

Introduction to NoSQL and MongoDB

MongoDB Webinar

@ Dr. Ambedkar Institute of Technology

24th Nov 2020

APARNA R

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RAMAIAH INSTITUTE OF TECHNOLOGY, BANGALORE

Outline for today

- NoSQL
 - Introduction
 - Strengths and weaknesses of NoSQL
- MongoDB
 - ✓ Installation
 - ✓ Functionality
 - ✓ Examples

NoSQL - Introduction

A NoSQL (originally "non-SQL" or "non-relational") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

Non-SQL databases have existed since the late 1960s, but the name "NoSQL" was only coined in the early 21st century triggered by the needs of Web 2.0 companies.

NoSQL databases are being extensively employed in big data and real-time web applications.

Reference: <https://en.wikipedia.org/wiki/NoSQL>

NoSQL - Introduction

NoSQL systems are often called "Not only SQL" to emphasize that they may support SQL-like query languages or sit alongside SQL databases in polyglot-persistent architectures.

(**Polyglot persistence** is the concept of using different data storage technologies to handle different data storage needs within a given enterprise and even software application.)

Taxonomy of NoSQL

classification based on data model

- **Key-value**



- **Graph database**



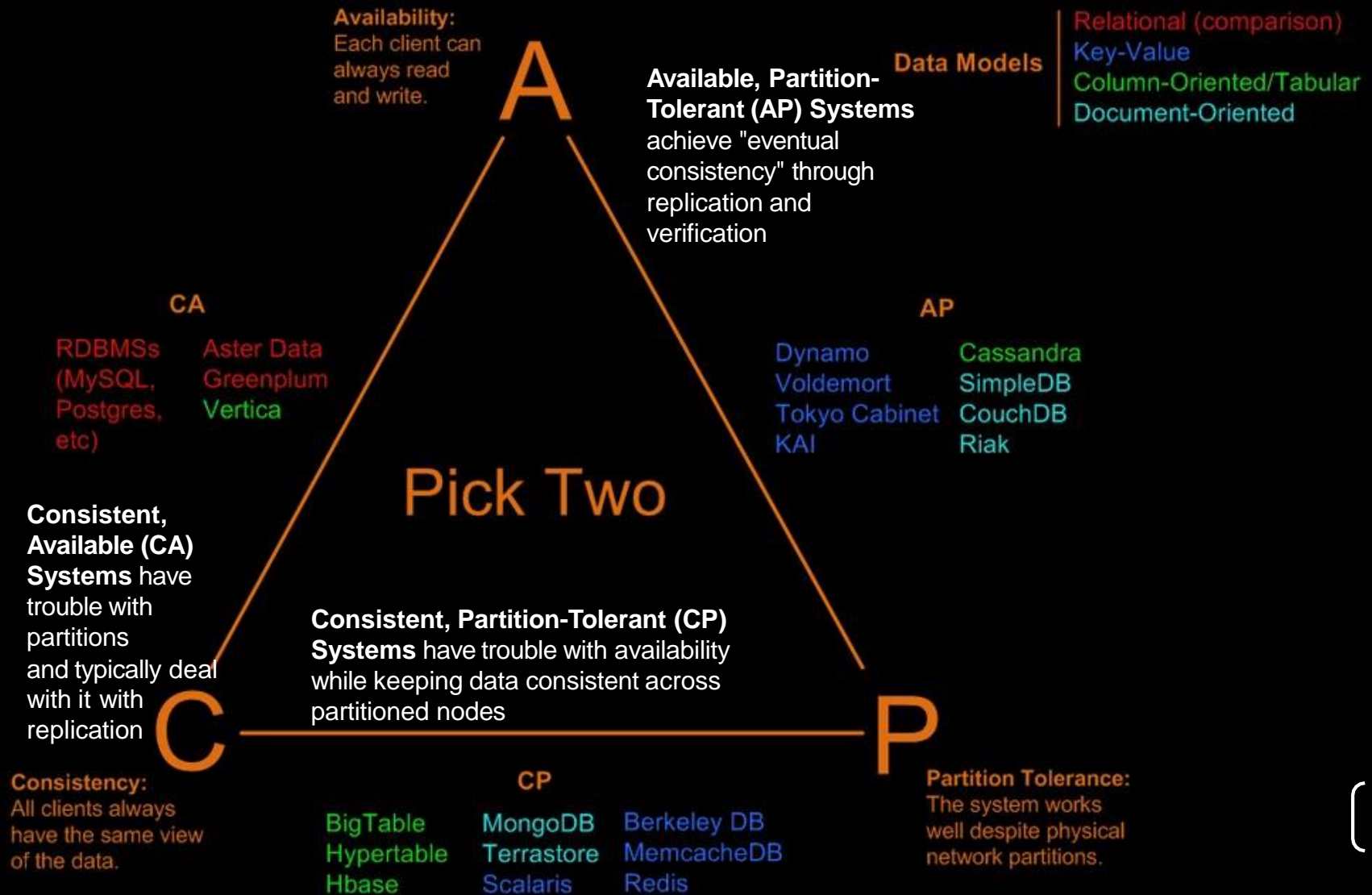
- **Document-oriented**



- **Column family**



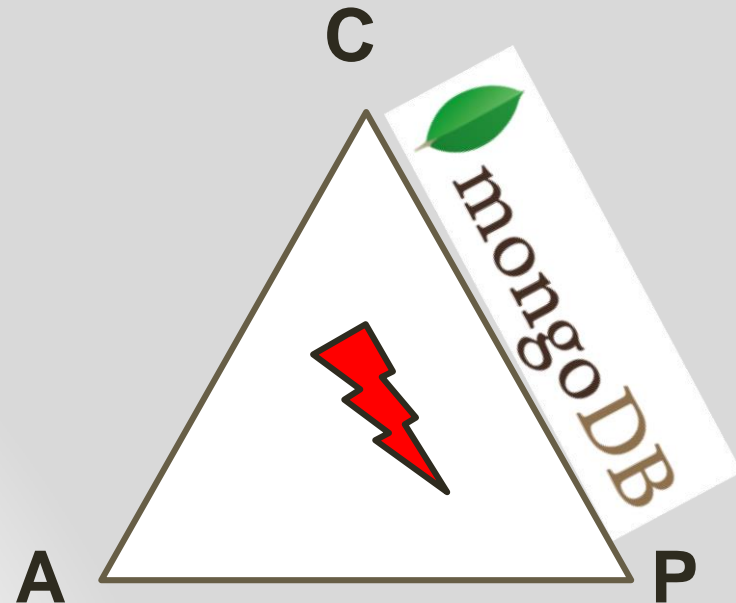
Visual Guide to NoSQL Systems



MongoDB: CAP approach

Focus on Consistency and Partition tolerance

- **Consistency**
 - all replicas contain the same version of the data
- **Availability**
 - system remains operational on failing nodes
- **Partition tolerance**
 - multiple entry points
 - system remains operational on system split



CAP Theorem:
satisfying all three at the same time is
