UNIT 4 NoSQL and MongoDB

Introduction: NoSQL = Not Only SQL

- They are designed not to replace SQL databases, but to provide a different perspective in storing data
- MongoDB is an opensource document database and leading NoSQL database
- MongoDB is written in C++ language
- MongoDB is multipurpose dataset that is used for modern applications and cloud environments

Q: Depict Structured versus unstructured data

- Data is collection of raw facts and figures→ processed to produce information

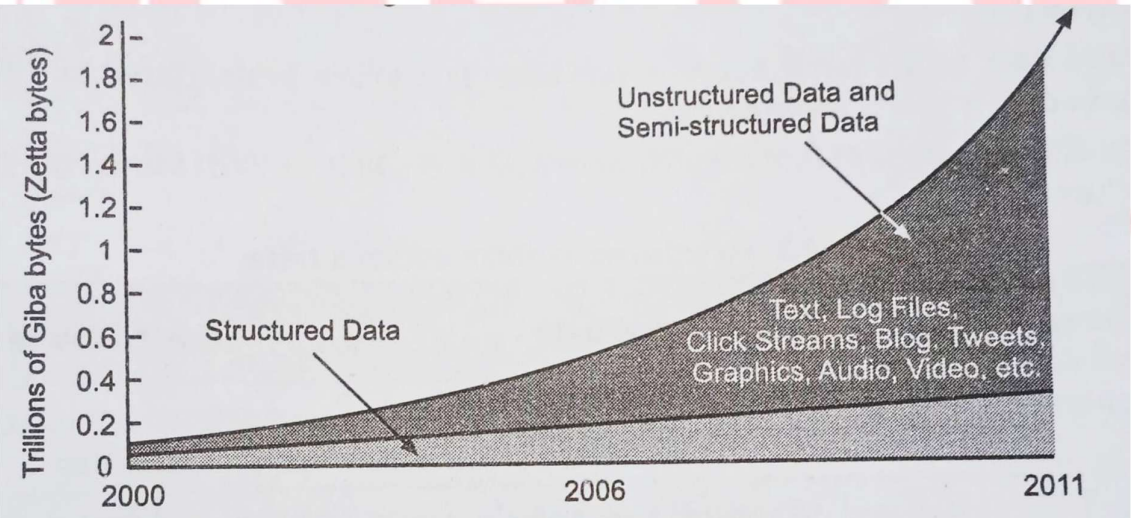


Fig. 4.1: The Rise of Unstructured and Semi-Structured Data Over the Years as Compared to Structured Data

[4.1]

Q: What are Types of data in NoSQL?

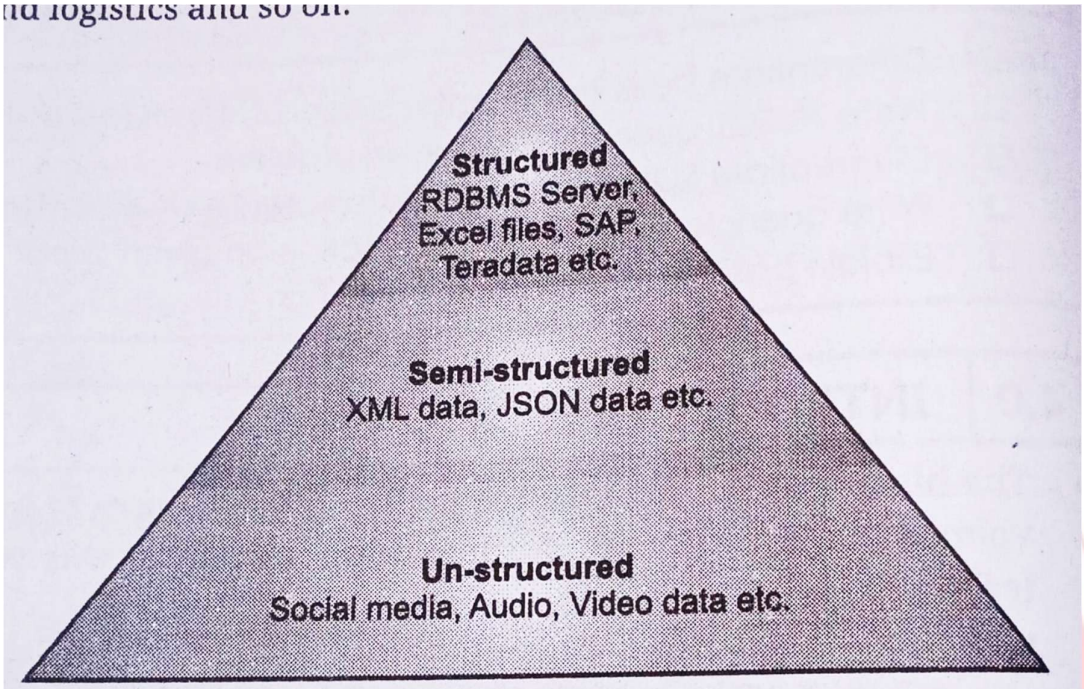


Fig. 4.2: Types of Data in NoSQL

1. **Structured data**

- Data that can be stored in fixed format(structure)
- Data is up-to-date, detailed, non-subjective and connected with past events
- E.g. excel files
- Structures data is – usually text files→columns with titles and rows with data
- This data is easy to be processed by data mining tools

2. **Unstructured data**

- Data with no fixed format
- Outdated, subjective in nature and related to future events
- Size is a big problem

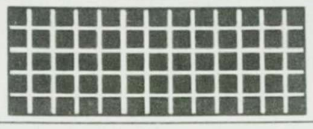
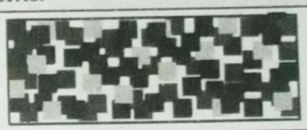
- Followed by difficulty in deriving values, getting results
- Challenging as compared to structured data
- E.g. video, audio, image, pdf, email
- Structured information can be derived from unstructured data
- But processing is time consuming

3. Semi-Structured data

- Combination of structured and unstructured data
- Different kind of data together
- E.g. XML, JSON

Q: Differentiate Between Structured data and Unstructured data

Table 4.1: Structured vs Unstructured Data

Sr. No.	Features	Structured Data	Unstructured Data
1.	Representation	Discrete rows and columns.	Less defined boundaries and easily addressable.
2.	Storage	Rational databases or spreadsheets, etc.	Unmanaged file structure.
3.	Metadata	Syntax.	Semantics.
4.	Integration tools	ETL or ELT.	Batch processing or Manual data entry that involves codes.
5.	Standard	SQL, ADO.NET, ODBC etc.	OpenXML, JSON, CSV etc.
6.	Databases	MySQL, Oracle etc.	Hadoop, MongoDB etc.
7.	Content	Text.	Text, Images, Audio, Video.
8.	Nature	Not subjective and concerned with past events.	Subjective related to future events.
9.	Diagram		

NoSQL Database concept

- NoSql database can answer challenges related to→ huge quantity, velocity, and variety of data → that is to be managed, searched and stored by modern database systems
-

Q: What is NoSQL?

- Not Only SQL
- Non-relational DBMS
- Designed for distributed data stores→ very large scale of data storing needs
- Facebook, Google
- It's an internet-scale database solution
- Its not a product or technology
- Not based on traditional RDBMS
- NoSQL→ provides process of storage and retrieval of data→ different that relational DB
- E.g. storing data in a document
- Data can be→ user information, social graphs, geographic locations etc.
- NoSQL db includes – MongoDB, Cassandra etc.
- It's an alternative for SQL and does not require any fixed table schemas
- It's a new generation database→ that addresses problem of Big Data era

Q: What are the General Characteristics of NoSQL?

