
mongoDB.

What is Database ?

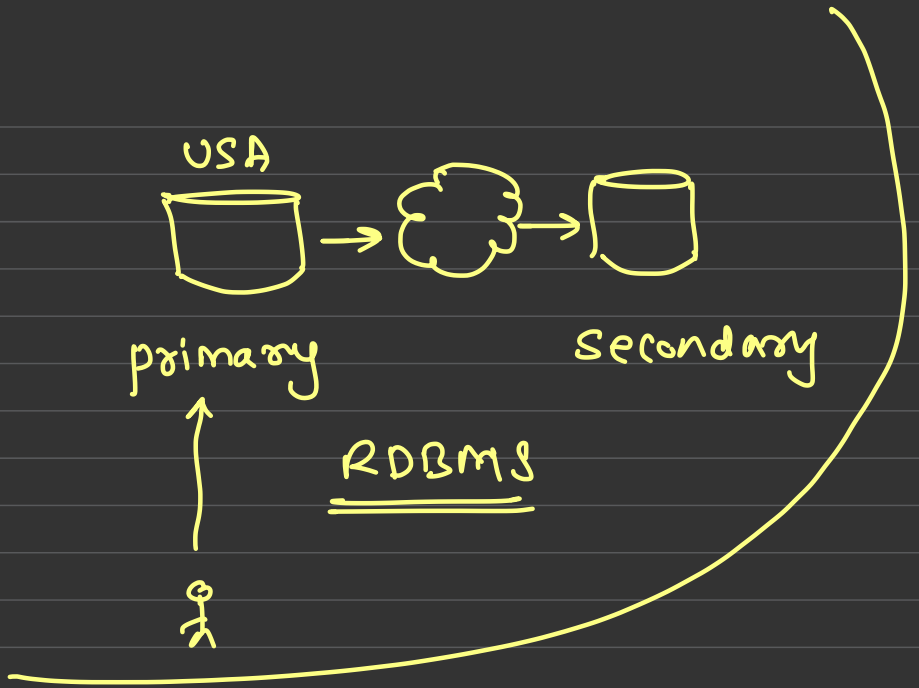
- A database is an organized collection of data, generally stored and accessed electronically from a computer system
- A database refers to a set of related data and the way it is organized
- Database Management System
  - Software that allows users to interact with one or more databases and provides access to all of the data contained in the database
- Types
  - RDBMS (SQL)
  - NoSQL

RDMBS

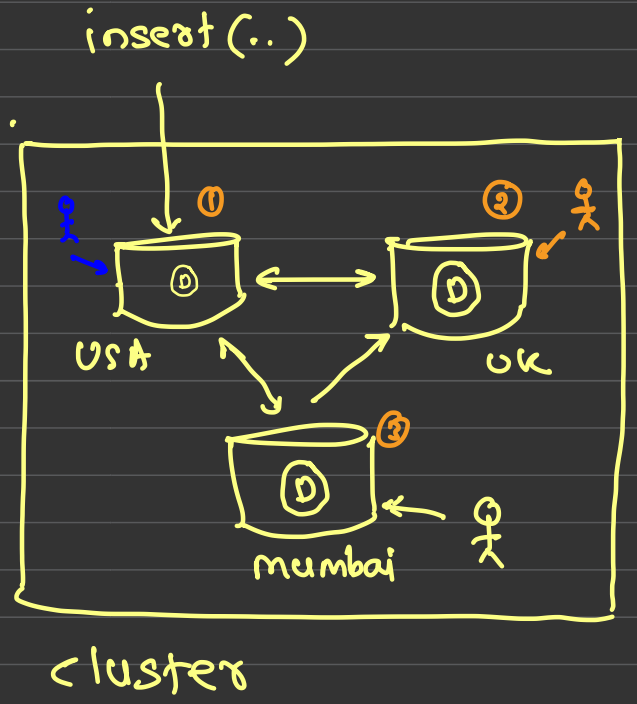
- The idea of RDBMS was borne in 1970 by E. F. Codd
- The language used to query RDBMS systems is SQL (Sequel Query Language)
- RDBMS systems are well suited for structured data held in columns and rows, which can be queried using SQL
- Supports: DML, DQL, DDL, DTL, DCL
- The RDBMS systems are based on the concept of ACID transactions
  - **Atomic**: implies either all changes of a transaction are applied completely or not applied at all
  - **Consistent**: data is in a consistent state after the transaction is applied
  - **Isolated**: transactions that are applied to the same set of data are independent of each other
  - **Durable**: the changes are permanent in the system and will not be lost in case of any failures

