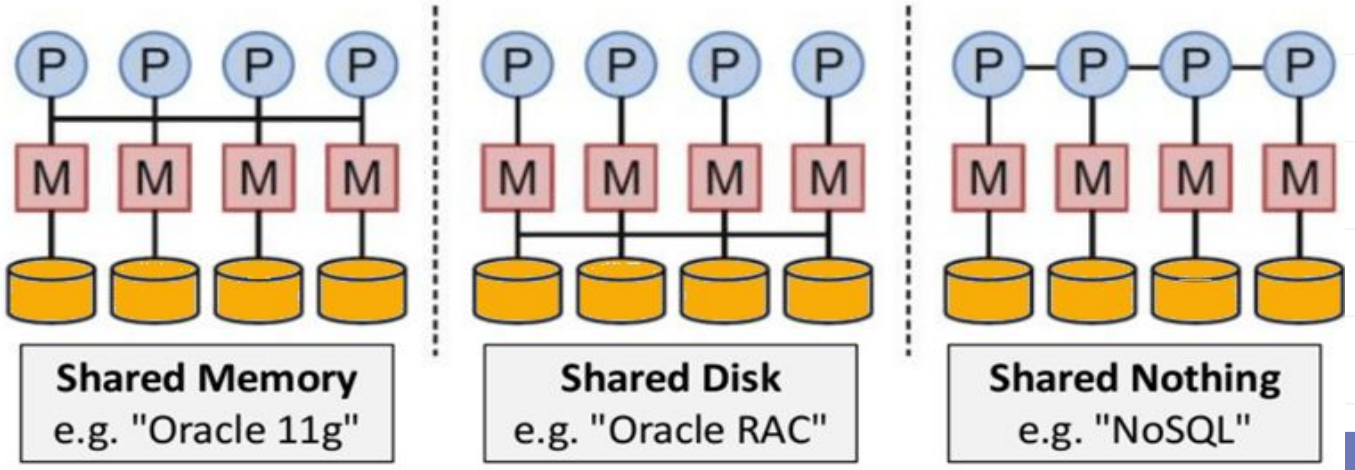
Simple API

- Offers easy to use interfaces for storage and querying data provided
- APIs allow low-level data manipulation & selection methods
- Text-based protocols mostly used with HTTP REST(**RE**presentational **S**tate **T**ransfer) with JSON
- REST: An architectural style for distributed hypermedia systems
- Web-enabled databases running as internet-facing services

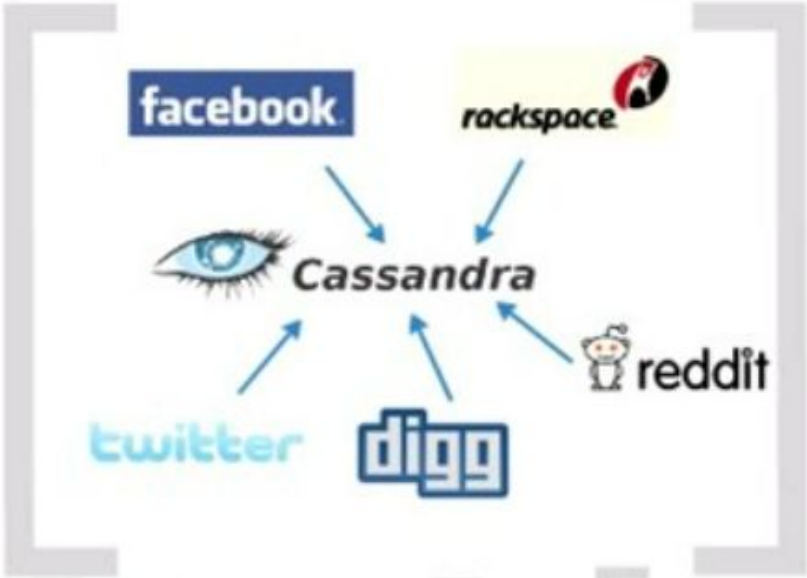
Features of NoSQL

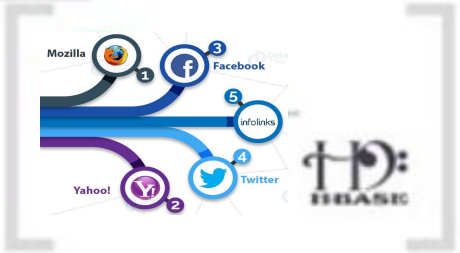
Distributed

- Multiple NoSQL databases can be executed in a distributed fashion
- Offers auto-scaling and fail-over capabilities
- Often ACID concept can be sacrificed for scalability and throughput
- Shared Nothing Architecture. This enables less coordination and higher distribution.



Who is using them?





Advantages of NoSQL

- Scale(horizontal): SQL databases are vertically scalable⇒you can increase the load on a single server by increasing things like RAM, CPU or SSD. NoSQL databases are horizontally scalable⇒ you handle more traffic by sharding, or adding more servers
- Simple data model (fewer joins)
- Offers a flexible schema design which can easily be altered without downtime or service disruption
- Handles big data which manages data velocity, variety, volume, and complexity
- Flexible as it can handle semi-structured, unstructured and structured data.
- Cheaper than relational database
- Creates a caching layer: Eliminates the need for a specific caching layer to store data
- Uses large binary objects for storing large data
- Bulk upload
- Excels at distributed database and multi-data center operations
- Real-time analysis:It can serve as the primary data source for online applications.
- No Single Point of Failure,Easy Replication
- Lower administration: NoSQL databases don’t need a dedicated high-performance server

Disadvantages of NoSQL

- No standardization rules
- Limited query capabilities
- RDBMS databases and tools are comparatively mature
- It does not offer any traditional database capabilities, like consistency when multiple transactions are performed simultaneously.
- When the volume of data increases it is difficult to maintain unique values as keys become difficult
- Doesn’t work as well with relational data
- The learning curve is stiff for new developers
- Open source options so not so popular for enterprises.

Q. Differentiate between RDBMS and NoSQL database. Dec 18 10 M

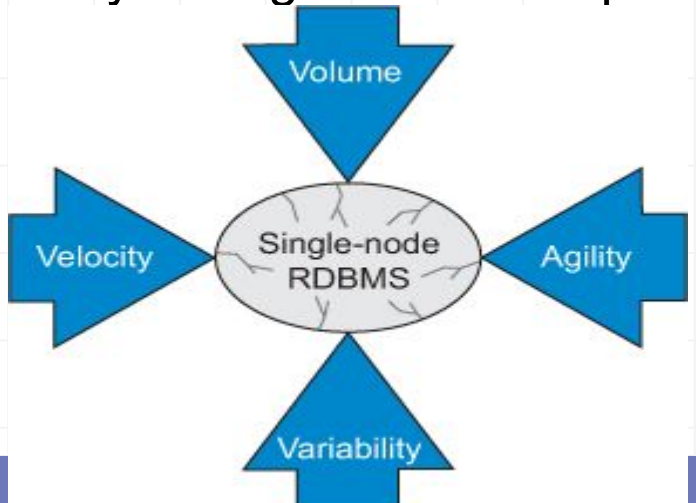
Sr	RDBMS	NoSQL
1	Have Fixed or Static or Predefined Schema	Have Dynamic Schema
2	Not Suited for Hierarchical data storage	Best Suited for Hierarchical data storage
3	Vertically Scalable	Horizontally Scalable
4	Follow ACID Properties	Follows CAP (Consistency, Availability, Partition Tolerance)
5	Supports Transactions (also complex transactions with joins)	Don’t Supports Transactions (support only simple transactions)
6	Manages only Structures Data	manage structured, unstructured and semi-structured data