- Provides high scalability, high availability and fault tolerance
- Supports distributed database architecture
- Thy are generated towards performance rather than transaction consistency
- Not based on relational model and SQL hence NoSQL
- Support very large amount of data

Q: Why do we use NoSQL? OR What is need of NoSQL?

- Data is being easier to access and capture through third party → Facebook, Google+ etc.
- Data has been increasing exponentially→ personal information, social graphs, geo location data, user generated contents, machine logging data etc.
- It is required to process huge amount of data
- SQL cannot handle it
- NoSQL can handle such data efficiently

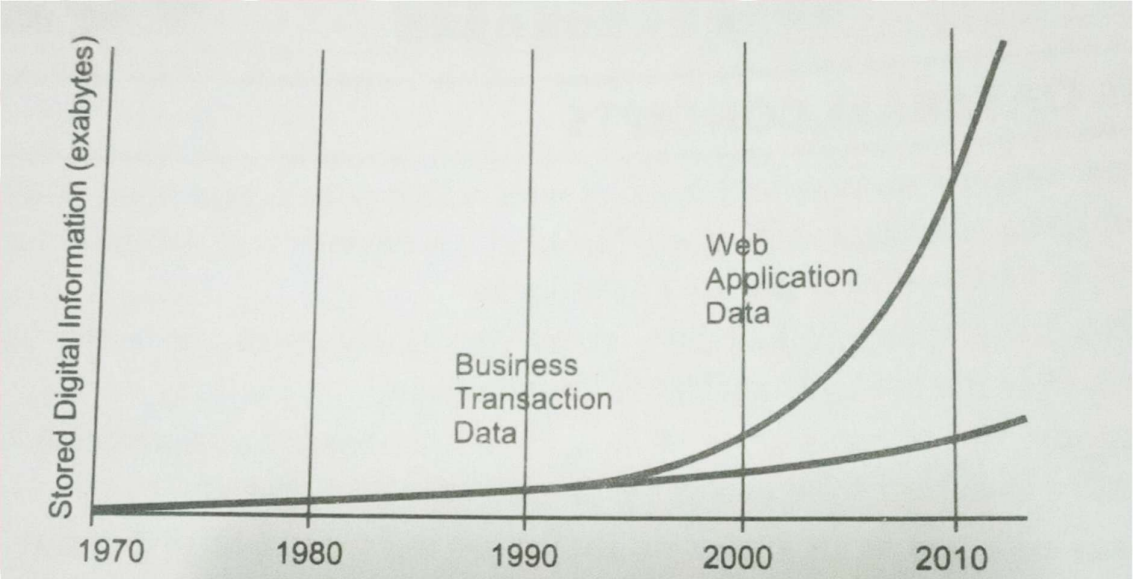


Fig. 4.3: Web Application Driving Data Growth

Q: Enlist Types of NoSQL database

1. **Document – oriented databases**
2. **Graph databases**
3. **Column – oriented databases**
4. **Key – value databases**

Q: Explain different types of NoSQL database

1. **Document – oriented databases/Document – Store databases**

- All data for a single entity can be stored as a document and documents can be stored together in collection
- Document can contain- all necessary information to describe an entity
- Documents in the collection are accessed via unique key

Data model for Document – oriented databases

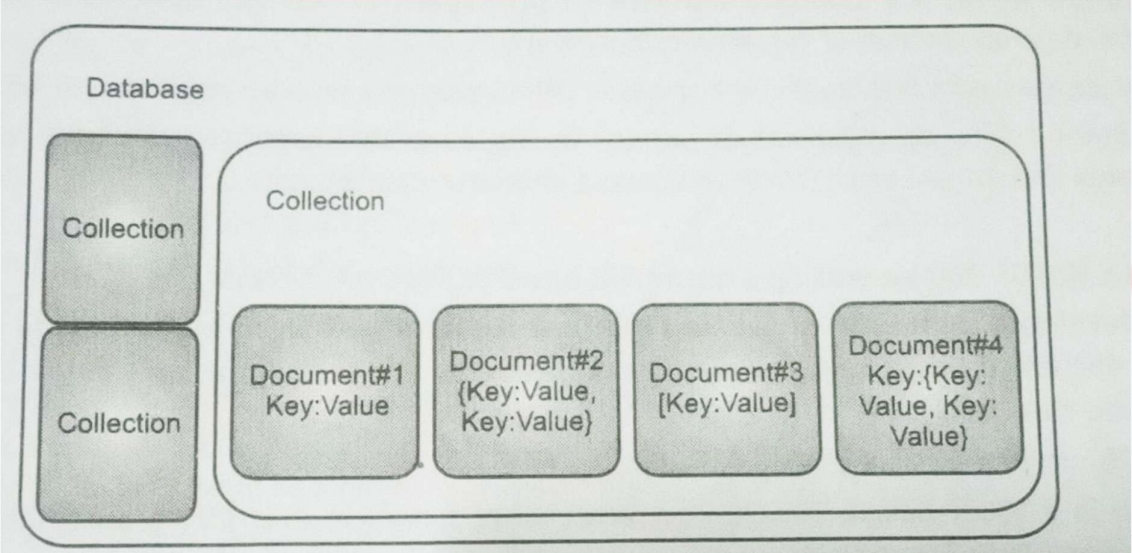


Fig. 4.4: Data Model for Document-Oriented Databases

- Unit of data is called – Document
- Table containing group of documents is called as- “collection”
- Fig. shows document-store database
- Database contains many collections
- Collection contains many documents
- each document might contain JSON document or XML document or Word document
- document database is suitable for Web based applications and applications exposing RESTful services

- (REST or Representational State Transfer is an architectural style that can be applied to web services to create and enhance properties like performance, scalability and modifiability.)

For ex.

1. **MongoDB:**

- Open-source database
- Used by companies and industries for wide variety of applications
- It's an agile database→ allows schemas to change quickly as application evolve
- Built for scalability, performance, high availability scaling→ from single server to large complex multisite architectures

2. **CouchDB:**

- It's a database that completely embraces Web
- Stores the data with JSON document
- Accesses the document and query indices with web browser→via HTTP, Index, →combine and transform document with JavaScript
- Works well in modern web and mobile apps
- We can serve web apps directly out of CouchDB
- Data and apps can be distributed using CouchDB's incremental replication

3. **Couchbase Server:**

- Originally known as Membase
- Open source, distributed, NoSQL document-oriented database→ optimized for interactive applications

- Applications must service→ many concurrent users for creating, sorting, retrieving aggregating manipulating and presenting data

4. MarkLogic Server:

- It's an enterprise NoSQL database
- It fuses together database internals, search style indexing and application server behaviors into a unified system
- Uses XML documents as its data model
- Stores documents within transactional repository
- Indexes the words and values from each of loaded documents

5. ClusterPoint Server:

- DB software of high-speed storage and large-scale processing of XML and JSON data → on clusters of commodity hardware
- Works as schema free document-oriented DBMS platform
- With Open-source AI
- Solves the problem of latency in Bigdata
- Billions of documents can be searched and analyzed fast

6. JasDB:

- It is a NoSQL database
- Uses document-based storage mechanism

7. RaptorDB:

- It is JSON based NoSQL document store database
- Offers automatic hybrid bitmap indexing and LINQ query filters

4.2.3.2 Graph databases/ Graph – store Database

- Database is represented as graphs
- Vertices and edges→ are nodes and relations in this graph
- Graph DB works better than directed graphs
- Used to store networked data