Scaling

- Scalability is the ability of a system to expand to meet your business needs
- E.g. scaling a web app is to allow more people to use your application
- Types
  - Vertical scaling
    - Add resources within the same logical unit to increase capacity
    - E.g. add CPUs to an existing server, increase memory in the system or expanding storage by adding hard drives
  - Horizontal scaling
    - Add more nodes to a system
    - E.g. adding a new computer to a distributed software application
    - Based on principle of distributed computing
- NoSQL databases are designed for Horizontal scaling
- So they are reliable, fault tolerant, better performance (at lower cost)



Distributed technology

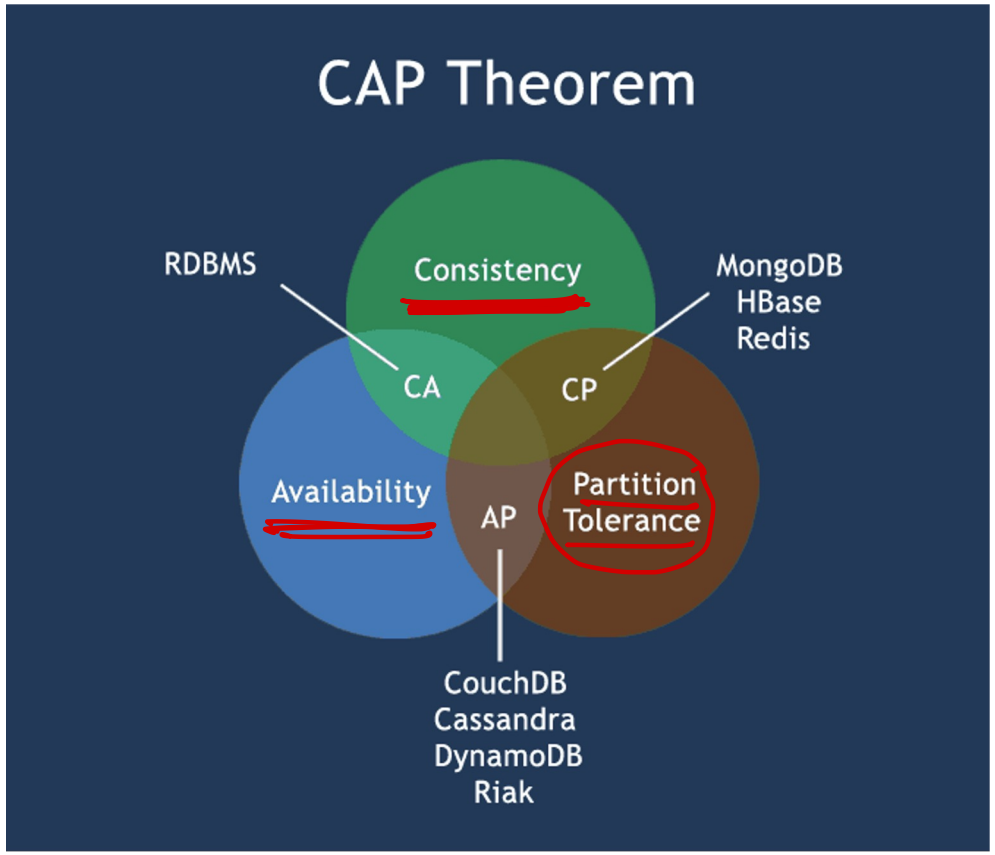


NoSQL

- Stands for Not Only SQL
- NoSQL is a term used to refer to non-relational databases
- The term NoSQL was coined by Carlo Strozzi in 1998 to name his lightweight Strozzi NoSQL open-source relational database
- Does not use any declarative query language (SQL)
- No predefined schema → loose schema
- Unstructured and unpredictable data
- Supports eventual consistency rather than ACID properties
- Prioritizes high performance, high availability and scalability
- In contrary to the ACID approach of traditional RDBMS systems, NoSQL solves the problem using an approach popularly called as BASE