Sr	RDBMS	NoSQL
7	have a single point of failure with failover.	have no single point of failure.
8	supports a powerful query language.	supports a very simple query language.
9	gives read scalability only	gives both read and write scalability.
10	Transactions written in one location	Transactions written in many location
11	used to handle data coming in low velocity.	used to handle data coming in high velocity.
12	Eg. MySQL, Oracle, Sqlite, PostgreSQL, MS-SQL	Eg. MongoDB, BigTable, Redis, RavenDB, Cassandra, HBase, Neo4j, CouchDB

NoSQL Business Drivers

- NoSQL: Paradigm Shift in terms of patterns, architectures and methods
- Earlier, Many organizations supporting single-CPU relational systems to fulfill the needs of their organizations
- Later, Businesses have found value in rapidly capturing and analyzing large amounts of variable data,
- making immediate changes in their businesses based on the information they receive.
- There is burden on single-CPU relational systems

- the business drivers volume, velocity, variability, and agility apply pressure to the single CPU system, resulting in the cracks.
- Volume and velocity refer to the ability to handle large datasets that arrive quickly
- Variability refers to how diverse data types don’t fit into structured tables,
- agility refers to how quickly an organization responds to business change.



Volume

- the key factor pushing organizations to look at alternatives to their current RDBMSs is a need to query big data using clusters of commodity processors.
- Until around 2005, performance concerns were resolved by purchasing faster processors.
- The ability to increase processing speed was no longer an option
- As chip density increased, heat could no longer dissipate fast enough without chip overheating.
- This phenomenon, known as the power wall, forced systems designers to shift their focus from increasing speed on a single chip to using more processors working together.

Volume

- The need to scale out (also known as horizontal scaling), rather than scale up (faster processors),
- moved organizations from serial to parallel processing
- Data problems are split into separate paths and sent to separate processors to divide and conquer the work.

Velocity

- Though big data problems are a consideration for many organizations moving away from RDBMSs
- the ability of a single processor system to rapidly read and write data is also key.
- Many single-processor RDBMSs are unable to keep up with the demands of real-time inserts and online queries to the database made by public-facing websites.
- RDBMSs frequently index many columns of every new row, a process which decreases system performance.
- When single-processor RDBMSs are used as a backend to a web store front, the random bursts in web traffic slow down response for everyone,
- tuning these systems can be costly when both high read and write throughput is desired.

Variability

- Companies that want to capture and report on exceptional data conflicts when attempting to use rigid database schema structures imposed by RDBMSs.
- For example, if a business unit wants to capture a few custom fields for a particular customer, all customer rows within the database need to store this information even though it doesn't apply.
- Adding new columns to an RDBMS requires the system be shut down and ALTER TABLE commands to be run.
- When a database is large, this process can impact system availability, costing time and money.

Agility

Agility is ability to accept change easily and quickly.

- Among the variety of agility dimensions such as
  - **model agility** (ease and speed of changing data modules),
  - **operational agility** (ease and speed of changing operational aspects)
  - **programming agility** (ease and speed of application development)
    - one of the most important is the ability to quickly and seamlessly scale an application to accommodate large amounts of data, users and connections.
- The most complex part of building applications using RDBMS is the process of putting data into and getting data out of the database
- If your data has nested and repeated subgroups of data structures, you need to include an object-relational mapping layer.

- The responsibility of object-relational mapping layer is to generate the correct combination of SQL statement to move object data to and from the RDBMS.
- This process is not simple and is associated with the largest barrier to rapid change when developing new or modifying existing applications.
- Generally object relational mapping requires experienced software developers.
- Even with experienced staff, small change requests can cause slowdowns in development and testing schedules.
- All these hurdles are best overcomed by NOSQL database.
- These databases are schematic and can be scaled down easily.
- They can accommodate application changes easily and can handle any volume of data efficiently.
- This agility has become business driver for NOSQL databases.

The CAP theorem, originally introduced as the CAP principle, can be used to explain some of the competing requirements in a distributed system with replication.

The three letters in CAP refer to three desirable properties of distributed systems with replicated data: consistency (among replicated copies), availability (of the system for read and write operations) and partition tolerance (in the face of the nodes in the system being partitioned by a network fault).

The CAP theorem states that it is not possible to guarantee all three of the desirable properties – consistency, availability, and partition tolerance at the same time in a distributed system with data replication.

The theorem states that networked shared-data systems can only strongly support two of the following three properties:

Consistency –

Consistency means that the nodes will have the same copies of a replicated data item visible for various transactions.

A guarantee that every node in a distributed cluster returns the same, most recent and a successful write.

Availability –

Availability means that each read or write request for a data item will either be processed successfully or will receive a message that the operation cannot be completed. Every non-failing node returns a response for all the read and write requests in a reasonable amount of time.

Partition Tolerance –

Partition tolerance means that the system can continue operating even if the network connecting the nodes has a fault that results in two or more partitions, where the nodes in each partition can only communicate among each other.

NoSQL Data Architecture Patterns

rchitecture Pattern is a logical way of categorizing data that will be stored on the Database.

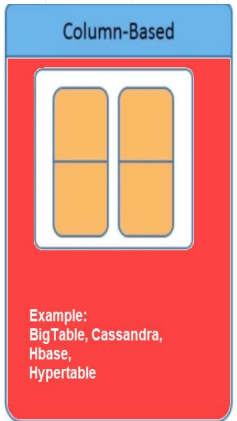
NoSQL is a type of database which helps to perform operations on big data and store it in a valid format.

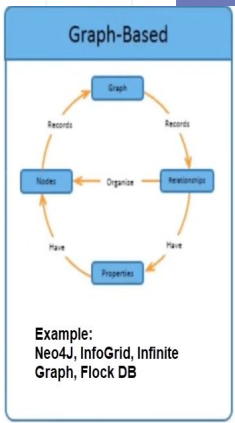
It is widely used because of its flexibility and a wide variety of services.

Architecture Patterns of NoSQL:

1. Key-Value Stores
2. Column Family (BigTable)Stores
3. Document Stores
4. Graph Stores

https://youtu.be/zG6CHYCx6ag





Key Value Pair Based

- Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load.
- Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.
- A value, which can be basically any piece of data or information, is stored with a key that identifies its location.
- It is one of the most basic NoSQL database example. This kind of NoSQL database is used as a collection, dictionaries, associative arrays, etc. Key value stores help the developer to store schema-less data. They work best for shopping cart contents.
- Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases. They are all based on Amazon's Dynamo paper.

Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

Benefits of Key-Value store

Simplicity. key value databases are quite simple to use. The straightforward commands and the absence of data types make work easier for programmers.

Speed. This simplicity makes key value databases quick to respond, provided that the rest of the environment around it is well-built and optimized.

Scalability. Unlike relational databases, which are only scalable vertically, key-value stores are also infinitely scalable horizontally.

Easy to move. The absence of a query language means that the database can be easily moved between different systems without having to change the architecture.

Reliability. Built-in redundancy comes in handy to cover for a lost storage node where duplicated data comes in place of what's been lost.

When to use a key-value database

1. When your application needs to handle lots of small continuous reads and writes, that may be volatile. Key-value databases offer fast in-memory access.
2. When storing basic information, such as customer details; storing web pages with the URL as the key and the webpage as the value; storing shopping-cart contents, product categories, e-commerce product details
3. For applications that don't require frequent updates or need to support complex queries.

Use cases for key-value databases

1. Session management on a large scale.
2. Using cache to accelerate application responses.
3. Storing personal data on specific users.
4. Product recommendations, storing personalized lists of items for individual customers.
5. Managing each player's session in massive multiplayer online games.

Key-value

- Basic API access:
  - get(key): extract the value given a key
  - put(key, value): create or update the value given its key
  - delete(key): remove the key and its associated value
  - execute(key, operation, parameters): invoke an operation to the value (given its key) which is a special data structure (e.g. List, Set, Map .... etc)

What are different architectural patterns in NoSQL? Explain Graph Data store and Column family store pattern with relevant example [May 19 10M]

Explain in detail key-value store NoSQL architectural pattern. Identify 2 applications that can use this pattern [May 18 5M]

Column-based stores

Column-oriented databases work on columns and are based on BigTable paper by Google.

Every column is treated separately.

Values of single column databases are stored contiguously.

column-store databases instead of organizing information into rows, it does so in columns.

This essentially makes them function the same way that tables work in relational databases.

As this is a NoSQL database, this data model makes them much more flexible

Row-oriented

ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

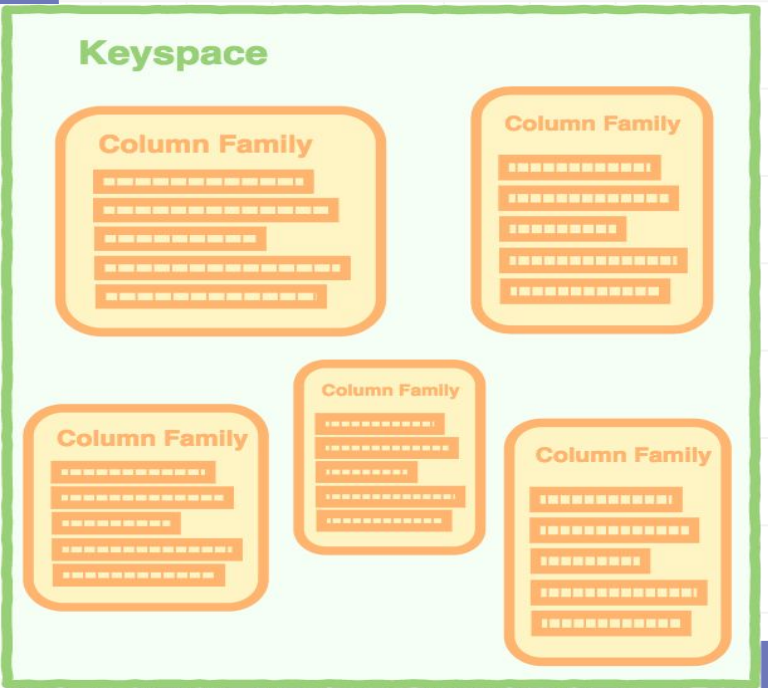
Name	ID
John	001
Karen	002
Bill	003

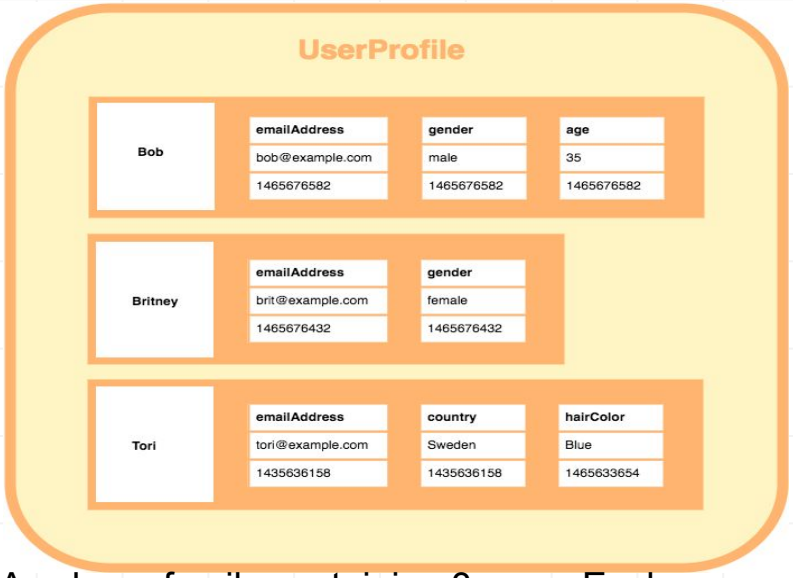
Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003

The Structure of a Column Store Database

Columns store databases use a concept called a keyspace.
A keyspace is kind of like a schema in the relational model.
The keyspace contains all the column families (kind of like tables in the relational model), which contain rows, which contain columns