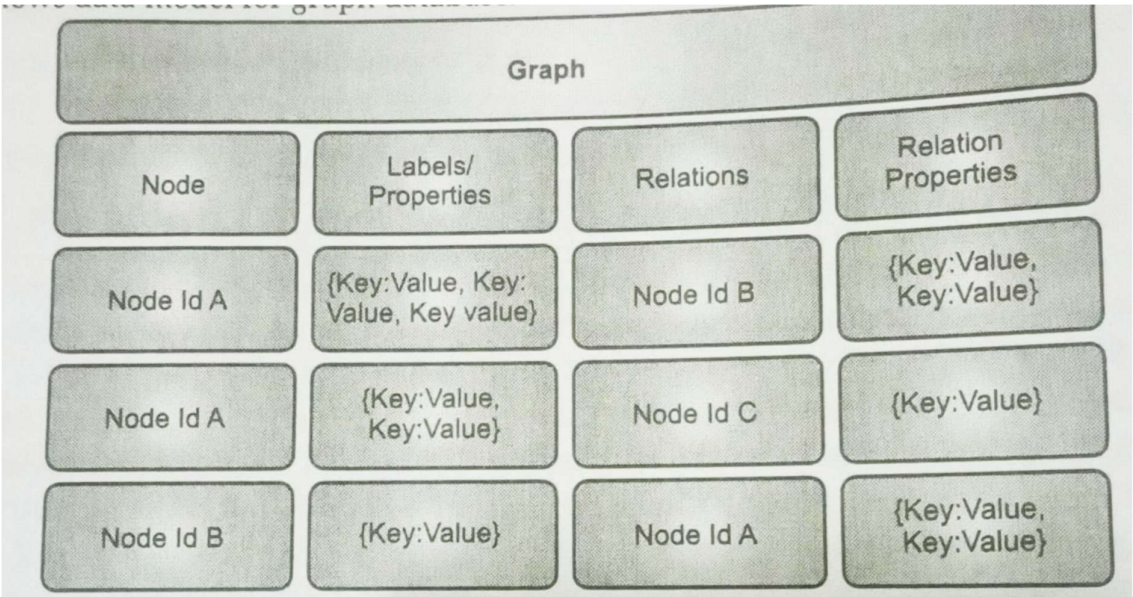


Fig. 4.6: **Data Model for Graph-Store Database**

Data Model for graph database

- Graph database is a 2-dimensional representation of graph
- Similar to tables
- Each graph contains- nodes, node properties, relation, relation properties ad columns
- There are values for each row for these columns
- Values for properties column-can have key-value pair
- Graph DB is suitable for- social media, network problems→ involving complex queries with more joins

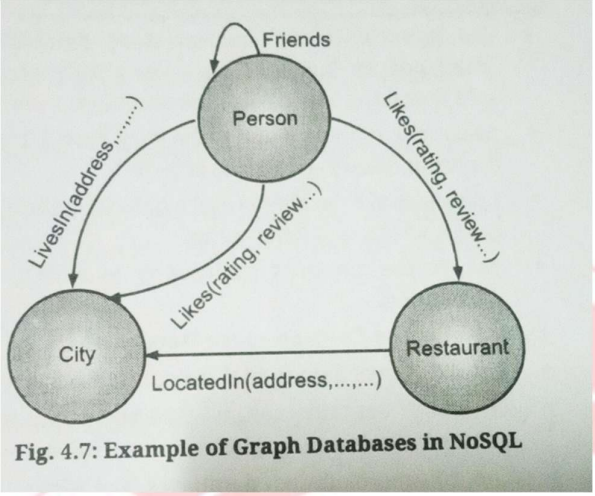


Fig. 4.7: Example of Graph Databases in NoSQL

For ex.

- Nodes have different type of relationship between them
- There is no limit to number and kind of relationship
- All can be represented in the same graph database
- Popular examples of graph database: -

1. Neo-4J:

- Java-based open-source NoSQL graph database
- Using Neo4J we can search social network data, connections between data are explored
- It can solve problems that require network probing
- Performance is high

2. Infinite Graph:

- Distributed graph database implemented in Java
- Belongs to the class of NoSQL
- Data technologies focused on graph data structure
- Graph data consist of objects or things(nodes) and relationship(edges) – connecting 2 or more nodes
- Developers may use infinite graph to build web and mobile applications

3. Allergo Graph:

- Modern, high performance, persistent graph database
- Efficient memory utilization with disk-based storage
- Can be scaled up to billions of quads → still maintaining superior performance
- Supports SPARQL, RDFS++ and prolog reasoning from numerous client apps

4. Virtuoso:

- It's a universal server for middleware and database engine
- Combines functionality of traditional RDBMS, ORDBMS, Virtual Database, RDF< XML<free-text, web development server and file server functionality→in a single system
- It's open-source edition known as Open Link Virtuoso

5. FlockDB:

- Open source distributed, fault tolerant graph data base
- Manages wide but shallow network graphs.
- Initially used by Twitter to store relationship between users
  - Ex. Followings and favorites

6. VertxDB:

- High performance graph database
- Supports automatic garbage collection
- Uses HTTP protocol for requests
- JASON for response data format
- Its API is inspired by FUSE file system API

4.2.3.3 Column – oriented databases/ Column – store databases.

- Development is based on Big Table White Paper published by Google

- Different approach than traditional RDBMS
- Supports more column and have wider table
- It supports group of columns with family name → "column family "or "Super column"
- Stores data in columns within a key space
- Key space is based on unique name, value, and timestamp.

Data model for Column – store databases

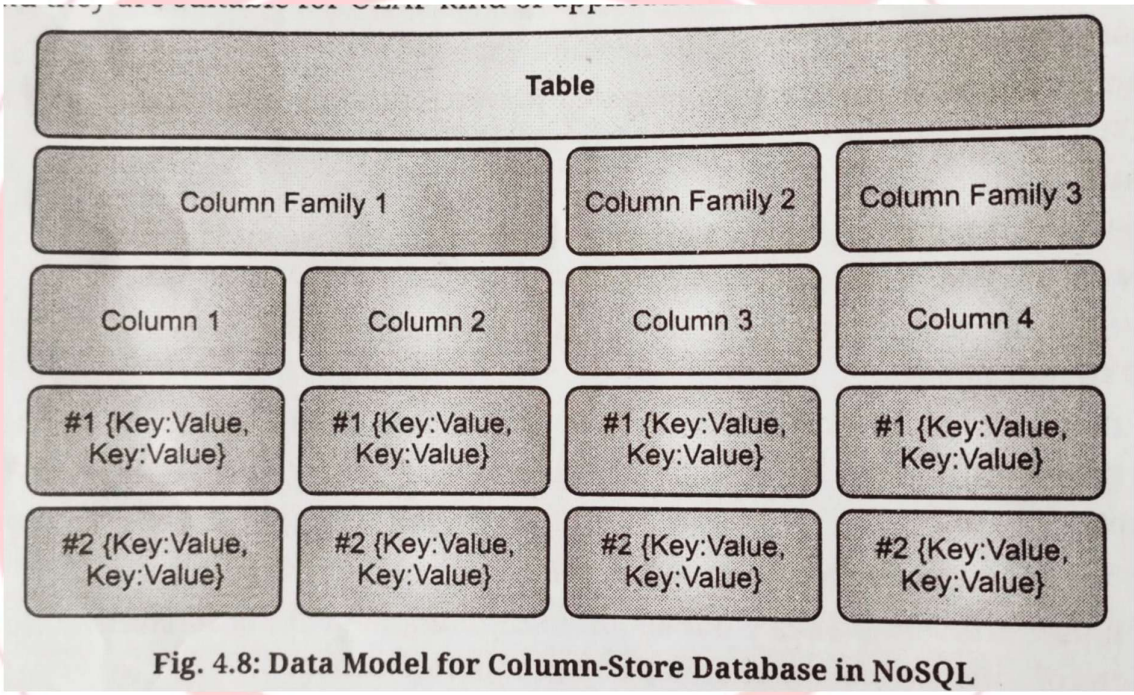


Fig. 4.8: Data Model for Column-Store Database in NoSQL

- Table contains column families
- Each column family contains many columns
- Values for column might be sparsely distributed with key – value pair
- This is an alternative to data warehousing database such as Teradata
- Suitable for OLAP (Online Analytical Processing) kind of application