CAP Theorem

- Also known as Brewer's theorem states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees
  - Consistency
    - Data is consistent after operation
    - After an update operation, all clients see the same data
  - Availability
    - System is always on (i.e. service guarantee), no downtime
  - Partition Tolerance
    - Wystem will continue to function even if it is partitioned into groups of servers that are not able to communicate with one another



BASE Theorm

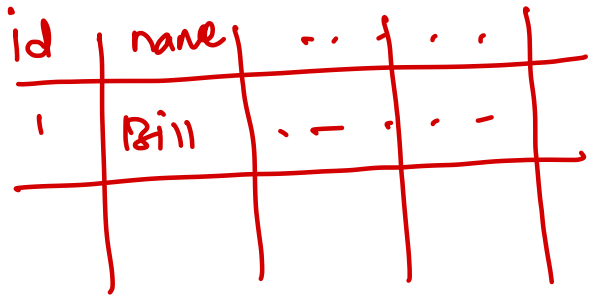
- Eric Brewer coined the BASE acronym
- BASE means
  - **Basically Available**: means the system will be available in terms of the CAP theorem.
  - **Soft state** indicates that even if no input is provided to the system, the state will change over time. This is in accordance to eventual consistency.
  - **Eventual consistency**: means the system will attain consistency in the long run, provided no input is sent to the system during that time.

NoSQL Types

- Following are the types of NoSQL database based on how it stores the data
  - Key Value
  - Document DB
  - Column Based DB
  - Graph DB

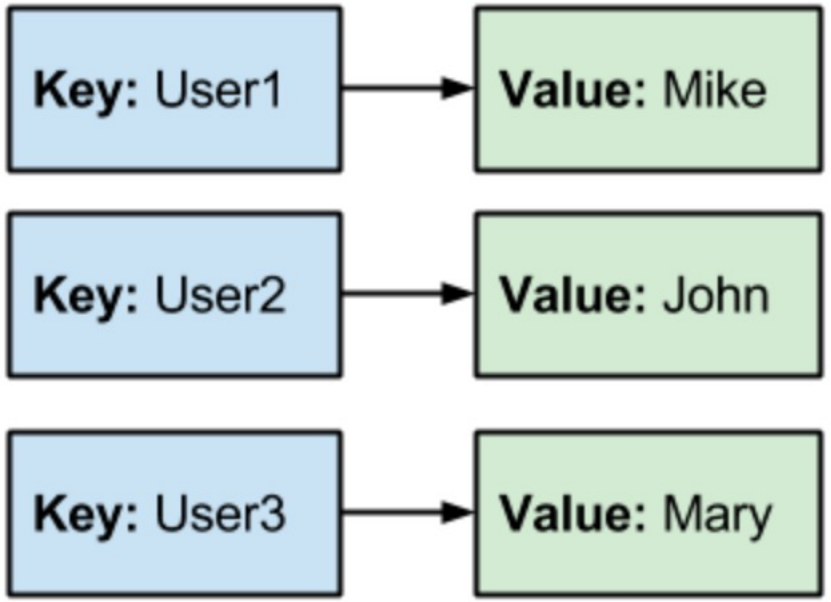
Key Value Database (cache)

- Data is stored as keys and values
- Provides super fast queries and ability to scale almost infinite data and performance level
- No relationships and weak/no schema
- E.g.
  - DynamoDB
  - Redis ***
  - Couchbase
  - ZooKeeper