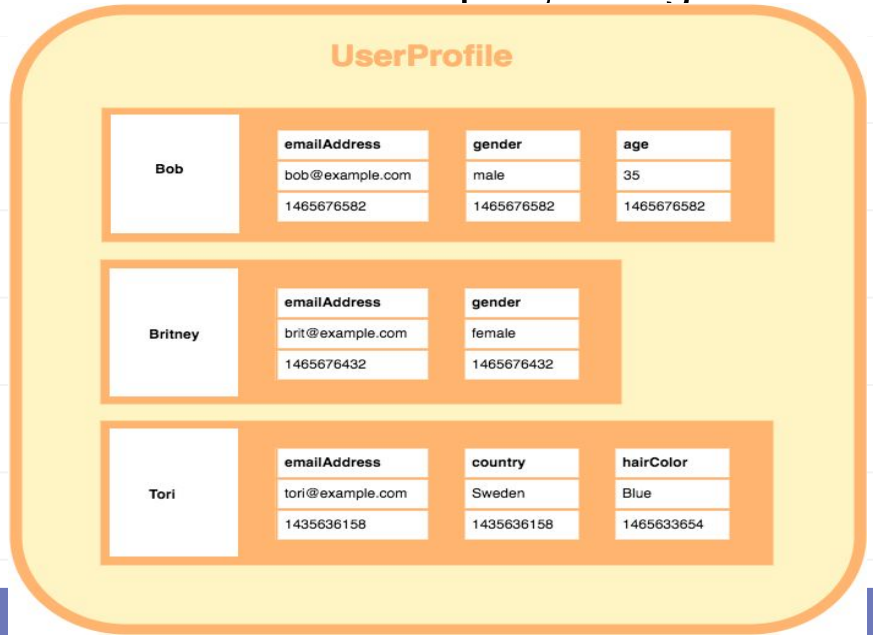


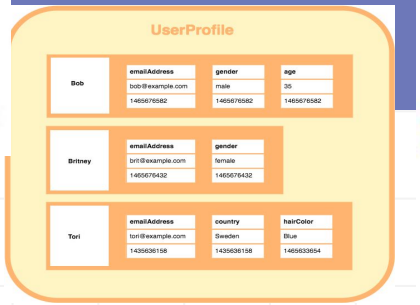


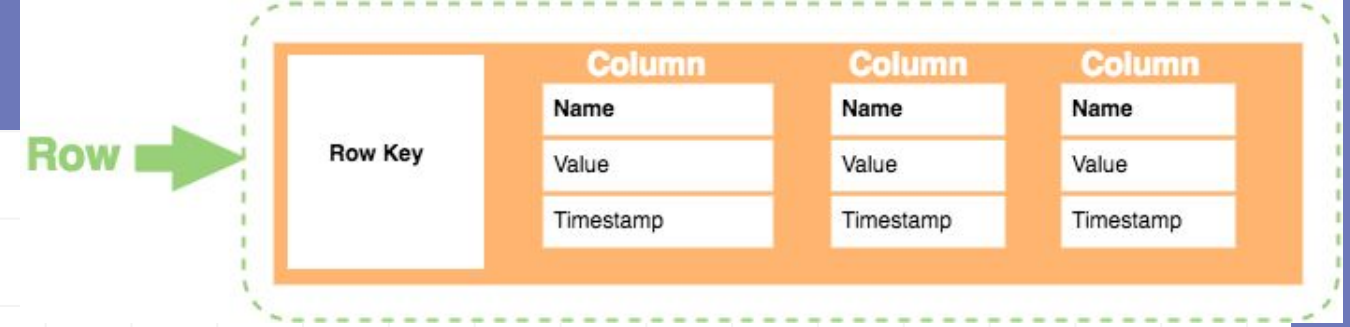
A column family containing 3 rows. Each row contains its own set of columns.

A column family consists of multiple rows.
Each row can contain a different number of columns to the other rows.
And the columns don't have to match the columns in the other rows (i.e. they can have different column names, data types, etc).
Each column is contained to its row. It doesn't span all rows like in a relational database.
Each column contains a name/value pair, along with a timestamp.

https://youtu.be/fnvsAj1-z2g







Row Key. Each row has a unique key, which is a unique identifier for that row.
Column. Each column contains a name, a value, and timestamp.
Name. This is the name of the name/value pair.
Value. This is the value of the name/value pair.
Timestamp. This provides the date and time that the data was inserted. This can be used to determine the most recent version of data.

- Example: (Cassandra column family--timestamps removed for simplicity)

```
UserProfile = {
    Cassandra = { emailAddress:"casandra@apache.org" , age:"20"}
    TerryCho = { emailAddress:"terry.cho@apache.org" , gender:"male"}
    Cath = { emailAddress:"cath@apache.org" ,
             age:"20",gender:"female",address:"Seoul"}
}
```

Benefits of Column Store Databases

Compression. Column stores are very efficient at data compression and/or partitioning.

Aggregation queries. Due to their structure, columnar databases perform particularly well with aggregation queries (such as SUM, COUNT, AVG, etc).

Scalability. Columnar databases are very scalable. They are well suited to massively parallel processing (MPP), which involves having data spread across a large cluster of machines – often thousands of machines.

Fast to load and query. Columnar stores can be loaded extremely fast. A billion row table could be loaded within a few seconds. You can start querying and analysing almost immediately.

Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, Library card catalogs,

HBase, Cassandra, Hypertable are NoSQL query examples of column based database.

Document-Oriented Database:

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document.

- The document is stored in JSON, XML, other semi-structured formats.
- JSON: JavaScript Object Notation is a data model, key-value pairs, which supports objects, records, structs, lists, array, maps, dates, Boolean with nesting
- The value is understood by the DB and can be queried.
- Documents are organized in a structure called collection.
- To make an analogy, document can be considered as a row in a table and collection can be considered as an entire table.

Document 1

```
{
  "id": "1",
  "name": "John Smith",
  "isActive": true,
  "dob": "1964-30-08"
}
```

Document 2

```
{
  "id": "2",
  "fullName": "Sarah Jones",
  "isActive": false,
  "dob": "2002-02-18"
}
```

Document 3

```
{
  "id": "3",
  "fullName":
  {
    "first": "Adam",
    "last": "Stark"
  },
  "isActive": true,
  "dob": "2015-04-19"
}
```

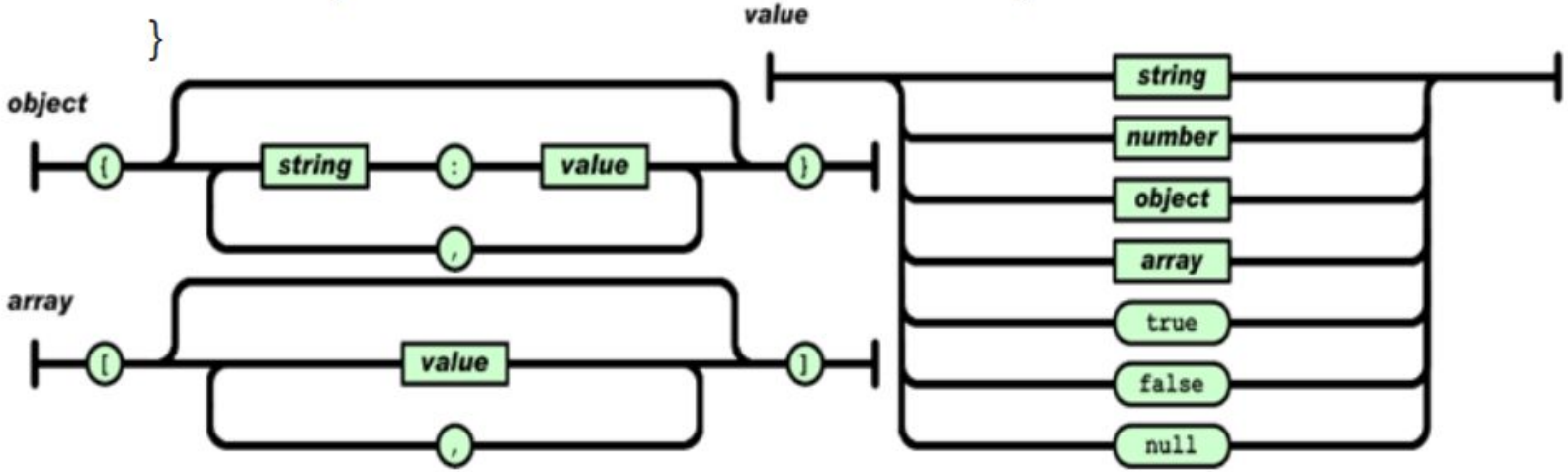
- Example: (MongoDB) document
  - {Name:"Jaroslav",