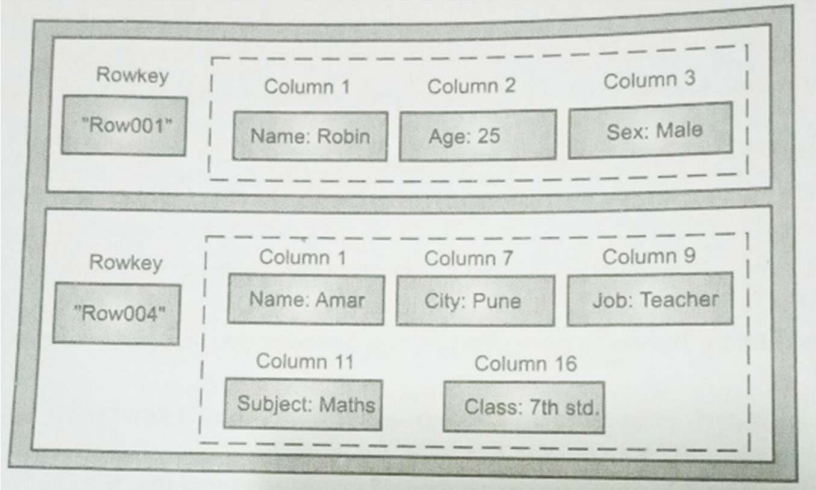


Fig. 4.9: **Example of Column-Store Database**

For ex.

1. **Hadoop/HBase:**

- Apache HBase is – open source, distributed, versioned, non-relational database→ modelled after Googles Big Table
- HBase provides Bigtable like capabilities on the top of Hadoop and HDFS

2. **Cassandra:**

- Apache Cassandra DB – scalable highly available without compromising performance.
- Linear scalability, fault tolerance
- Makes it perfect platform for mission-critical data
- Data is replicated at multiple data centres

3. **HyperTable:**

- High performance, open source massively scalable DB modelled after Bigtable

4. **Accumulo:**

- Based on Googles Big table design
- Built on top of apache Hadoop, Zookeeper and Thrift

- Provides scale-based access control and server-side programming mechanism

5. **Amazon SimpleDB:**

- Highly available, flexible, non-relational data store
- Offloads the work of database administration
- Developer can store and query data items via web services requests
- Using Amazon simple DB – we can focus on application development – without worrying about infrastructure, availability, maintenance, schema, index management, or performance tuning

4.2.3.4 Key – value databases/ Tuple – store databases

- Developed on the basis of – Dynamo whitepaper – published by Amazon
- Simplest type of NoSQL database
- Completely schema-less
- Key can point to any type of data
- Data is stored in simple key-value pair→<key>:<value>
- Key is used to retrieve the value from the table

Data model for Column – store databases

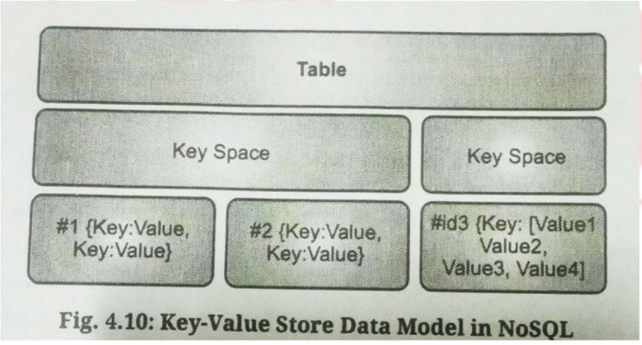


Fig. 4.10: Key-Value Store Data Model in NoSQL

- **Table:** contains many key spaces
- Key space: considered as rows

- Used for building – simple, non-complex, high available applications

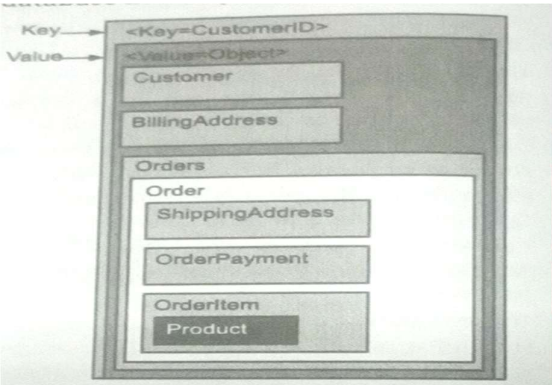


Fig. 4.11: Example of Key-Value Store Database in NoSQL

For ex.

1. **Redis:**

- It is a NoSQL key-value data store
- It is a data structure server

2. **Riak:**

- It uses simple key-value model for object storage
- Object consist of unique key and value
- It is stored in a namespace called bucket
- We can store- text, images, JSON/XML/HTML documents, user session data, backups, log files etc.

3. **Aerospike:**

- World fastest, more reliable in-memory open source NoSQL DB
- Operates with unprecedented speed
- Wide range of applications→from web portals to universal profile stores for real-time bidding an cross-channel marketing platforms

4. **FoundationDB:**

- Supports ACID transactions
- High performance with NoSQL and highly scalable
- Supports global transactions over any number of keys

5. **Amazon DynamoDB:**

- Fast and fully managed by NoSQL DB
- Simple and cost efficient
- Can retrieve any amount of data and serve any amount of request traffic
- It's a great fit for gaming, ad tech, mobile etc.

6. **Berkeley DB:**

- BDB- it's a s/w library with high performance embedded db
- It is written in C with API bindings for C++, C#, PHP, JAVA< Perl, Python, Ruby, Tcl, Smalltalk etc.
- Supports multiple data items with single key
- It's a non-relational DB
- Supports thousands of threads and concurrent processes

7. **Oracle NoSQL:**

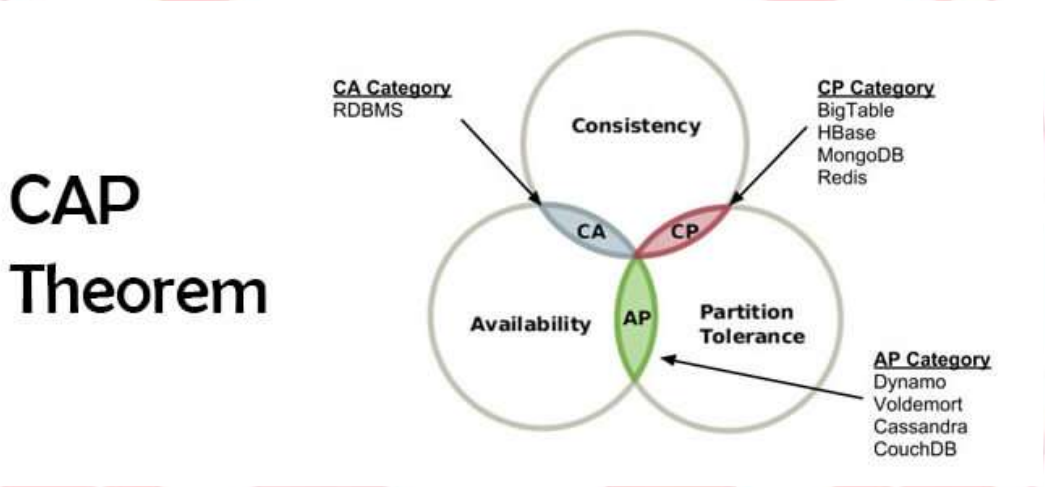
- Distributed key-value database
- **Features :** reliability, scalability, availability
- Data is stored on nodes
- Uses hash value of primary key
- Storage nodes are replicated to ensure high availability
- Applications are written in JAVA/C API to read and write data