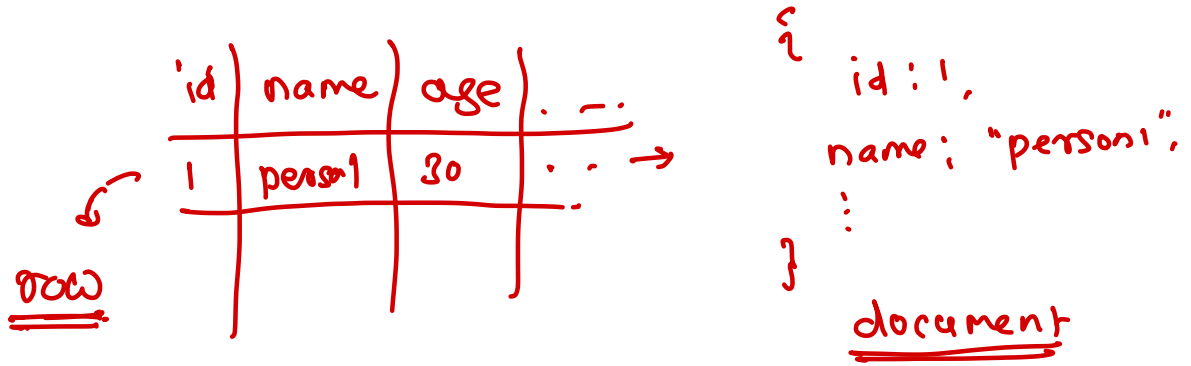
table

id : 1 name: Bill



Document DB

- Data is stored as Documents
- Document: collection of key-value pairs with structure
- Great performance when interacting with documents
- E.g.
  - MongoDB ***
  - CouchDB
  - RethinkDB



Document 1

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 2

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 3

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Column Based DB

- Data is stored as columns rather than rows
- Fast queries for datasets
- Slower when looking at individual records
- Great for warehousing and analytics
- E.g.
  - Redshift
  - Cassandra
  - HBase
  - Vertica

} Big Data

id	name	age
1	p1	10
2	p2	30

rows

⇒

id	1	2
name	p1	p2
age	10	30

Select id from person; ⇒

ID	Name
0001	Roffle
0002	Penny
0003	Winkie

Graph DB

- Designed for dynamic relationship
- Stores data as nodes and relationships between those nodes
- Ideal for human related data like social media
- Often as custom query language
- E.g.
  - Neo4J
  - Giraph
  - OrientDB

Advantages of NoSQL

- High scalability
  - This scaling up approach fails when the transaction rates and fast response requirements increase. In contrast to this, the new generation of NoSQL databases is designed to scale out (i.e. to expand horizontally using low-end commodity servers).
- Manageability and administration
  - NoSQL databases are designed to mostly work with automated repairs, distributed data, and simpler data models, leading to low manageability and administration.
- Low cost
  - NoSQL databases are typically designed to work with a cluster of cheap commodity servers, enabling the users to store and process more data at a low cost.
- Flexible data models
  - NoSQL databases have a very flexible data model, enabling them to work with any type of data; they don't comply with the rigid RDBMS data models. As a result, any application changes that involve updating the database schema can be easily implemented.

Disadvantages of NoSQL

- Maturity
  - Most NoSQL databases are pre-production versions with key features that are still to be implemented. Thus, when deciding on a NoSQL database, you should analyze the product properly to ensure the features are fully implemented and not still on the To-do list.
- Support
  - Support is one limitation that you need to consider. Most NoSQL databases are from start-ups which were open sourced. As a result, support is very minimal as compared to the enterprise software companies and may not have global reach or support resources.
- Limited Query Capabilities
  - Since NoSQL databases are generally developed to meet the scaling requirement of the web-scale applications, they provide limited querying capabilities. A simple querying requirement may involve significant programming expertise.
- Administration
  - Although NoSQL is designed to provide a no-admin solution, it still requires skill and effort for installing and maintaining the solution.
- Expertise
  - Since NoSQL is an evolving area, expertise on the technology is limited in the developer and administrator community.

NoSQL Scenarios

- When to use NoSQL ?
  - Large amount of data (TBs)
  - Many Read/Write operations
  - Economical scaling
  - Flexible Schema
- Examples
  - Social Media
  - Recordings
  - Geospatial analysis
  - Information processing

- When NOT to use NoSQL ?
  - Need ACID transactions
  - Fixed multiple relations
  - Need joins
  - Need high consistency
- Examples
  - Financial transactions
  - Business operations

SQL vs NoSQL

	SQL	NoSQL
Types	All types support SQL standard	Multiple types exists, such as document stores, key value stores, column databases, etc
History	Developed in 1970	Developed in 2000s
Examples	SQL Server, Oracle, MySQL	MongoDB, HBase, Cassandra
Data Storage Model	Data is stored in rows and columns in a table, where each column is of a specific type	The data model depends on the database type. It could be Key-value pairs, documents etc
Schemas	Fixed structure and schema	Dynamic schema. Structures can be accommodated
Scalability	Scale up approach is used	Scale out approach is used
Transactions	Supports ACID and transactions	Supports partitioning and availability
Consistency	Strong consistency	Dependent on the product [Eventual Consistency]
Support	High level of enterprise support	Open source model
Maturity	Have been around for a long time	Some of them are mature; others are evolving