    Address:"Malostranske nám. 25, 118 00 Praha 1”,

    Grandchildren: {Claire: "7", Barbara: "6", "Magda: "3", "Kirsten: "1", "Otis: "3", Richard: "1“}

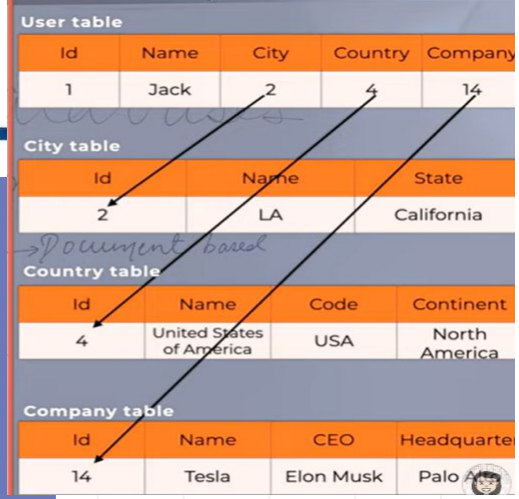
    Phones: [“123-456-7890”, “234-567-8963”]

    }



```
{
    "_id": "tomjohnson",
    "firstName": "Tom",
    "middleName": "William",
    "lastName": "Johnson",
    "email": "tom.johnson@digitalocean.com",
    "department": ["Finance", "Accounting"]
}
```

```
{
    "_id": "tomjohnson",
    "firstName": "Tom",
    "middleName": "William",
    "lastName": "Johnson",
    "email": "tom.johnson@digitalocean.com",
    "department": ["Finance", "Accounting"],
    "socialMediaAccounts": [
        {
            "type": "facebook",
            "username": "tom_william_johnson_23"
        },
        {
            "type": "twitter",
            "username": "@tomwilliamjohnson23"
        }
    ]
}
```




```
{
   "_id": "tomjohnson",
   "firstName": "Tom",
   "middleName": "William",
   "lastName": "Johnson",
   "email": "tom.johnson@digitalocean.com",
   "department": ["Finance", "Accounting"],
   "socialMediaAccounts": [
      {
         "type": "facebook",
         "username":
"tom_william_johnson_23"
      },
      {
         "type": "twitter",
         "username": "@tomwilliamjohnson23"
      }
   ]
}
```

biggest benefit of document-store is that everything is available in a single database, rather than having information spread across several linked databases

The document type is mostly used for CMS systems, blogging platforms, real-time analytics & e-commerce applications.

It should not use for complex transactions which require multiple operations or queries against varying aggregate structures.

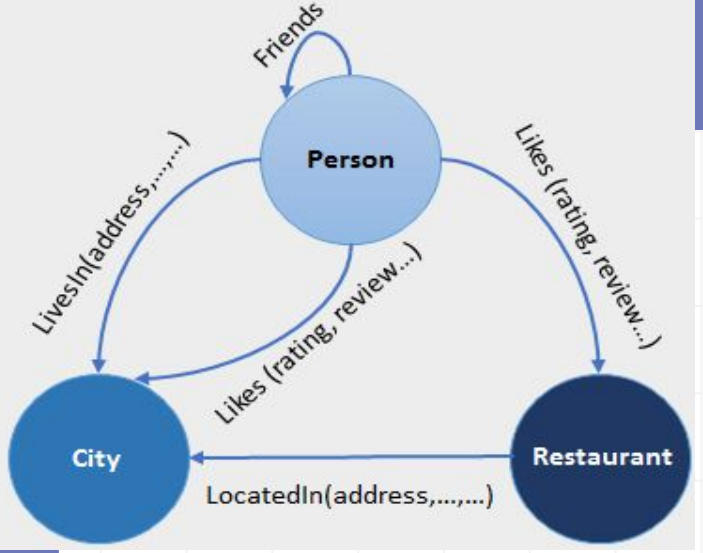
Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes are popular Document originated DBMS systems.

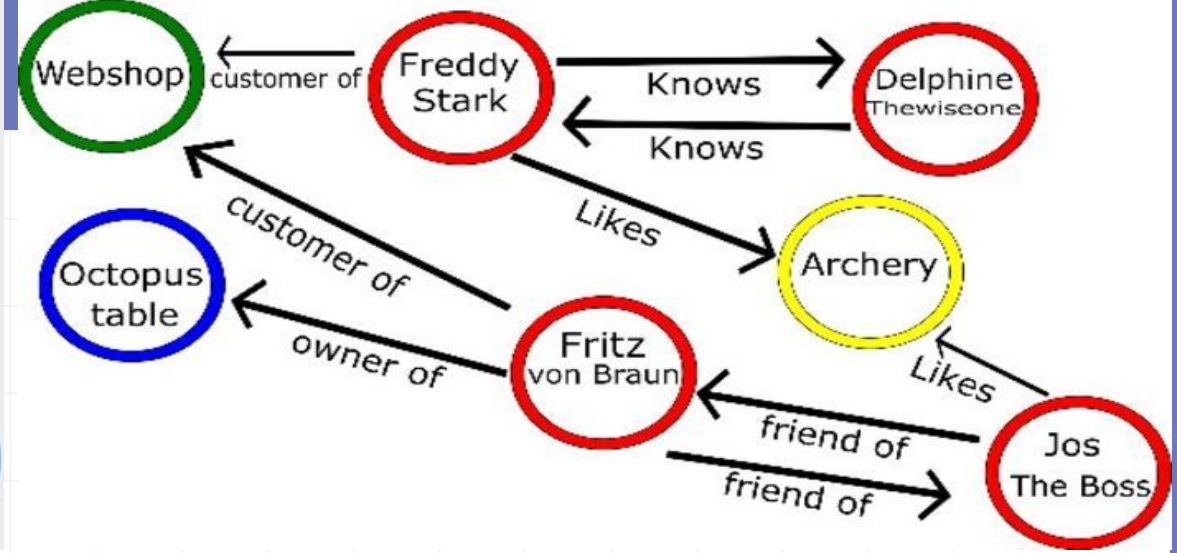
https://towardsdatascience.com/8-examples-to-query-a-nosql-database-fc3dd1c9a8c

Graph-Based Database

A graph type database stores entities as well the relations amongst those entities.
The entity is stored as a node with the relationship as edges.
An edge gives a relationship between nodes.
Every node and edge has a unique identifier.
graphs primarily work on the concept of multi-relational data ‘pathways’.

primarily are composed of two components:
The Node: This is the actual piece of data itself.
It can be the number of viewers of a youtube video, the number of people who have read a tweet, or it could even be basic information such as people’s names, addresses, and so forth.
The Edge: This explains the actual relationship between two nodes.
Interestingly enough, edges can also have their own pieces of information, such as the nature of the relation between two nodes. Similarly, edges might also have directions describing the flow of said data.





Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature. Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them.

Graph base database mostly used for social networks, logistics, spatial data.

