Table of Contents

[MONGODB 2](#_Toc115426296)

[JSON VERSUS BSON(BINARY JSON) 2](#_Toc115426297)

[WHY BSON? 2](#_Toc115426298)

[INSTALLATION 2](#_Toc115426299)

[MONGODB SHELL 2](#_Toc115426300)

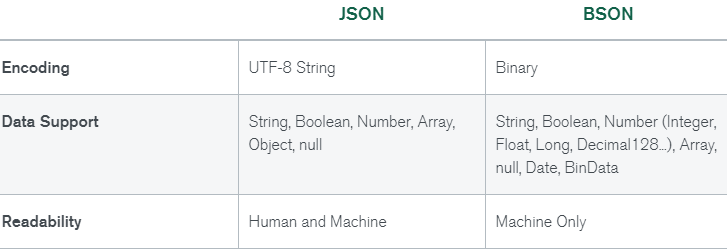
[MONGODB DATABASE, COLLECTIONS AND DOCUMENTS 2](#_Toc115426301)

[CRUD OPERATION 3](#_Toc115426302)

# MONGODB

* No SQL Server
* They the documents are serverless

## JSON VERSUS BSON(BINARY JSON)

* MongoDB use BSON to store data.
* The conversion of JSON (we use JSON input while creating documents) to BSON is done by MongoDB drivers. 

### WHY BSON?

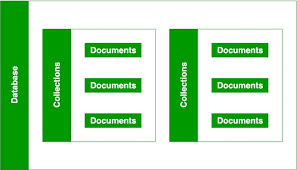
* BSON’s binary structure encodes type and length information, which allows it to be traversed much more quickly compared to JSON.
* BSON adds some non-JSON-native data types, like dates and binary data, without which MongoDB would have been missing some valuable support.

## INSTALLATION

### MONGODB SHELL

|  |  |
| --- | --- |
|  | * Mongo Db shell call be installed separately [https://www.mongodb.com/try/download/shell ] |

## MONGODB DATABASE, COLLECTIONS AND DOCUMENTS



## CRUD OPERATION

|  |  |
| --- | --- |
| TO SHOW DBs | ***show dbs*** |
| CREATING A DB | ***use <database\_name>***  This will not create a DB but it will be created only after we add a collection to it. |
| ADDING A DOCUMENT | ***db.<collection\_name>.<command>***   |  |  | | --- | --- | | **EXAMPLE**  db.flights.insertOne({  "departureAirport": "MUC",  "arrivalAirport": "SFO",  "aircraft": "Airbus A380",  "distance": 12000,  "intercontinental": true  }) |  | |
| INSERT MANY | db.flightData.insertMany([  {  "departureAirport": "MUC",  "arrivalAirport": "SFO",  "aircraft": "Airbus A380",  "distance": 12000,  "intercontinental": true  },  {  "departureAirport": "LHR",  "arrivalAirport": "TXL",  "aircraft": "Airbus A320",  "distance": 950,  "intercontinental": false  }  ]) |
| RETRIEVING DOCUMENTS  (all documents) | db.<collection\_name>.find() |
| RETRIEVING DOCUMENTS BASED IN FILTERS | **db.flightData.find({intercontinental:false})**  **USING OPERATOR**  **db.flightData.find({distance:{$gt:1000}})** |
| DELETING DOCUMENTS  (This deletes one data matching the filter) | **DELETE ONE:** **db.flightData.deleteOne({ "departureAirport": "LHR"})**  **DELETE MANY:** deleteMany() can be used to all document matching the filter criteria.  **db.flightData.deleteMany({},{delayed:'5 Hrs'})** |
| UPDATING DOCUMENT | **UPDATING AN EXISTIING DOCUMENT**  db.flightData.updateOne({aircraft:"Airbus A380"},{$set:{distance:15000}})  **ADDING A NEW FIELD IN A DOCUMENT**  **db.flightData.updateOne({aircraft:"Airbus A380"},{$set:{delayed:'2 Hrs'}})**  **UPDATE MANY**  **db.flightData.updateMany({},{$set:{delayed:'5 Hrs'}})** |
| * In the update operation we use “set” operator – for the data we want to update * If the field in not present in the document, it will add a new field | |

## EMBEDDED DOCUMENTS

## SCHEMA AND RELATIONS