MongoDB

MongoDB Overview

- Developed by 10gen in 2007
- Publicly available in 2009
- Open-source database which is controlled by 10gen
- Document oriented database → stores JSON documents
- Stores data in binary JSON — BSON
- Design Philosophy
  - MongoDB wasn't designed in a lab and is instead built from the experiences of building large scale, high availability, and robust systems

Features

- **Indexing**
  - MongoDB supports generic secondary indexes and provides unique, compound, geospatial, and full-text indexing capabilities as well
  - Secondary indexes on hierarchical structures such as nested documents and arrays are also supported and enable developers to take full advantage of the ability to model in ways that best suit their applications
- **Aggregation**
  - MongoDB provides an aggregation framework based on the concept of data processing pipelines
  - Aggregation pipelines allow you to build complex analytics engines by processing data through a series of relatively simple stages on the server side, taking full advantage of database optimizations
- **Special collection and index types**
  - MongoDB supports time-to-live (TTL) collections for data that should expire at a certain time, such as sessions and fixed-size (capped) collections, for holding recent data, such as logs
- **File storage**
  - MongoDB supports an easy-to-use protocol for storing large files and file metadata

MongoDB Ecosystem

MongoDB Database

Self – Managed / Enterprise

Atlas (Cloud)

Mobile

CloudManager / OpsManager

Compass

GUI

BI Connectors

MongoDB Charts

Stitch

Serverless Query API

Serverless Functions

Database Triggers

Real-Time Sync

Install MongoDB

- Install MongoDB by downloading community edition (https://www.mongodb.com/download-center/community)
- Linux and Mac Users
  - Extract the downloaded file somewhere in the disk
  - Set the environment path to use the tools without going to the bin directory

    > export PATH=<path>/bin:$PATH
  - You can also include the above line in the ~/.bash_profile or ~/.basrc file
- Windows Users
  - Install the MongoDB by following all the steps in the installation wizard
  - Set the the environment path to include the <path>/bin

- Create a directory somewhere in the disk to store the data [generally on Linux or Mac, create a directory named data on the root (/): /data/]

What have you downloaded?

- **mongo**: The Command Line Interface to interact with the db ⇒ mongosh
- **mongod**
  - This is the database server
  - It is written in C, C++ and JS
- **mongodump**: It dumps out the Binary of the Database(BSON)
- **mongoexport**: Exports the document to Json, CSV format
- **mongoimport**: To import some data into the DB
- **mongorestore**: to restore anything that you've exported
- **mongostat**: Statistics of databases

JSON

- Defined as part of the JavaScript language in the early 2000s by JavaScript creator Douglas Crockford
- It wasn't until 2013 that the format was officially specified
- JavaScript objects are simple associative containers, wherein a string key is mapped to a value
- JSON shows up in many different cases:
  - APIs
  - Configuration files
  - Log messages
  - Database storage
- However, there are several issues that make JSON less than ideal for usage inside of a database
  - JSON is a text-based format, and text parsing is very slow
  - JSON's readable format is far from space-efficient, another database concern
  - JSON only supports a limited number of basic data types

JSON


```
{
        "_id": 1,
         "name" : { "first" : "John", "last" : "Backus" },
        "contribs" : [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],
        "awards" : [
                { "award" : "W.W. McDowell Award", "year" : 1967, "by" : "IEEE Computer Society" },
                { "award" : "Draper Prize", "year" : 1993, "by" : "National Academy of Engineering" }
        ]
}
```

