



# Cognixia

A Collabera LEARNING SOLUTIONS COMPANY

## MongoDB – Best NoSQL DB



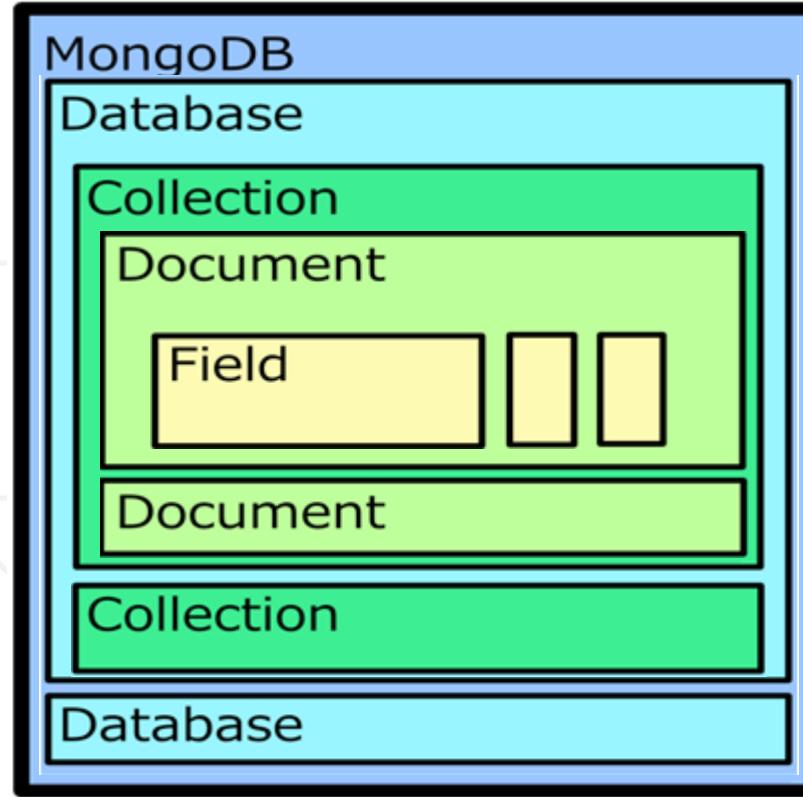
# Introduction

- MongoDB is an open-source document database and leading NoSQL database.
- MongoDB is written in C++.
- It is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability.
- Works on concept of collection and document.

# Introduction

- Database is a physical container for collections.
- Each database gets its own set of files on the file system.
- A single MongoDB server typically has multiple databases.