impossible

Benefits of NoSQL

Elastic Scaling

- RDBMS scale up – bigger load, bigger server
- NO SQL scale out – distribute data across multiple hosts seamlessly

DBA Specialists

- RDMS require highly trained expert to monitor DB
- NoSQL require less management, automatic repair and simpler data models

Big Data

- Huge increase in data RDMS: capacity and constraints of data volumes at its limits
- NoSQL designed for big data

Benefits of NoSQL

Flexible data models

- Change management to schema for RDMS have to be carefully managed
- NoSQL databases more relaxed in structure of data
 - Database schema changes do not have to be managed as one complicated change unit
 - Application already written to address an amorphous schema

Economics

- RDMS rely on expensive proprietary servers to manage data
- No SQL: clusters of cheap commodity servers to manage the data and transaction volumes
- Cost per gigabyte or transaction/second for NoSQL can be lower than the cost for a RDBMS

Drawbacks of NoSQL

- Support
 - RDBMS vendors provide a high level of support to clients
 - Stellar reputation
 - NoSQL – are open source projects with startups supporting them
 - Reputation not yet established

Drawbacks of NoSQL

- **Administration**

- RDMS administrator well defined role
- No SQL's goal: no administrator necessary
however NO SQL still requires effort to maintain

- **Lack of Expertise**

- Whole workforce of trained and seasoned RDMS developers
- Still recruiting developers to the NoSQL camp

- **Analytics and Business Intelligence**

- **RDMS designed to address this niche**

- NoSQL designed to meet the needs of an Web 2.0 application - not designed for ad hoc query of the data
 - Tools are being developed to address this need

How does NoSQL vary from RDBMS?