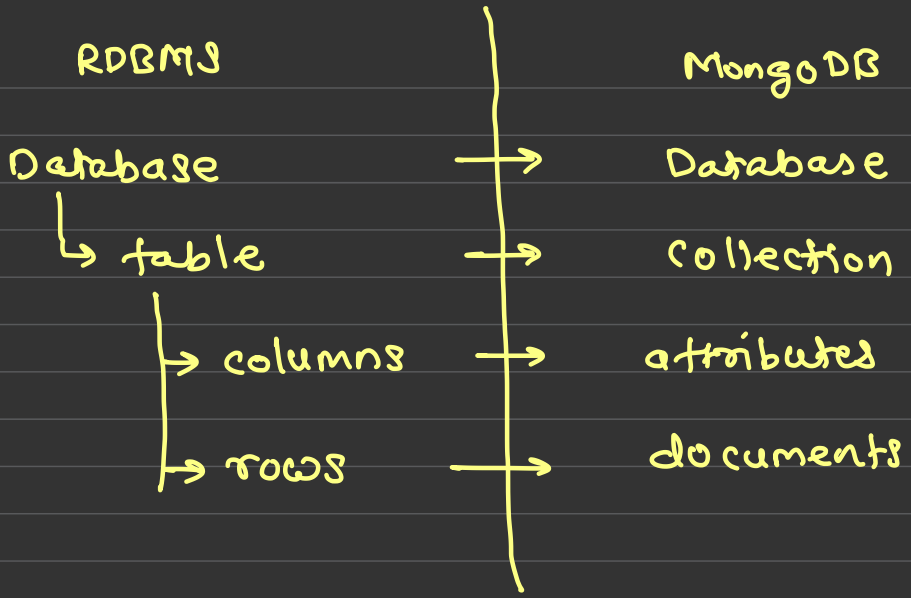
BSON

- BSON simply stands for “Binary JSON”
- Binary structure encodes type and length information, which allows it to be parsed much more quickly
- It has been extended to add some optional non-JSON-native data types
- It allows for comparisons and calculations to happen directly on data
- MongoDB stores data in BSON format both internally, and over the network
- Anything you can represent in JSON can be natively stored in MongoDB

	JSON	BSON
Encoding	UTF-8 String	Binary
Data Support	String, Boolean, Number, Array	String, Boolean, Number (Integer, Float, Long, Decimal128...), Array, Date, Raw Binary
Readability	Human and Machine	Machine Only

Data Types

- **String** – This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid
- **Integer** – This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server
- **Boolean** – This type is used to store a boolean (true/ false) value
- **Double** – This type is used to store floating point values
- **Min/ Max keys** – This type is used to compare a value against the lowest and highest BSON elements
- **Arrays** – This type is used to store arrays or list or multiple values into one key
- **Timestamp** – ctimestamp. This can be handy for recording when a document has been modified or added
- **Object** – This datatype is used for embedded documents
- **Null** – This type is used to store a Null value
- **Symbol** – it's generally reserved for languages that use a specific symbol type
- **Date** – This datatype is used to store the current date or time in UNIX time format
- **Object ID** – This datatype is used to store the document's ID
- **Binary data** – This datatype is used to store binary data
- **Code** – This datatype is used to store JavaScript code into the document


RDBMS
MongoDB
Database
Database
table
collection
columns
attributes
rows
documents

MongoDB Terminology

- **Database**
  - This is a container for collections like in RDMS wherein it is a container for tables
  - Each database gets its own set of files on the file system
  - A MongoDB server can store multiple databases
- **Collection**
  - This is a grouping of MongoDB documents
  - A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL
  - Collections don't enforce any sort of structure
- **Document**
  - A record in a MongoDB collection is basically called a document
  - The document, in turn, will consist of field name and values
- **Field**
  - Field names are strings
  - A name-value pair in a document
  - A document has zero or more fields
  - Fields are analogous to columns in relational databases

Document

- MongoDB stores data records as BSON documents
- Maximum size of document is 16MB
- Restrictions
  - The field name _id is reserved for use as a primary key
  - Field names **cannot** contain the null character
  - Top-level field names **cannot** start with the dollar sign ($) character

_id field

- Each document requires a unique _id field that acts as a primary key
- If an inserted document omits the _id field, the MongoDB driver automatically generates an ObjectId for the _id field
- Behaviors
  - By default, MongoDB creates a unique index on the _id field during the creation of a collection
  - The _id field is always the first field in the documents. If the server receives a document that does not have the _id field first, then the server will move the field to the beginning.
  - The _id field may contain values of any BSON data type, other than an array
- Autogenerated _id (of type ObjectId) will be of 12 bytes which contains
  - Timestamp: 4 bytes
  - Machine Id: 3 bytes
  - Process Id: 2 bytes
  - Counter: 3 bytes

CRUD operations

Database Operations

- List existing databases
  > show dbs
  > show databases

- Create and use database
  > use <db name>

- Get the selected database name
  > db

- Show the database statistics
  > db.stats()