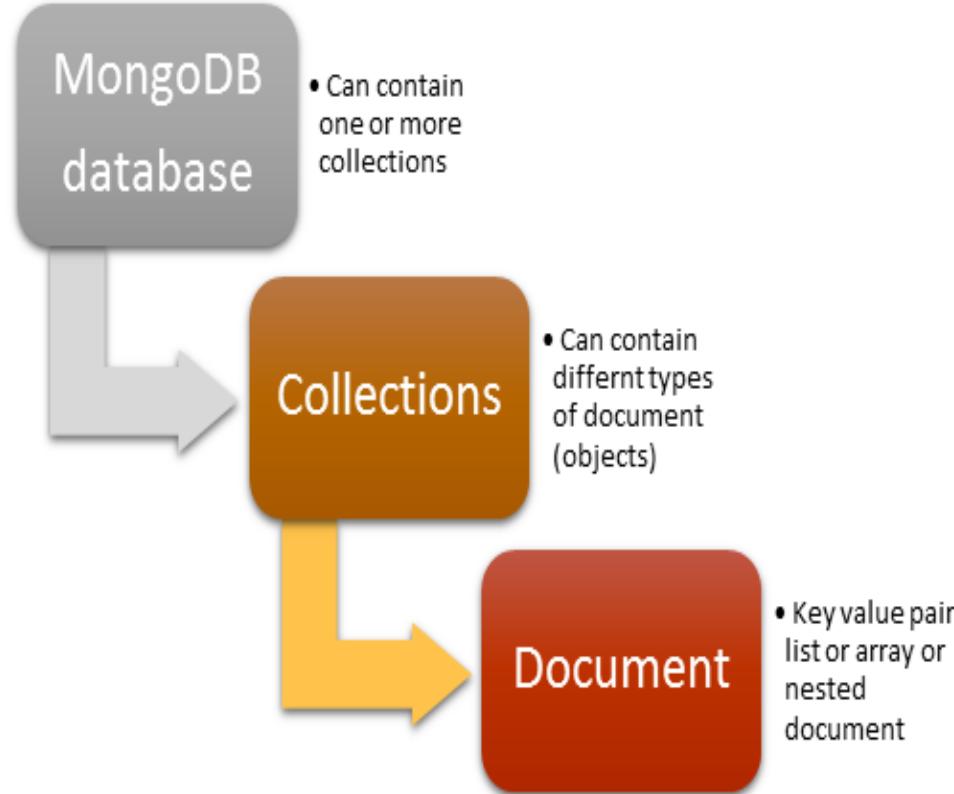
# Introduction



# Introduction

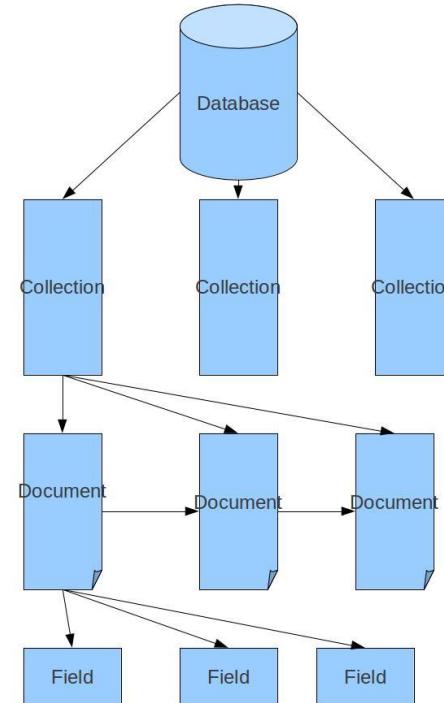
- Collection is a group of MongoDB documents.
- All documents in a collection are of similar or related purpose.

# Introduction



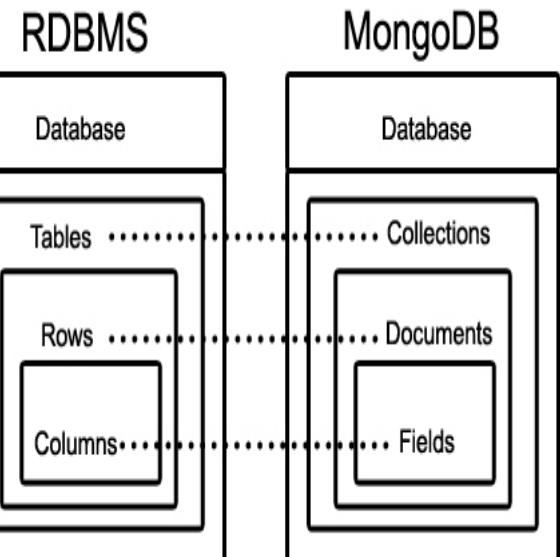
# Introduction

- A document is a set of key-value pairs.
- Documents have dynamic schema.
- Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.



# MongoDB Vs RDBMS

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo



# Advantages of MongoDB over RDBMS

- **Schema less** – MongoDB is a document database in which one collection holds different documents.
- Number of fields, content and size of the document can differ from one document to another.
- Structure of a single object is clear.
- No complex joins.
- Deep query-ability.
- Tuning.
- Ease of scale-out
- Conversion/mapping of application objects to database objects not needed.
- Uses internal memory for storing the (windowed) working set, enabling faster access of data.