Neo4J, Infinite Graph, OrientDB, FlockDB are some popular graph-based databases.

Document databases

Store data elements in document-like structures that encode information in formats such as JSON.

+

Common uses include content management and monitoring web and mobile applications.

+

EXAMPLES

Couchbase Server, CouchDB, MarkLogic, MongoDB

Graph databases

Emphasize connections between data elements, storing related "nodes" in graphs to accelerate querying.

+

Common uses include recommendation engines and geospatial applications.

+

EXAMPLES

AllegroGraph, Amazon Neptune, ArangoDB, IBM Db2 Graph, Neo4j

Key-value stores

Use a simple data model that pairs a unique key and its associated value in storing data elements.

+

Common uses include storing clickstream data and application logs.

+

EXAMPLES

Aerospike, Amazon DynamoDB, Azure Table Storage, Redis, Riak

Wide-column stores

Also called table-style databases, they store data across tables that can have very large numbers of columns.

+

Common uses include internet search and other large-scale web applications.

+

EXAMPLES

Accumulo, Cassandra, Google Cloud Bigtable, HBase, ScyllaDB

Variations of NoSQL architectural patterns

The key-value store, graph store, Bigtable store, and document store patterns can be modified by focusing on a different aspect of system implementation.

variations on

1. the architectures that use RAM or solid state drives (SSDs),
2. how the patterns can be used on distributed systems or modified to create enhanced availability.
3. how database items can be grouped together in different ways to make navigation over many items easier.

Customization for RAM or SSD stores

Some NoSQL products are designed to specifically work with one type of memory;
for example, Memcache, a key-value store, was specifically designed to see if items are in RAM on multiple servers.
A key-value store that only uses RAM is called a RAM cache;
it's flexible and has general tools that application developers can use to store global variables, configuration files, or intermediate results of document transformations.
A RAM cache is fast and reliable, and can be thought of as another programming construct like an array, a map, or a lookup system

SSD systems provide permanent storage and are almost as fast as RAM for read operations.

The Amazon DynamoDB key-value store service uses SSDs for all its storage, resulting in high-performance read operations.

Write operations to SSDs can often be buffered in large RAM caches, resulting in fast write times until the RAM becomes full.

Distributed stores

- NoSQL data architecture patterns vary as you move from a single processor to multiple processors that are distributed over data centers in different geographic regions.
- The ability to elegantly and transparently scale to a large number of processors is a core property of most NoSQL systems.
- the process of data distribution is transparent to the user, meaning that the API doesn't require you to know how or where your data is stored. But knowing that your NoSQL software can scale and how it does this is critical in the software selection process.

If your application uses many web servers, each caching the result of a long-running query, it's most efficient to have a method that allows the servers to work together to avoid duplication.

This mechanism, known as memcache.

Whether we use NoSQL or traditional SQL systems, RAM continues to be the most expensive and precious resource in an application server's configuration.

If you don't have enough RAM, your application won't scale.

The solution used in a distributed key-value store is to create a simple, lightweight protocol that checks whether any other server has an item in its cache.

If it does, this item is quickly returned to the requester and no additional searching is required

Grouping items

In the key-value store section, we looked at how web pages can be stored in a key-value store using a website URL as the key and the web page as the value.
You can extend this construct to file systems as well.
In a filesystem, the key is the directory or folder path, and the value is the file content.
But unlike web pages, filesystems have the ability to list all the files in a directory without having to open the files.
If the file content is large, it would be inefficient to load all of the files into memory each time you want a listing of the files.

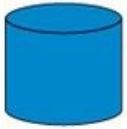
To make this easier and more efficient, a key-value store can be modified to include additional information in the structure of the key to indicate that the key-value pair is associated with another key-value pair, creating a collection, or general-purpose structures used to group resources

The implementation of a collection system can also vary dramatically based on what NoSQL data pattern you use.
Key-value stores have several methods to group similar items based on attributes in their keys.
Graph stores associate one or more group identifiers with each triple.
Big data systems use column families to columns.
Document stores use a concept of a document collection.

NoSQL solution for Big Data

- A big data class problem is any business problem that's so large that it can't be easily managed using a single processor.
- Big data problems force you to move away from a single-processor environment toward the more complex world of distributed computing.
- Though great for solving big data problems, distributed computing environments come with their own set of challenges

One database



- Easy to understand
- Easy to set up and configure
- Easy to administer
- Single source of truth
- Limited scalability

OR

Many databases

- Data partitioning
- Replication
- Clustering
- Query distribution
- Load balancing
- Consistency/Syncing
- Latency/Concurrency
- Clock synchronization
- Network bottlenecks/failures
- Multiple data centers
- Distributed backup
- Node failure
- Voting algorithms for error detection
- Administration of many systems
- Monitoring
- Scalable if designed correctly

Typical Big Data Use Cases

Bulk image processing —

- Organizations like NASA regularly receive terabytes of incoming data from satellites or even rovers on Mars.
- NASA uses a large number of servers to process these images and perform functions like image enhancement and photo stitching.
- Medical imaging systems like CAT scans and MRIs need to convert raw image data into formats that are useful to doctors and patients.
- Custom imaging hardware has been found to be more expensive than renting a large number of processors on the cloud when they’re needed.
- For example, the New York Times converted 3.3 million scans of old newspaper articles into web formats using tools like Amazon EC2 and Hadoop for a few hundred dollars.

- **Public web page data —**
- Publicly accessible pages are full of information that organizations can use to be more competitive.
- They contain news stories, RSS feeds(Really Simple Syndication) is a web feed that allows users and applications to access updates to websites), new product information, product reviews, and blog postings.
- Not all of the information is authentic.
- There are millions of pages of fake product reviews created by competitors or third parties paid to disparage other sites.
- Finding out which product reviews are valid is a topic for careful analysis.

Remote sensor data —

- Small, low-power sensors can now track almost any aspect of our world.
- Devices installed on vehicles track location, speed, acceleration, and fuel consumption, and tell your insurance company about your driving habits.
- Road sensors can warn about traffic jams in real time and suggest alternate routes.
- You can even track the moisture in your garden, lawn, and indoor plants to suggest a watering plan for your home

- **Event log data —**

Computer systems create logs of read-only events from web page hits (also called clickstreams), email messages sent, or login attempts. Each of these events can help organizations understand who's using what resources and when systems may not be performing according to specification.

Event log data can be fed into operational intelligence tools to send alerts to users when key indicators fall out of acceptable ranges.