Collection operations

- Get the list of collections

  > show collections

- Create Collection

  > db.createCollection('contacts')

- Drop Collection

  > db.contacts.drop()

Create Document (Insert data)

- Create one document

  > db.contacts.insert({ name: ‘amit’, mobile: ‘7709859986’ })

- Create many documents

```
> db.contacts.insertMany([
      { name: ‘contact 1’, address: ‘pune’ },
      { name: ‘contact 2’, address: ‘mumbai’ }
])
```

- **Note: if you are passing the _id field, make sure that it is unique. If it is not unique, the document will not get inserted**

Read/Find Documents (Query data)

- Find documents

  > db.contacts.find()

- Returns cursor on which following operations allowed
  - **hasNext():** returns if cursor can iterate further
  - **next():** returns the next document
  - **skip(n):** skips first n documents
  - **limit(n):** limit the result to n
  - **count():** returns the count of result
  - **toArray():** returns an array of document
  - **forEach(fn):** Iterates the cursor to apply a JavaScript function to each document from the cursor
  - **pretty():** Configures the cursor to display results in an easy-to-read format
  - **sort():** sorts documents
- Shell by default returns 20 records. Press ”it” for more results

Filtering results

> db.contacts.find({ name: ‘amit’ }) : select * from contacts where name = ‘amit’;

> db.contacts.find({ name: /amit/ }) : select * from contacts where name like ‘%amit%’;

- Relational operators
  - $eq, $ne, $gt, $lt, $gte, $lte, $in, $nin
- Logical opeators
  - $and, $or, $nor, $not
- Element operators
  - $exists, $type
- Evaluation operators
  - $regex, $where, $mod
- Array operators
  - $size, $elemMatch, $all, $slice

Query Projection

- Projection: selecting required fields while finding the documents
- E.g.
  - db.contacts.find({}, {name: 1})
- Required fields can be added the projection list with value 1
- Non-required fields can be added the projection list with value 0
- Can not mix both required and non-required in the projection

Update Document

- Syntax
  - db.<collection>.update(criteria, newObject)
- E.g.
  - > db.contacts.update({ name: ‘amit’ }, { $set: { address: ’Pune’} })
- Update operators
  - $set, $inc, $push, $each, $pull
- In place updates are faster (e.g. $inc) than setting new object

Delete document

> db.contacts.remove(criteria)

> db.contacts.deleteOne(criteria);

> db.contacts.deleteMany(criteria);