# Why Use MongoDB?

- **Document Oriented Storage** – Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication and high availability
- Auto-sharding
- Rich queries
- Fast in-place updates
- Professional support by MongoDB

## Where to Use MongoDB?

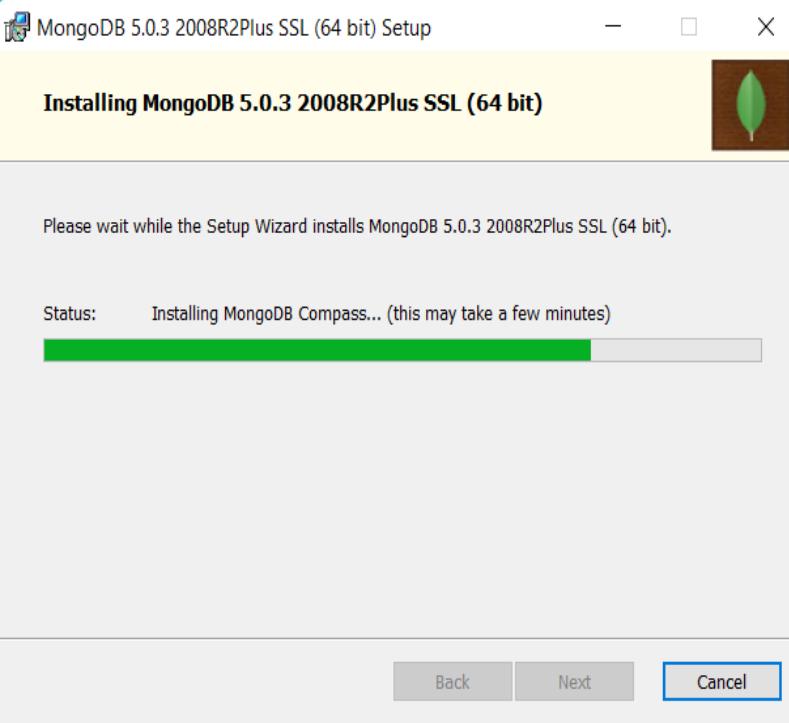
- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

# Download and Install MongoDB

- <https://www.mongodb.com/try/download/community>
- For Mac using Homebrew: [Install MongoDB Community Edition on macOS — MongoDB Manual](#)

The screenshot shows the MongoDB download page at https://www.mongodb.com/try/download/community. The top navigation bar includes links for Products, Solutions, Resources, Company, Pricing, a search icon, Sign In, and a Try Free button. Below the navigation, there are four main service categories: Atlas (MongoDB as a service), On-premises (MongoDB locally), Tools (Boost productivity), and Mobile & Edge (Realm Datastore). The "MongoDB Community Server" section is highlighted. It describes the Community version as a flexible document data model with support for ad-hoc queries, secondary indexing, and real-time aggregations. It also mentions MongoDB Atlas for managed services. A call-to-action button says "Give it a try with a free, highly-available 512 MB cluster." To the right, a "Available Downloads" sidebar allows users to filter by Version (5.0.3 current), Platform (Windows), Package (msi), and provides a "Download" button with a file icon and a "Copy Link" option. At the bottom of the sidebar, there are links for "Current releases & packages" and "Development releases".

Double-click the .msi file. Don't change any defaults. Click Install and finish.



# Download and Install MongoDB Shell

- [MongoDB Shell Download | MongoDB](https://www.mongodb.com/try/download/shell)

MongoDB Shell

MongoDB Shell is the quickest way to connect, configure, and administer your MongoDB database. It provides a modern and extensible command-line experience that is easy to learn and use. It features clear error messages. These are just some of the features that make MongoDB Shell the easiest way to manage your databases. The MongoDB Shell is a standalone application, released under the Apache 2 license.

Completed the MongoDB Shell Setup Wizard

Click the Finish button to exit the Setup Wizard.

Provides a command-line interface to MongoDB, making it easier to manage your databases. MongoDB Shell is open-source and released under the Apache 2 license.