- Looser schema definition
- Applications written to deal with specific documents/ data
 - Applications aware of the schema definition as opposed to the data
- Designed to handle distributed, large databases
- Trade offs:
 - No strong support for ad hoc queries but designed for speed and growth of database
 - Query language through the API
 - Relaxation of the ACID properties

RDB ACID to NoSQLBASE

Atomicity

Consistency

Isolation

Durability



Basically

Available (CP)

Soft-state
(State of system may change over time)

Eventually
consistent

(Asynchronous propagation)



What is MongoDB?

- Developed by 10gen --- Founded in 2007
- A document-oriented, NoSQL database
 - Hash-based, *schema-less database*
 - No Data Definition Language
 - In practice, this means you can store hashes with any keys and values that you choose
 - Keys are a basic data type but in reality stored as strings
 - Document Identifiers (`_id`) will be created for each document, field name reserved by system
 - Application tracks the schema and mapping
 - Uses BSON format (Based on JSON – B stands for Binary)
- Written in C++
- Supports APIs (drivers) in many computer languages
 - JavaScript, Python, Ruby, Perl, Java, Java Scala, C#, C++, Haskell, Erlang

MongoDB...

MongoDB is a cross-platform, document oriented database written in C++ that provides

- High performance.
- High availability.
- Easy scalability.

MongoDB works on concept of collection and document.

Functionality of MongoDB

- Dynamic schema
 - No DDL
- Document-based database
- Secondary indexes
- Query language via an API
- Atomic writes and fully-consistent reads
 - If system configured that way
- Master-slave replication with automated failover (replica sets)
- Built-in horizontal scaling via automated range-based partitioning of data (sharding)
- No joins nor transactions

Why use MongoDB?

- Simple queries
- Functionality provided applicable to most web applications
- Easy and fast integration of data
 - No ERD diagram

BUT

- Not well suited for heavy and complex transactions systems

Installation – Different ways

1. If you want to install MongoDB server and client and use it

MongoDB is available in two server editions :
Community and Enterprise editions

Download from here

<https://docs.mongodb.com/manual/installation/>

Lists the installation steps for various Operating system platforms

2. Use readily available online MongoDB tutorials

<https://www.jdoodle.com/online-mongodb-terminal/>

<https://www.humongous.io/app/>

To name a few...

Installation – Different ways

3. Use a cloud MongoDB service – MongoDB Atlas

Built for agile teams who'd rather spend time building apps than managing databases.

Available on AWS, Azure, and GCP.

<https://www.mongodb.com/cloud/atlas>

Local

4 DBS 2 COLLECTIONS

★ FAVORITE





HOST
127.0.0.1:27017CLUSTER
StandaloneEDITION
MongoDB 4.2.8 Community

Filter your data

- > admin
- > config
- > local
- > test1

Databases

CREATE DATABASE

| Database Name ^ | Storage Size | Collections | Indexes | |
|-----------------|--------------|-------------|---------|---|
| admin | 20.0KB | 0 | 1 |  |
| config | 24.0KB | 0 | 2 |  |
| local | 20.0KB | 1 | 1 |  |
| test1 | 36.0KB | 1 | 1 |  |