> db.contacts.deleteMany({}); → delete all docs, but not collection

> db.contacts.drop(); → delete all docs & collection as well : efficient

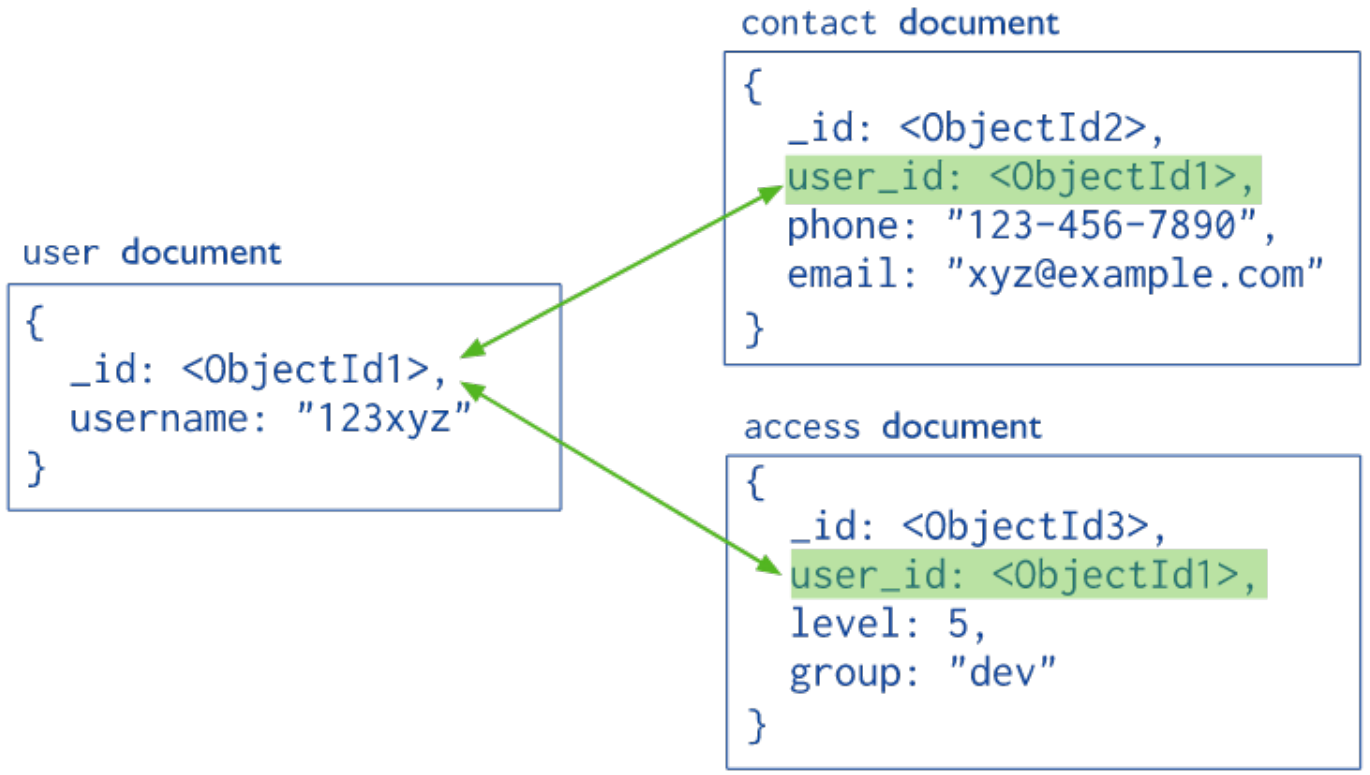
Data Modeling

Embedded data model

- Suitable for one-to-one or one-to-many relationship
- Faster read operation
- Related data fetch in single db operation
- Atomic update of document
- Document growth reduce write performance and may lead to fragmentation

Normalized data model

- Use normalized data model
  - to represent more complex many-to-many relationships
  - to model large hierarchical data sets
- Reduce data duplication



Aggregation Pipeline

Overview

- Aggregation operations process data records and return computed results
- Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result
- Documents enter a multi-stage pipeline that transforms the documents into aggregated results
- Pipeline
  - The MongoDB aggregation pipeline consists of stages
  - Each stage transforms the documents as they pass through the pipeline
  - Pipeline stages do not need to produce one output document for every input document
    - e.g., some stages may generate new documents or filter out documents

Operators

- $project: select columns (existing or computed)
- $match: where clause (criteria)
- $group: group by
  - { $group: { _id: <expr>, <field1>: { <accum1> : <expr1> }, ... } }
  - The possible accumulators are: $sum, $avg, etc.
- $unwind: extract array elements from array field
- $lookup: left outer join
- $out: put result of pipeline in another collection (last operation)

Optimization

Indexes

- Indexes support the efficient execution of queries in MongoDB
- Without indexes, MongoDB must perform a *collection scan*, i.e. scan every document in a collection, to select those documents that match the query statement
- If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect
- MongoDB defines indexes at the collection level and supports indexes on any field or sub-field of the documents in a MongoDB collection
- Create Index

  > db.collection.createIndex(<key and index type specification>, <options>)
- Drop Index

  > db.collection.dropIndex(<key and index type>)
- Get Indexes

  > db.collection.get_indexes()
- The default name for an index is the concatenation of the indexed keys and each key's direction in the index (i.e. 1 or -1) using underscores as a separator
  - For example, an index created on { item : 1, quantity: -1 } has the name item_1_quantity_-1

Indexes - Types

- Regular index (Single Key)
- Composite index
- Unique index
- TTL index
- Geospatial indexes

Capped Collections

- Capped collections are fixed sized collections for high-throughput insert and retrieve operations
- They maintain the order of insertion without any indexing overhead
- The oldest documents are auto-removed to make a room for new records. The size of collection should be specified while creation
- The update operations should be done with index for better performance. If update operation change size, then operation fails
- Cannot delete records from capped collections. Can drop collection
- E.g.

  > db.createCollection("logs", { capped: true, size: 4096 });

  if size is below 4096, 4096 is considered. Higher sizes are roundup by 256
- Benefits
  - Incredible read/write speed (>10000 operations per seconds)
  - Availability of cursor