- Mobile phone data
- Social media data
- Game data

They read and write to **key-value stores or distributed file systems like Amazon's Simple Storage Service (S3) or Hadoop Distributed File System (HDFS)** and may not need the advanced features of a document store or an RDBMS.
Depend upon demand and need different NoSQL structures are used

NoSQL solutions should

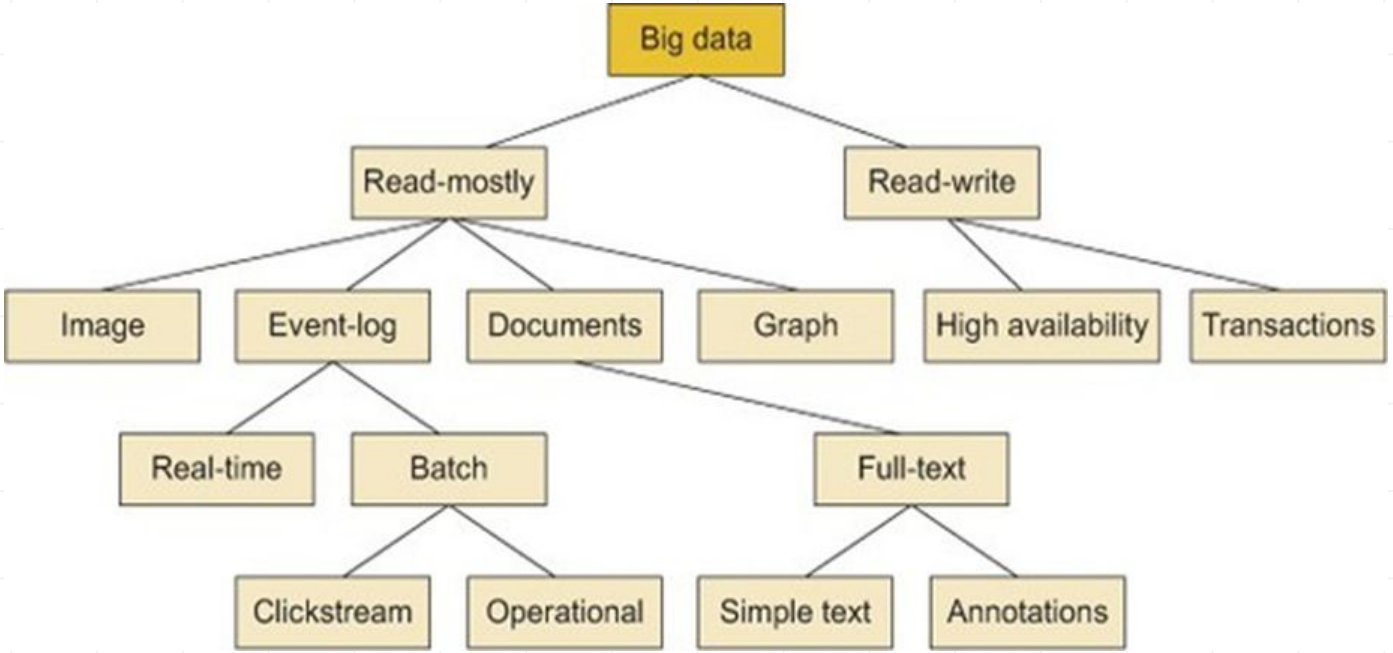
- Be efficient with input and output and scale linearly with growing data size.
- Be operationally efficient. Organizations can't afford to hire many people to run the servers.

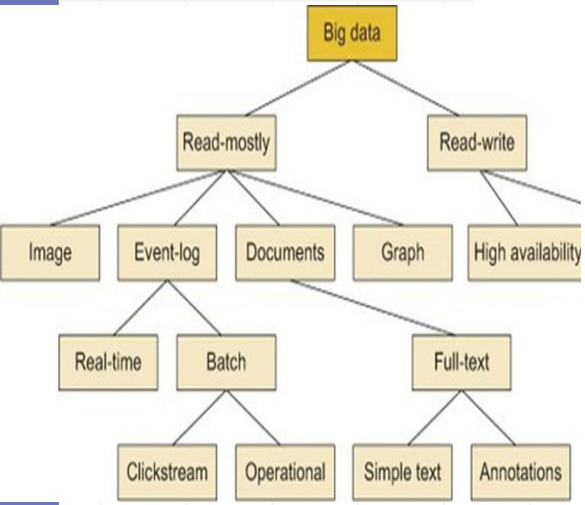
NoSQL solutions should

- Meet the challenges of distributed computing
- Require that reports and analysis be performed by non programmers using simple tools—not every business can afford a full-time Java programmer to write on-demand queries.

Understanding the types of big data problems

There are many types of big data problems, each requiring a different combination of NoSQL systems





- Read-mostly —Read-mostly data is the most common classification. It includes data that's created once and rarely altered.

This type of data is typically found in data warehouse applications but is also identified as a set of non-RDBMS items like images or video, event-logging data, published documents, or graph data.

Event data includes things like retail sales events, hits on a website, system logging data, or real-time sensor data.

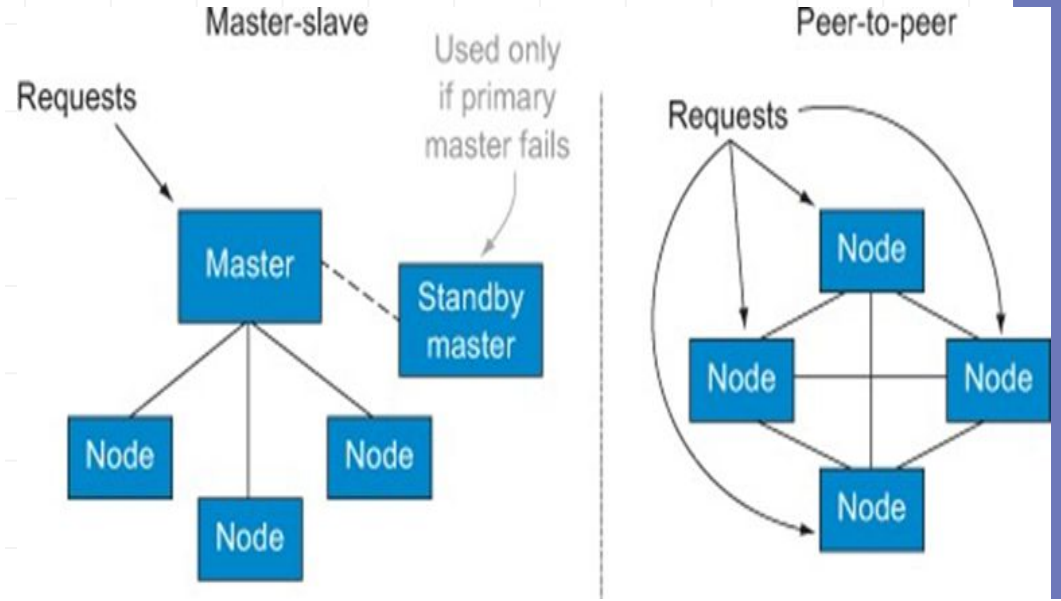
Full-text documents —This category of data includes any document that contains natural-language text like the English language.

An important aspect of document stores is that you can query the entire contents of your office document in the same way you would query rows in your SQL system.

Choosing distribution models: master-slave versus peer-to-peer

From a distribution perspective, there are two main models: master-slave and peer-to-peer.

- where all incoming database requests (reads or writes) are sent to a single master node and redistributed from there.
- The master node is called the NameNode in Hadoop.
- This node keeps a database of all the other nodes in the cluster and the rules for distributing requests to each node



the peer-to-peer model stores all the information about the cluster on each node in the cluster. If any node crashes, the other nodes can take over and processing can continue.