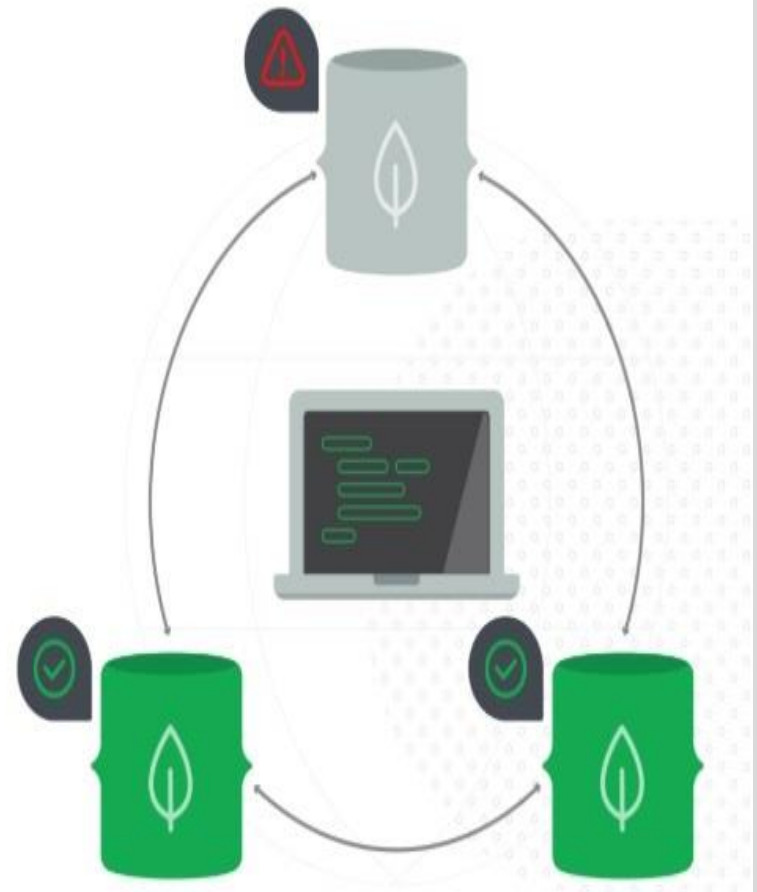
MongoDB Atlas

Cloud-hosted MongoDB service on AWS, Azure and Google Cloud. Deploy, operate, and scale a MongoDB database in just a few clicks



Always on and durable

With MongoDB Atlas, your self-healing clusters are made up of geographically distributed database instances to ensure no single point of failure. Want even better availability guarantees? Get multi-region fault tolerance by enabling cross-region replication. MongoDB Atlas also includes powerful features to enhance reliability for your mission-critical production databases, such as continuous backups and point-in-time recovery.



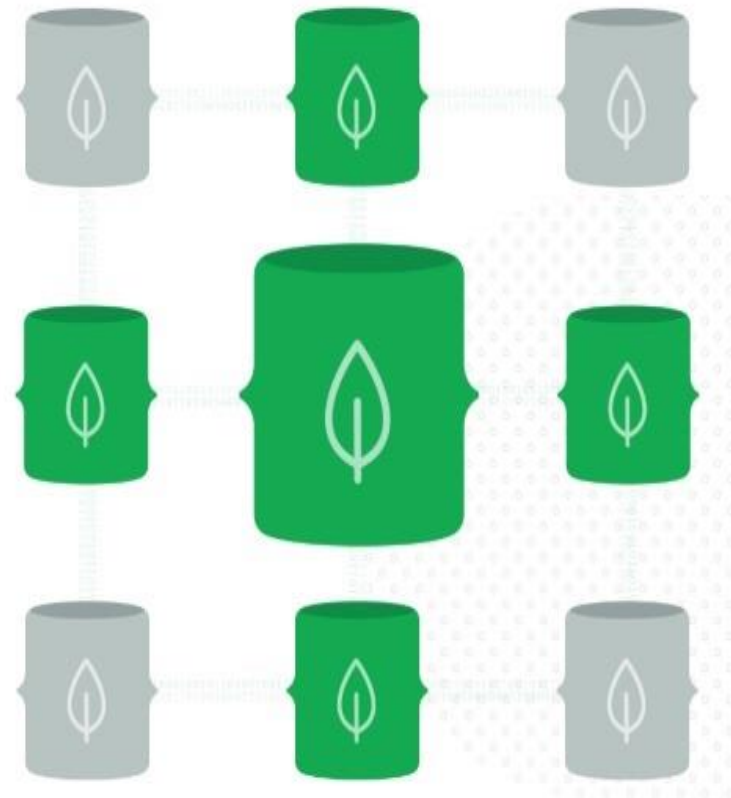


Secure from the start

MongoDB Atlas makes it easy to control access to your database. Your database instances are deployed in a unique Virtual Private Cloud (VPC) to ensure network isolation. Other security features include IP whitelisting or VPC Peering, always-on authentication, encryption at rest and encryption in transit, sophisticated role-based access management, and more.

Fully automated and elastic

MongoDB Atlas automates infrastructure provisioning, setup, and deployment so your teams can get the database resources they need, when they need them. Patches and minor version upgrades are applied automatically. And when you need to modify your cluster — whether it's to scale out or perform an upgrade — MongoDB Atlas lets you do so in a few clicks with no downtime window required.



MongoDB: Hierarchical Objects

- A MongoDB instance may have zero or more 'databases'
- A database may have zero or more 'collections'.
- A collection may have zero or more 'documents'.
- A document may have one or more 'fields'.
- MongoDB 'Indexes' function much like their RDBMS counterparts.