CAP and BASE Theorems/Brewer Theorem

- NoSQL follows CAP theorem → also called as Brewer Theorem
- Basic requirements of CAP theorem are:

1. **Consistency:** data should remain consistent after any kind of transaction on DB
2. **Availability:** system is always available without any down-time
3. **Partition tolerance:** system must function reliably after partitioning of server though they may not communicate with each other

- **CAP suggests to fulfill at least 2 requirements**



1. **CA (Consistency & Availability)**

- Nodes are always in contact and are available
- System fails if partition occurs

2. **CP (Consistency with Partitioning)**

- After partitioning – consistent data will be available→may not be accessible all the time

3. **AP (Availability with Partitioning)**

- System is available under partitioning
- But some of the data may be incorrect

BASE Theorem

- **Basically Available (BA):** system will be available in terms of CAP theorem
- **Soft (S) state:** indicates that→ even if no input is provided to the system- state will change over time
- **Eventual (E) Consistency:** system will attain consistency in long run→provided no input is sent to the system during that time

4.2.4 NoSQL Data Modeling

"A query first data modelling technique that puts emphasis on application workflow with the ultimate goal of defining and iterating schemas that match real world application usage"

Definitions

Query First

- NOSQL data modelling focus on the information that a user is most likely to request taking different action on applications
- This data is stored independently →can be duplicated to be served faster

Emphasis on application workflow

- NoSQL focuses upon users needs as well as app development workflow→ allowing flexibility in object schema updates and data duplication rather than enforcing strict schema, relationship and application architecture like SQL

Defining and iterating schemas that match real-world usage

- As we develop applications- we deal with user feedback regarding to missing features, bugs, etc.→with NoSQL data modelling → iterating your schema to match these needs and deliver new app release is considerably faster than SQL
- Fig shows various models of NoSQL:

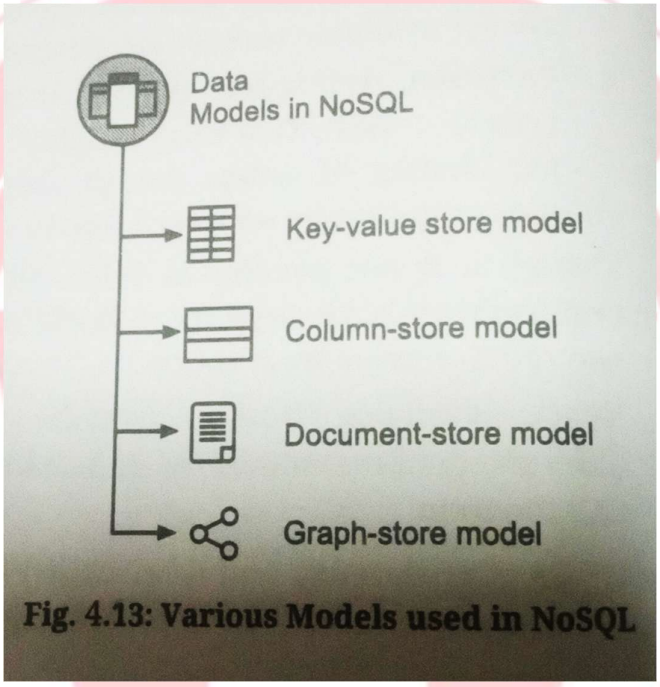


Fig. 4.13: Various Models used in NoSQL

Summery table

- NoSQL main data models, characteristics, and typical usages are explained in following tables:

Table 4.2: NoSQL Data Models

Sr. No.	Data Model	Characteristics	Typical Applications
1.	Column-store	Ideal for storing sparse tables. Rapid aggregation.	OLAP systems, data mining, data analytics etc.
2.	Document-oriented	Ideal to store any container like data in form of a hierarchical tree. Be indexable and searchable.	Online streaming, web pages, image and video storage etc.
3.	Key-value store	Ideal to associate large data files to a simple string. Simple interfaces.	Lookup tables, dictionaries, image and video stores etc.
4.	Graph-store	Ideal to store graph of data. Have a fast loop process. Not easy to scale-out.	Social networking, cloud management, security and access control management, logistics, etc.

- Following Table 4.3 lists features comparison of NoSQL data models:

Table 4.3: Feature comparison of NoSQL Data Models

Sr. No.	Feature	Column-Oriented	Document-Store	Key-Value Store	Graph-Store
1.	Table-like schema support (columns)	Yes	No	No	Yes
2.	Complete update/fetch	Yes	Yes	Yes	Yes
3.	Partial update/fetch	Yes	Yes	Yes	No
4.	Query/filter on value	Yes	Yes	No	Yes
5.	Aggregate across rows	Yes	No	No	No
6.	Relationship between entities	No	No	No	Yes
7.	Cross-entry view support	No	Yes	No	No
8.	Batch fetch	Yes	Yes	Yes	Yes
9.	Batch update	Yes	Yes	Yes	No

Q: What are the Benefits of NoSQL?

1. **Global Availability:** by replicating data across multiple servers, data centres or cloud resources
2. **Flexible Data modelling:** NoSQL implements flexible and fluid data models
3. **High Availability:** data is distributed equally among multiple resources → application remains available even if one node fails

4. **High Scalability:** NoSQL can quickly and non-disruptively change hardware
5. **Low cost:** designed to work with cheap commodity servers→ stores and processes the data at low cost
6. **Better Performance:** as compared to traditional RDBMS
7. **Variety of data:** supports any type of data→ structured , semi-structured, unstructured→ logs, images, audios, videos, graphs, JPEG, JSON, XML→ without any preprocessing
8. **Open-source:** NoSQL is open source
9. **NO schema or Fixed data models:** no predefined schema→ so format or data model change does not disrupt application
10. **Flexible data models:** can work with any type of data→ changes in the database schema can be easily implemented
11. **High Performance:** regarding data reading and writing

Q: What are the Disadvantages of NoSQL?

- Does not have defined standard
- Need to select appropriate data model for- document key value, wide column or graph
- GUI tools are not flexibly available in market

Q:Comparison between SQL and NoSQL database system

- Following is a list of differences between SQL and NoSQL database.

Table 4.4: Differences between SQL and NoSQL database

Sr. No.	SQL	NoSQL
1.	SQL stores data in the table.	NoSQL stores data in the form of document-store, key-value-store, graph-store etc.
2.	SQL provides standard platform to run complex queries.	NoSQL cannot provide any standard platform to run complex queries.
3.	It supports the structured and organized data.	It supports unstructured data.
4.	Strong consistency.	Dependent on the product. Few chose to provide strong consistency whereas few provide eventual consistency.
5.	High level of enterprise support is provided.	Open source model. Support through third parties or companies building the open source products.
6.	Available through easy to use GUI interfaces.	Querying may require programming expertise and knowledge.
7.	It uses structured query language to manipulate database.	There is not declarative query language for manipulating data.
8.	SQL works on ACID (Atomicity, Consistency, Isolation and Durability) properties.	NoSQL follows CAP (Consistency, Availability and Partition tolerance) theorem.
9.	Data and its relationships are stored in separate tables.	NoSQL does not have any pre-defined schema.
10.	Examples include Oracle, SQLite, MySQL, PostgreSQL etc.	Examples include MongoDB, Cassandra, Hbase, Neo4j, BigTable etc.
11.	SQL databases are primarily called as Relational Databases (RDBMS).	NoSQL database are primarily called as non-relational or distributed database.
12.	SQL databases are vertically scalable.	NoSQL databases are horizontally scalable.
13.	SQL databases use a powerful language "Structured Query Language (SQL)" to define and manipulate the data.	In NoSQL databases, collection of documents is used to query the data. It is also called "Unstructured Query Language (UnQL)". It varies from database to database.
14.	SQL databases are best suited for complex queries.	NoSQL databases are not so good for complex queries because these are not as powerful as SQL queries.