NoSQL systems handle big data problems

take a look at four of the most popular ways NoSQL systems handle big data challenges.

1. Moving queries to the data, not data to the queries

With the exception of large graph databases, most NoSQL systems use commodity processors that each hold a subset of the data on their local shared-nothing drives.

When a client wants to send a general query to all nodes that hold data, it's more efficient to send the query to each node than it is to transfer large datasets to a central processor.

2. Using hash rings to evenly distribute data on a cluster

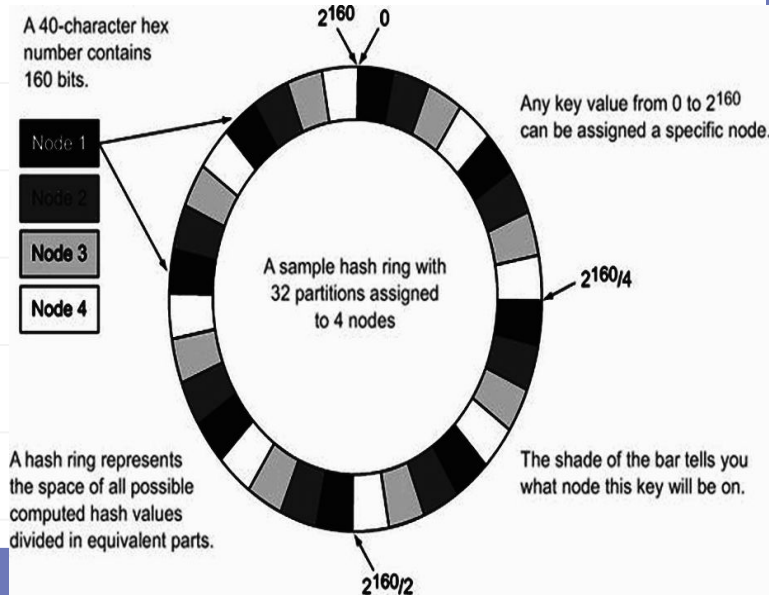
One of the most challenging problems with distributed databases is figuring out a consistent way of assigning a document to a processing node.

Using a hash ring technique to evenly distribute big data loads over many servers with a randomly generated 40-character key is a good way to evenly distribute a network load.

Using a hash ring to assign a node to a key that uses a 40-character hex number.

This number can be expressed in 2160 bits. The first bits in the hash can be used to map a document directly to a node.

This allows documents to be randomly assigned to nodes and new assignment rules to be updated as you add nodes to your cluster.



3. Using replication to scale reads

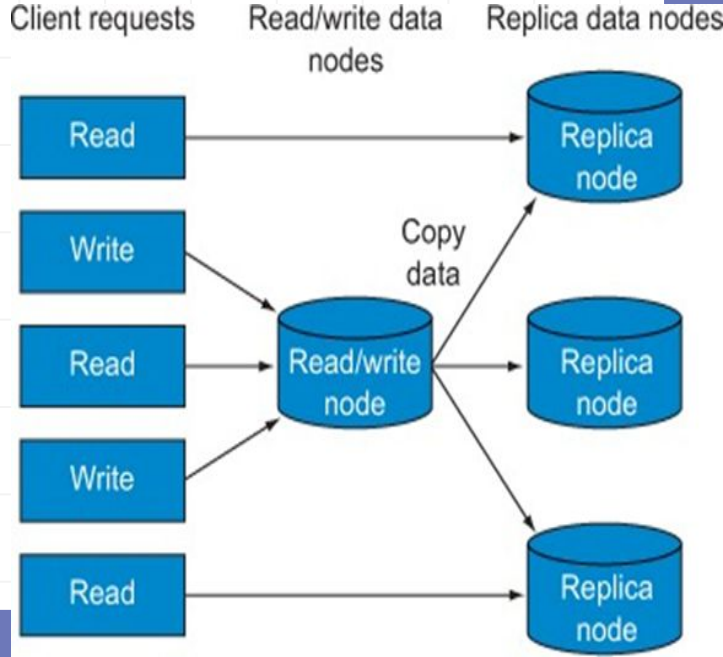
Till now we have seen how databases use replication to make backup copies of data in real time and how load balancers can work with the application layer to distribute queries to the correct database server. Now let's look at how using replication allows you to horizontally scale read requests

How you can replicate data to speed read performance in NoSQL systems.

All incoming client requests enter from the left. All reads can be directed to any node, either a primary read/write node or a replica node.

All write transactions can be sent to a central read/write node that will update the data and then automatically send the updates to replica nodes.

The time between the write to the primary and the time the update arrives on the replica nodes determines how long it takes for reads to return consistent results.



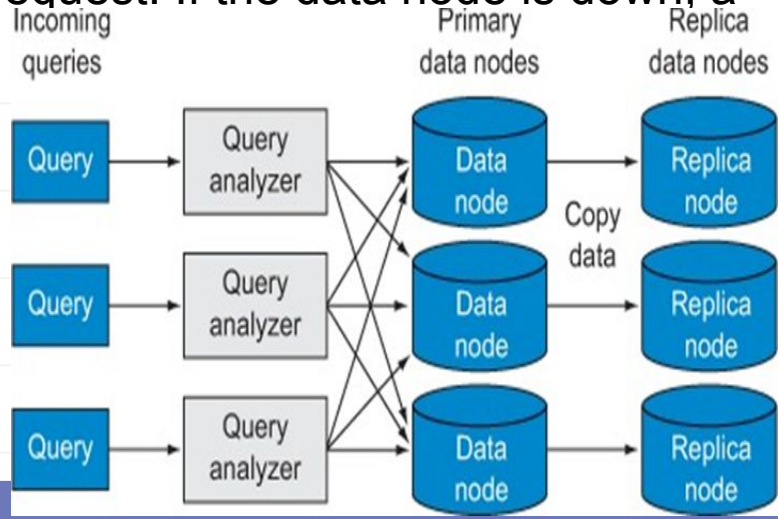
This replication strategy works well in most cases.
There are only a few times when you must be concerned about the lag time between a write to the read/write node and a client reading that same record from a replica

4. Letting the database distribute queries evenly to data nodes

In order to get high performance from queries that span multiple nodes, it's important to separate the concerns of query evaluation from query execution

NoSQL systems move the query to a data node, but don't move data to a query node. In this example, all incoming queries arrive at query analyzer nodes. These nodes then forward the queries to each data node. If they have matches, the documents are returned to the query node. The query won't return until all data nodes (or a response from a replica) have responded to the original query request. If the data node is down, a query can be redirected to a replica of the data node.

The approach shown in figure is one of moving the query to the data rather than moving the data to the query. This is an important part of NoSQL big data strategies. In this instance, moving the query is handled by the database server, and distribution of the query and waiting for all nodes to respond is the sole responsibility of the database, not the application layer.



Thank You!!

BIG
DATA
Semi-Structured
Volume
Metadata
Zettabyte
Framework
Text analytics
Semantic Metadata
VAST