RDB Concepts to NO SQL

| RDBMS | | MongoDB |
|-------------|---|-------------------|
| Database | ➡ | Database |
| Table, View | ➡ | Collection |
| Row | ➡ | Document (BSON) |
| Column | ➡ | Field |
| Index | ➡ | Index |
| Join | ➡ | Embedded Document |
| Foreign Key | ➡ | Reference |
| Partition | ➡ | Shard |

Collection is not strict about what it stores

Schema-less

Hierarchy is evident in the design

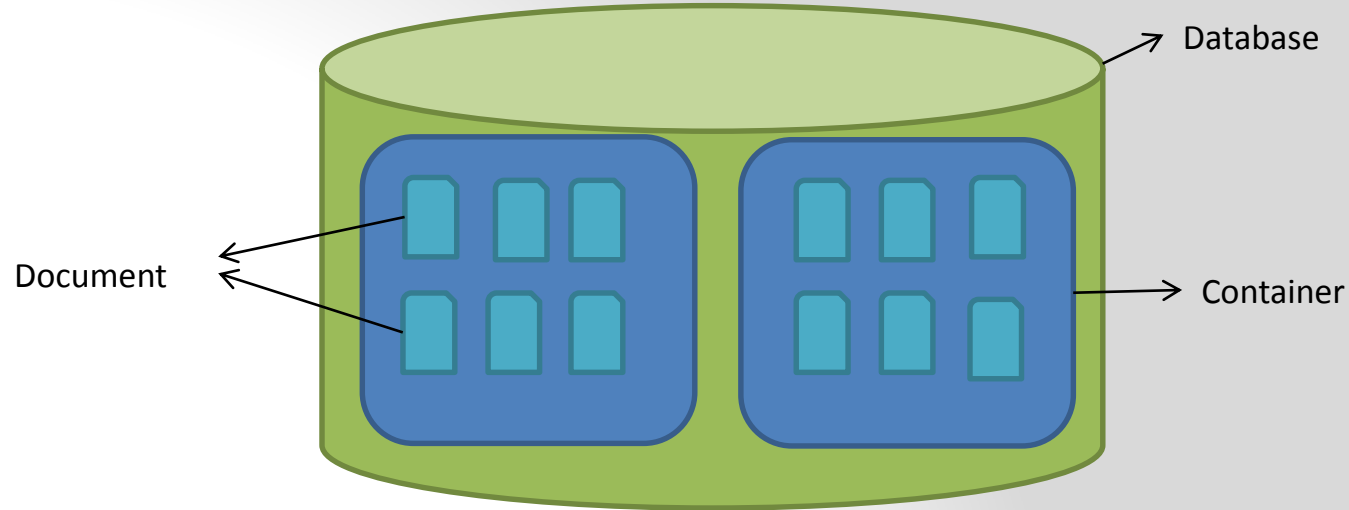
Embedded Document

MongoDB Processes and configuration

- Mongod – Database instance
- Mongos - Sharding processes
 - Analogous to a database router.
 - Processes all requests
 - Decides how many and which *mongods* should receive the query
 - *Mongos* collates the results, and sends it back to the client.
- Mongo – an interactive shell (a client)
 - Fully functional JavaScript environment for use with a MongoDB
- You can have one *mongos* for the whole system no matter how many *mongods* you have OR you can have one local *mongos* for every client if you wanted to minimize network latency.

MongoDB architecture

Architecture : -



BSON format

- Binary-encoded serialization of JSON-like documents
- Zero or more key/value pairs are stored as a single entity
- Each entry consists of a field name, a data type, and a value
- Large elements in a BSON document are prefixed with a length field to facilitate scanning

Schema Free

- MongoDB does not need any pre-defined data schema
- Every document in a collection could have different data
 - Addresses NULL data fields

```
{name: "will",  
  eyes: "blue",  
  birthplace: "NY",  
  aliases: ["bill", "la ciacco"],  
  loc: [32.7, 63.4],  
  boss: "ben"}
```

```
{  
  name: "jeff",  
  eyes: "blue",  
  loc: [40.7, 73.4],  
  boss: "ben"}
```

```
{  
  name: "ben",  
  hat: "yes"}
```

```
{name: "brendan",  
  aliases: ["el diablo"]}
```

```
{name: "matt",  
  pizza: "DiGiorno",  
  height: 72,  
  loc: [44.6, 71.3]}
```

JSON format

- Data is in name / value pairs
- A name/value pair consists of a field name followed by a colon, followed by a value:
 - Example: “name”: “R2-D2”
- Data is separated by commas
 - Example: “name”: “R2-D2”, race : “Droid”
- Curly braces hold objects
 - Example: {“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”}
- An array is stored in brackets []
 - Example[{“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”}, {“name”: “Yoda”, affiliation: “rebels”}]

Document(JSON) structure

The document has simple structure and very easy to understand the content

JSON is smaller, faster and lightweight compared to XML.