Sr. No.	SQL	No-SQL
1	SQL databases are primarily called as Relational Databases (RDBMS)	NoSQL database are primarily called as non-relational or distributed database.
2	SQL databases are table based databases	NoSQL databases are document based, key-value pairs, graph databases or wide-column stores
3	SQL databases have predefined schema	NoSQL databases have dynamic schema for unstructured data
4	SQL databases are vertically scalable	NoSQL databases are horizontally scalable.
5	SQL databases uses SQL (structured query language) for defining and manipulating the data, which is very powerful	NoSQL database, queries are focused on collection of documents. Sometimes it is also called as UnQL (Unstructured Query Language)
6	SQL database examples:	NoSQL database examples:

	MySql, Oracle, Sqlite, Postgres and MS-SQL	MongoDB, BigTable, Redis, RavenDb, Cassandra, Hbase, Neo4j and CouchDb
7	SQL databases are good fit for the complex query intensive environment	NoSQL databases are not good fit for complex queries.
8	SQL databases are not best fit for hierarchical data storage.	NoSQL database fits better for the hierarchical data storage as it follows the key-value pair way of storing data similar to JSON data.
	SQL databases emphasizes on ACID properties	NoSQL database follows the Brewers CAP theorem
9	SQL databases as either open-source or close-sourced from commercial vendors.	NoSQL databases can be classified on the basis of way of storing data as graph databases, key-value store databases, document store databases, column store database and XML databases.

NoSQL Using MONGODB

- Mongo DB is leading NoSQL document store platform→ enables organizations to handle Big Data
- It is an open-source document DB
- Stores the data inside a collection
- Scalable, high-performance, highly available
- Stires data as binary JSON (Java Script Object Notation→ also known as BSON- Binary JSON
- It has different schemas

Following reasons of MONGODB became most popular NoSQL database.

1. **High scalability:** data can be shared across multiple servers
2. **High availability:** data replication makes it highly available
3. **Document – Oriented:** data is stored in the form of documents→ no need to convert from rows and columns
4. **No SQL Injection:** data is stores as objects so → no SQL injection possible as with SQL string
5. **High performance:** it is one of the high performing DB in the world today→ supports heavy traffic

Some of MongoDB features

1. **Document – oriented storage:** it has JSON-style documents with dynamic schemas with simplicity and power
2. **Wide set of queries:** it has a rich set of queries
3. **Easy to programme:** document-oriented storage→ easily programmable
4. **Scalability:** easily scalable across LAN and WAN
5. **Allows replacement:** it allows replacement of complete database or set of fields with its update() command
6. **Supports MapReduce:** it supports map/reduce framework for batch processing of data and aggregation operation
7. **Sharding:** if load increases it can be distributed to other nodes across computer network→called sharding

Data model for Mongodb

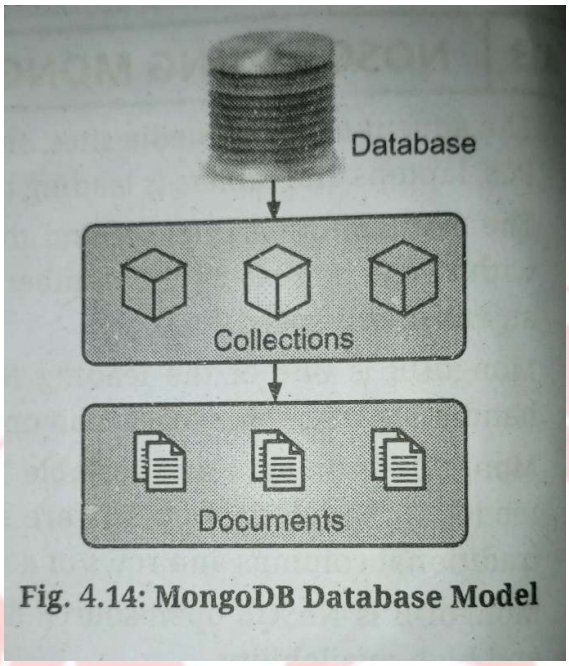


Fig. 4.14: MongoDB Database Model

MongoDB contains following elements

1. **Database:** it's a physical container for collections→each DB gets its own set of files on the file system
2. **Collection:**
    - it's a group of MongoDb docuents.
    - Equivalent to RDBMS tables
    - Collection exists withing a single database
    - Collections do not enforce schema
    - Document within a collection can have different fields
    - Documents within a collection are similar or related purpose
3. **Document:**
    - It is a set of key-value pairs
    - Documents have dynamic schema

- Dynamic schema: - documents in the same collection do not need to have the same set of fields of structure and common fields in collection documents – may hold different types of data

MongoDB stores data in BSON – Binary Encoded JSON Document which supports a huge rich collection type

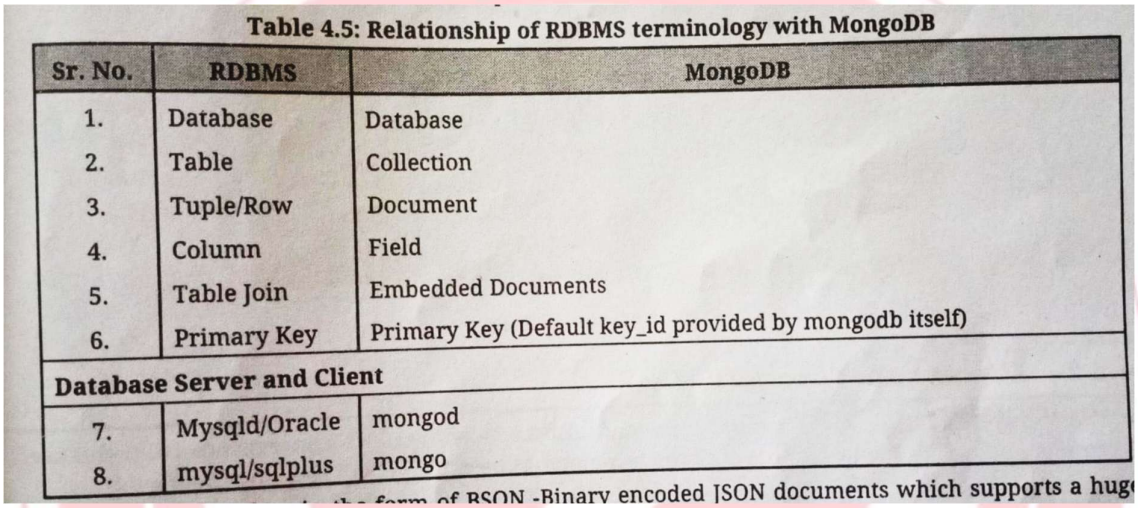


Table 4.5: Relationship of RDBMS terminology with MongoDB

Sr. No.	RDBMS	MongoDB
1.	Database	Database
2.	Table	Collection
3.	Tuple/Row	Document
4.	Column	Field
5.	Table Join	Embedded Documents
6.	Primary Key	Primary Key (Default key_id provided by mongodb itself)
Database Server and Client		
7.	Mysqld/Oracle	mongod
8.	mysql/sqlplus	mongo

form of BSON -Binary encoded JSON documents which supports a huge

Following are some of MongoDB benefits and strengths:

1. **Dynamic schema:** schema can be changed without modifying any existing data
2. **Scalability:** highly scalable according to business needs
3. **Manageability:** doesn't require any database administrator. (user friendly)
4. **Speed:** high performing speed
5. **Flexibility:** can add columns without affecting existing rows

Introduction to MongoDB Shell

- MongoDB is accessed using MongoDB shell
- It's a command line interface that uses JavaScript like syntax

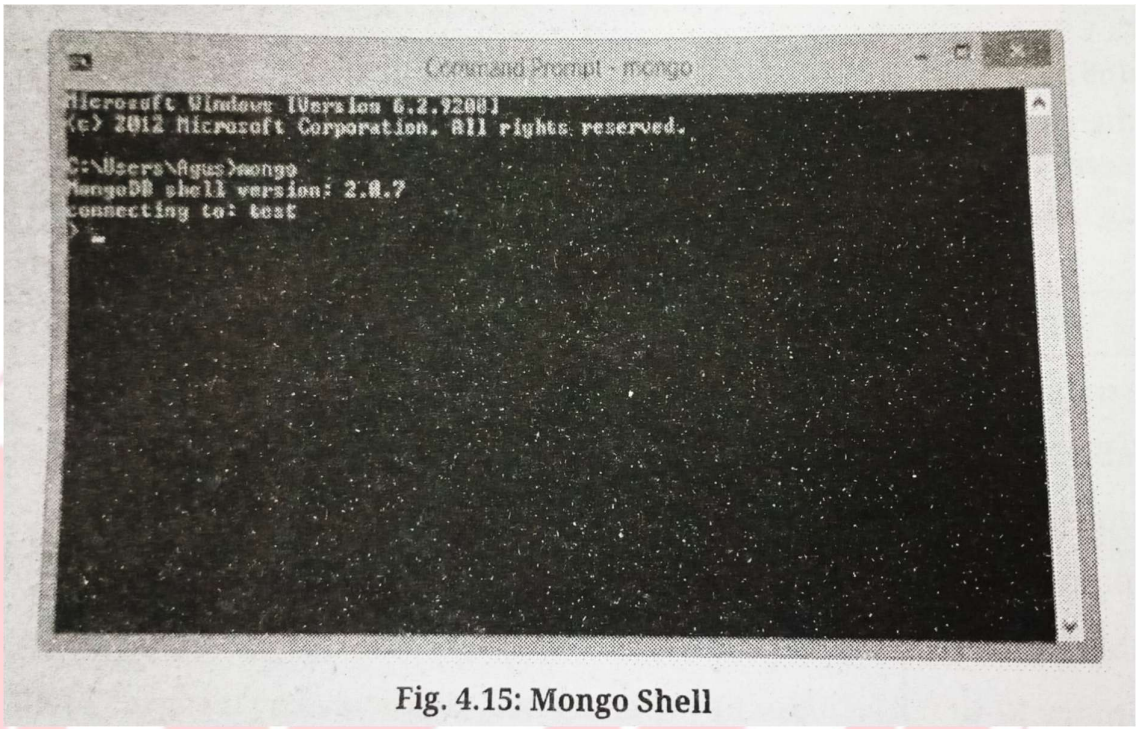
- To start MongoDB shell we must call Mongo shell
- On command prompt type mongo and press enter
- 

Fig. 4.15: Mongo Shell

Running MongoDB Shell

Several commands

1. **help <option>**

Displays help information for MongoDB shell commands.

Examples:

- help → General help for shell commands.
- db.help() → Help for database-related methods.
- db.collection.help() → Help for collection methods.

```
> help
  db.help()              →help on db methods
  db.mycoll.help()         →help on collection methods
```

show dbs →show database names
show collections →show collections in current database
exit →quit the mongo shell

2. **use <database>**

   Switches to a specific database. If the database doesn't exist, it will be created when you insert data.

   e.g.

   > use myDatabase

   switched to db myDatabase

3. **show <option>**

   Used to **list databases, collections, users, or roles**.

   I. **Dbs** show dbs → Lists all databases
   II. **Collections** show collections → Lists all collections in the current database.
   III. **Profile** Displays the **current database profiling information**, which helps monitor performance by showing slow queries or operations.
   IV. show users → Lists users in the current database
   V. show roles → Lists roles in the current database.
   VI. **log (name)** Displays the **specified log** from the MongoDB server. Useful for debugging or monitoring

4. **exist**

Closes the MongoDB shell session.

- Let us take a simple mathematical program:

```
>x= 100
100
>x/ 5;
20
```

- We can also use the JavaScript libraries:

```
> "Hello, World!".replace("World", "MongoDB");
    Hello, MongoDB!
```

- We can even define and call JavaScript functions:

```
> function factorial (n) {
... if (n <= 1) return 1;
... return n * factorial(n - 1);
... }
> factorial (5);
120
```

Q: Explain MongoDB client

A MongoDB client is a software tool or library that allows users or applications to connect to a MongoDB server and perform operations such as storing, retrieving, updating, and deleting data.

Types of MongoDB Clients:

1. MongoDB Shell (mongosh): A command-line interface used to interact with MongoDB using JavaScript-like syntax.
2. MongoDB Compass: A graphical user interface (GUI) that helps users visualize data, run queries, and manage collections easily.

3. Programming Language Drivers: MongoDB provides official drivers for languages like Python (pymongo), Node.js (mongodb), Java, C#, and more. These drivers allow developers to integrate MongoDB into their applications.

   Purpose:

- Enables communication between the user and the MongoDB server.
- Supports CRUD operations, indexing, aggregation, and administrative tasks.

Basic operations with MongoDB shell

The basic operations of Mongo DB are CRUD operations.

Create, Read, Update & delete documents.

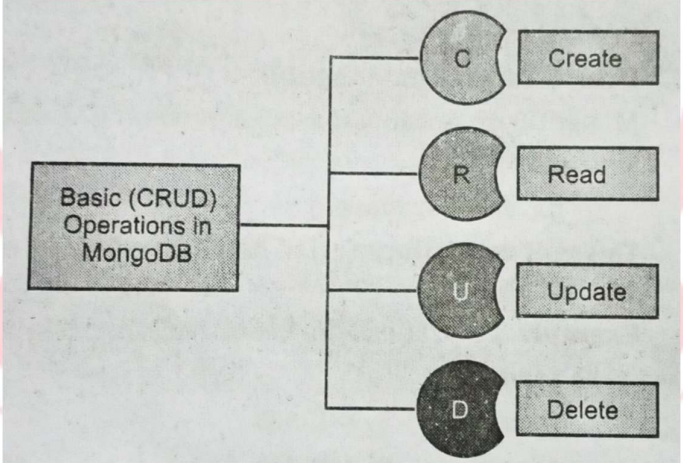


Fig. 4.16: CRUD Operations in MongoDB

CRUD operations

1. **Create operation**

The create or insert operations are used to addnew document into the collection.

If the collection is not exist, then the insert operation will create the
collection.

The different methods to insert document into a collection:

- db.collection.insertone ( )
- db.collection.insertmany ( )

E.g:

Db.student.insertOne
(
{
Name:”Kunal”
Age: “17”
Status: “file pending”
grade: “A”
}
)

2. **Read operation**

It is used to retrieve the documents from the collection.

The find ( ) command is used to queries a collection for documents or
simply to retrieve the documents from the collection.

Syntax:

Db.collection.find ( )

E.g:

db.student.find ( ) – for all documents
or you can retrieve specific document with the help of attributes of that

document.