For data delivery between servers and browsers, JSON is a better choice

Easy in parsing, processing, validating in all languages

JSON can be mapped more easily into object oriented system.

```
[  
{  
  "Name": "Tom",  
  "Age": 30,  
  "Role": "Student",  
  "University": "CU",  
}  
{  
  "Name": "Sam",  
  "Age": 32,  
  "Role": "Student",  
  "University": "OU",  
}  
]
```

Difference Between XML And JSON

| XML | JSON |
|--|--|
| It is a markup language. | It is a way of representing objects. |
| This is more verbose than JSON. | This format uses less words. |
| It is used to describe the structured data. | It is used to describe unstructured data which include arrays. |
| JavaScript functions like <i>eval()</i> , <i>parse()</i> doesn't work here. | When <i>eval</i> method is applied to JSON it returns the described object. |
| <p>Example:</p> <pre><car> <company>Volkswagen</company> <name>Vento</name> <price>800000</price> </car></pre> | <pre>{ "company": Volkswagen, "name": "Vento", "price": 800000 }</pre> |

Why JSON?

JSON is faster and easier than XML when you are using it in AJAX web applications:

Steps involved in exchanging data from web server to browser involves:

Using XML

1. Fetch an XML document from web server.
2. Use the XML DOM to loop through the document.
3. Extract values and store in variables.
4. It also involves type conversions.

Using JSON

1. Fetch a JSON string.
2. Parse the JSON string using `eval()` or `parse()` JavaScript functions.

MongoDB Features

- Document-Oriented storage
- Full Index Support
- Replication & High Availability
- Auto-Sharding
- Querying
- Fast In-Place Updates
- Map/Reduce functionality

Agile

Scalable

Index Functionality

- B+ tree indexes
- An index is automatically created on the `_id` field (the primary key)
- Users can create other indexes to improve query performance or to enforce Unique values for a particular field
 - Supports single field index as well as Compound index
- Like SQL order of the fields in a compound index matters
 - If you index a field that holds an array value, MongoDB creates separate index entries for *every* element of the array
- Sparse property of an index ensures that the index only contain entries for documents that have the indexed field. (so ignore records that do not have the field defined)
- If an index is both unique and sparse – then the system will reject records that have a duplicate key value but allow records that do not have the indexed field defined

CRUD operations

- Create
 - `db.collection.insert(<document>)`
 - `db.collection.save(<document>)`
 - `db.collection.update(<query>, <update>, { upsert: true })`
- Read
 - `db.collection.find(<query>, <projection>)`
 - `db.collection.findOne(<query>, <projection>)`
- Update
 - `db.collection.update(<query>, <update>, <options>)`
- Delete
 - `db.collection.remove(<query>, <justOne>)`

Collection specifies the collection or the 'table' to store the document

Create Operations

`db.collection` specifies the collection or the 'table' to store the document

- `db.collection_name.insert(<document>)`
 - Omit the `_id` field to have MongoDB generate a unique key
 - Example `db.parts.insert({type: "screwdriver", quantity: 15 })`
 - `db.parts.insert({_id: 10, type: "hammer", quantity: 1 })`
- `db.collection_name.update(<query>, <update>, { upsert: true })`
 - Will update 1 or more records in a collection satisfying query
- `db.collection_name.save(<document>)`
 - Updates an existing record or creates a new record

Read Operations

- `db.collection.find(<query>, <projection>).cursor` modified
 - Provides functionality similar to the `SELECT` command
 - `<query>` where condition, `<projection>` fields in result set
 - Example: `var PartsCursor = db.parts.find({parts: "hammer"}).limit(5)`
 - Has cursors to handle a result set
 - Can modify the query to impose limits, skips, and sort orders.
 - Can specify to return the 'top' number of records from the result set
- `db.collection.findOne(<query>, <projection>)`