E.g:

```
db.student.find(
{
name:”Kunal”
}
)
```

3. **Update operation**

This operation is used to modify the existing documents in a collection.

Different methods are used for updation as

-db.collection.updateone ( )

-db.collection.updateMany ( )

-db.collection.replaceOne ( )

The Mongo DB uses the update operation for a single collection. One can update the all documents without specifying any criteria.

One you can update specific document by providing specific criteria.

E.g: db.collaction.updateMany ( )

E.g:

```
db.student.updateOne (
{
Name:{$ SN: “Kunal”} update filter
},
{ $ set: { status: “completed”} update action.
}
)
```

4. **Delete operation**

It is used to remove documents from a collection.

Different methods to delete documents are

-db.collection.deleteOne ( )
-db.collection.deleteMany ( )
The delete operation is performed on a single collection.
To delete the specific document you have to provide the specified criteria as per requirements.
E.g:

```
db.student.deleteMany ( ) or
db.student.deleteOne(
{
Name:{$SN: “Kunal”}
}
)
```

Lets see in details

1 Creating database in MongoDB

MongoDB creates a database when you switch to it and insert data.

Syntax:

use myDatabase

This switches to myDatabase. It will be created when you insert the first document.

MongoDB creates the database when you insert the first document.

2. Drop database in MongoDB

Deletes the current database and all its collections.

Syntax:

db.dropDatabase()

Use this only when you're sure you want to remove the entire database.

This removes the database and all its collections permanently.

3. Create collection in MongoDB

To create a new collection:

db.createCollection("students")

Alternatively, a collection is created automatically when a document is inserted.

4. Drop Collection

To delete a specific collection:

db.students.drop()

This removes the collection and its documents.

5. Insert (Create) Document

To insert one document:

db.students.insertOne({ name: "Amit", age: 20 })

To insert multiple documents:

```
db.students.insertMany([
 { name: "Riya", age: 21 },
 { name: "Karan", age: 22 }
])
```

6. Query Document

To retrieve documents:

db.students.find({ age: { $gt: 20 } })

Table Query operations

Operator	Description	Example
$eq	Equal to	{ age: { $eq: 20 } }
$gt	Greater than	{ age: { $gt: 18 } }
$lt	Less than	{ age: { $lt: 25 } }
$in	Matches any value	{ name: { $in: ["Amit", "Riya"] } }
$and	Logical AND	{ $and: [{ age: 20 }, { name: "Amit" }] }
$or	Logical OR	{ $or: [{ age: 20 }, { age: 22 }] }

7. Update Document

(i) update () method

To update a field:

```
db.students.updateOne(
  { name: "Amit" },
  { $set: { age: 21 } }
)
```

(ii) save () method

To insert or update a document:

```
db.students.save({ _id: ObjectId("..."), name: "Amit", age: 22 })
```

8. Delete Document

(i) Remove only one

db.students.deleteOne({ name: "Amit" })

(ii) Remove all documents

db.students.deleteMany({})

Basic data types

1. **Integer** Stores whole numbers as either 32-bit or 64-bit signed integers. Suitable for numerical data like counts or ages.
2. **Boolean** Stores binary values (true or false) to represent states like active/inactive.
3. **Double** Used for floating-point numbers, ideal for storing decimal values like prices or percentages.
4. **String** Used to store text data in UTF-8 format. It is the most commonly used data type for textual information.
5. **Min/Max Keys**: Special types used for internal comparisons, representing the lowest and highest BSON elements.
6. **Object** (Embedded Document): Stores nested documents, enabling hierarchical data structures.
7. **Object Id**: A unique identifier automatically generated for each document. It is 12 bytes long and includes a timestamp, machine ID, process ID, and counter.
8. **Date** Stores date and time as a 64-bit integer, representing milliseconds since the Unix epoch
9. **Binary Data** Used for storing non-textual data like images or files in binary format.
10. **Symbol** Similar to strings but reserved for specific use cases in certain programming languages.
11. **Null** Represents the absence of a value, useful for optional fields.

12. **Array** Allows storing multiple values in a single field. Arrays can contain values of the same or different data types.
13. **Timestamp** Stores a 64-bit value for time tracking, often used for versioning.
14. **Regular Expression**: Stores regex patterns for querying string fields.

Arrays

Arrays are values which can be interchangeably referred for both ordered operating as lists, stack or queues or for unordered operations as sets.

Arrays in Mongo DB are able to store different data types values.

E.g:

{

“things”: [“pi”, 3.14]

}

Mongo DB enables atomic updates which helps to modify the contents

of arrays.

Embedded Documents

Definition:

Embedded documents are documents stored inside another document as a field. This allows MongoDB to represent related data

in a single document, improving read performance and reducing the need for joins.

Example:

Let's say we have a collection called students. Each student has an address stored as an embedded document.

```
db.students.insertOne({
  name: "Amit",
  age: 21,
  address: {
    street: "MG Road",
    city: "Pune",
    pincode: 411001
  }
})
```

Querying Embedded Documents

To find students living in Pune:

```
db.students.find({ "address.city": "Pune" })
```

Querying with MongoDB

Find () Function

Used to retrieve documents from a collection.

```
db.collection.find({ query }, { projection })
```

First parameter: query criteria

Second parameter: projection (optional – specifies which fields to return)

Simplying Which keys to return

Use projection to include or exclude fields.

db.students.find({ age: 20 }, { name: 1, _id: 0 })

name: 1 → include name

_id: 0 → exclude _id

Limitations

- Cannot perform complex joins like SQL
- Limited support for multi-document transactions (in older versions)
- Query performance depends on indexing

Query criteria

Defines conditions to match documents.

db.students.find({ age: { $gt: 18 } })

Query Conditions

MongoDB supports various operators:

Operator	Description	Example
$eq	Equal to	{ age: { $eq: 20 } }

Operator	Description	Example
$gt	Greater than	{ age: { $gt: 18 } }
$lt	Less than	{ age: { $lt: 25 } }
$ne	Not equal	{ age: { $ne: 20 } }

OR queries

Use $or to match any condition.

```
db.students.find({
  $or: [{ age: 20 }, { name: "Amit" }]
})
```

$not

Negates a condition.

```
db.students.find({
  age: { $not: { $gt: 25 } }
})
```

$ Conditional Semantics

MongoDB uses logical operators like $and, $or, $not, $nor to combine conditions.

```
db.students.find({
  $and: [{ age: { $gt: 18 } }, { city: "Pune" }]
})
```

Types Specific queries

Null

Query documents where a field is null:

```
db.students.find({ email: null })
```

Regular expressions

Match patterns in strings:

```
db.students.find({ name: /Amit/i })
```

Querying arrays

MongoDB supports powerful array queries.

$all

Match all values in array

```
db.students.find({ subjects: { $all: ["Math", "Science"] } })
```

$Size

Match array length

```
db.students.find({ subjects: { $size: 3 } })
```

$Slice operator

Return part of array (used in projection)

```
db.students.find({}, { subjects: { $slice: 2 } })
```

Returning a Matching Array element

Use $elemMatch to match specific array elements.

```
db.students.find({
  scores: { $elemMatch: { subject: "Math", marks: { $gt: 80 } } }
})
```

Array and Range Query Interactions

You can combine array queries with range conditions.

```
db.students.find({
  marks: { $gt: 50, $lt: 90 }
})
```

Querying on Embedded Documents

Use dot notation to access nested fields.

```
db.students.find({ "address.city": "Mumbai" })
```

$Where queries

Use JavaScript expressions for complex logic.

```
db.students.find({
  $where: "this.age > 18 && this.city == 'Pune'"
})
```

→where is slower and should be used carefully.