Query Operators

| Name | Description |
|-------------|---|
| \$eq | Matches value that are equal to a specified value |
| \$gt, \$gte | Matches values that are greater than (or equal to a specified value |
| \$lt, \$lte | Matches values less than or (equal to) a specified value |
| \$ne | Matches values that are not equal to a specified value |
| \$in | Matches any of the values specified in an array |
| \$nin | Matches none of the values specified in an array |
| \$or | Joins query clauses with a logical OR returns all |
| \$and | Join query clauses with a logical AND |
| \$not | Inverts the effect of a query expression |
| \$nor | Join query clauses with a logical NOR |
| \$exists | Matches documents that have a specified field |

Update Operations

- `db.collection_name.insert(<document>)`
 - Omit the `_id` field to have MongoDB generate a unique key
 - Example `db.parts.insert({type: "screwdriver", quantity: 15 })`
 - `db.parts.insert({_id: 10, type: "hammer", quantity: 1 })`
- `db.collection_name.save(<document>)`
 - Updates an existing record or creates a new record
- `db.collection_name.update(<query>, <update>, { upsert: true })`
 - Will update 1 or more records in a collection satisfying query
- `db.collection_name.findAndModify(<query>, <sort>, <update>,<new>, <fields>,<upsert>)`
 - Modify existing record(s) – retrieve old or new version of the record

Delete Operations

- `db.collection_name.remove(<query>, <justone>)`
 - Delete all records from a collection or matching a criterion
 - `<justone>` - specifies to delete only 1 record matching the criterion
 - Example: `db.parts.remove(type: /^h/ }`) - remove all parts starting with h
 - `Db.parts.remove()` – delete all documents in the parts collections

CRUD examples

```
> db.user.insert({  
  first: "John",  
  last : "Doe",  
  age: 39  
})
```

```
> db.user.find (  
  { "_id" : ObjectId("51"),  
    "first" : "John",  
    "last" : "Doe",  
    "age" : 39  
  }  
)
```

```
> db.user.update(  
  { "_id" : ObjectId("51") },  
  {  
    $set: {  
      age: 40,  
      salary: 7000  
    }  
  }  
)
```

```
> db.user.remove(  
  "first": /^J/  
)
```

The insert() Method

To insert data into MongoDB collection, you need to use MongoDB's **insert()** or **save()** method .

The basic syntax of **insert()** command is as follows –

“db.COLLECTION_NAME.insert(document)”

Example: -

```
db.StudentRecord.insert (
{
  "Name": "Tom",
  "Age": 30,
  "Role": "Student",
  "University": "CU",
},
{
  "Name": "Sam",
  "Age": 22,
  "Role": "Student",
  "University": "OU",
}
)
```

The find() Method

To query data from MongoDB collection, you need to use MongoDB's **find()** method.

The basic syntax of **find()** method is as follows –

“db.COLLECTION_NAME.find()”

find() method will display all the documents in a non-structured way.

To display the results in a formatted way, you can use **pretty()** method.

“db.mycol.find().pretty() “

Example: -

```
db.StudentRecord.find().pretty()
```

The remove() Method

MongoDB's **remove()** method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

deletion criteria – (Optional) deletion criteria according to documents will be removed.

justOne – (Optional) if set to true or 1, then remove only one document.

Syntax

db.COLLECTION_NAME.remove(DELETION_CRITERIA)

Remove based on DELETION_CRITERIA

```
db.StudentRecord.remove({"Name": "Tom"})
```

Remove Only One:-Removes first record

```
db.StudentRecord.remove(DELETION_CRITERIA,1)
```

Remove all Records

```
db.StudentRecord.remove()
```

Other functions

limit () method: to limit the number of documents retrieved

```
db.writers.find().pretty().limit(2)
```

skip () method

It is also possible to skip some documents from a MongoDB database.

```
db.writers.find().pretty().skip(1)
```


Other functions

sort() method:

For sorting your MongoDB documents, you need to make use of the *sort()* method.

This method will accept a document that has a list of fields and the order for sorting.

For indicating the sorting order, you have to set the value *1 (for ascending order)* or *-1 (for descending order)* with the specific entity based on which the ordering will be set and displayed.

The basic syntax for the sort() method is:

```
db.collection_name.find().sort({FieldName1: sort order 1 or -1,  
FieldName2: sort order})
```

**Thank
You**