Available Downloads

Version: 1.1.0

Platform: Windows 64-bit (8.1+) (MSI)

Package: msi

Download Copy Link

Documentation

# Start the Shell – Ensure you are logged in in MongoDB Compass

```
cmd mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
```

```
C:\Program Files\MongoDB\Server\5.0\bin>mongosh
Current Mongosh Log ID: 6171d25e1f485a65b3d84070
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Using MongoDB:          5.0.3
Using Mongosh:          1.1.0
```

```
For mongosh info see: https://docs.mongodb.com/mongodb-shell/
```

```
To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.
```

```
-----
The server generated these startup warnings when booting:
2021-10-21T16:11:16.197-04:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----
```

```
test> show dbs;
MongoshInvalidInputError: [COMMON-10001] 'dbs;' is not a valid argument for "show".
test> show dbs;
admin      41 kB
config     111 kB
local      41 kB
my_first_db 41 kB
test>
```

# Log into MongoDB Compass

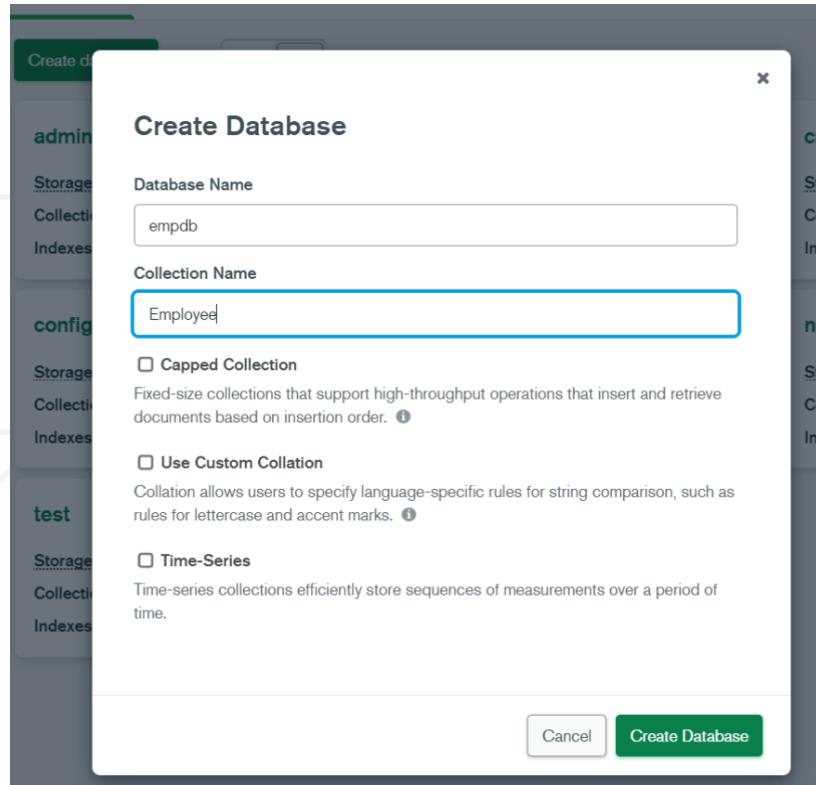
The screenshot shows the MongoDB Compass application interface. On the left, the main dashboard displays the connection status "MongoDB Compass - localhost:27017" and lists "3 DBS" and "1 COLLECTIONS". It also shows the host "localhost:27017", cluster type "Standalone", and edition "MongoDB 5.0.3 Community". A search bar at the bottom allows filtering by database names like "admin", "config", and "local".

On the right, a large window titled "New Connection" is open, prompting the user to "Paste your connection string (SRV or Standard)" in a text input field. Below the input field is a "Connect" button. To the right of the input field, there is a section titled "Fill in connection fields individually" with a link "Fill in fields individually".

Below the connection dialog, there are several informational sections:

- "New to Compass and don't have a cluster?" with a link to "MongoDB Atlas".
- A "CREATE FREE CLUSTER" button.
- "How do I find my connection string in Atlas?" with a link to "See example".
- "If you don't already have a cluster, you can create one for free using MongoDB Atlas."
- "How do I format my connection string?" with a link to "See example".

# Create a new Database



# Create a new Collection

The screenshot shows the MongoDB Compass interface. On the left, the sidebar displays the database structure under the 'Local' section. The 'empdb' database is selected, showing its collections: 'Employee' (selected), 'local', 'notesapp', 'test', 'config', 'cmsshoppingcart', 'admin', and 'ClassicModels'. The 'Employee' collection is expanded, showing its storage details: Storage size: 4.10 kB, Documents: 0, Avg. document size: 0 B, Indexes: 1, and Total index size: 4.10 kB. The main area is titled 'Collections' and features a green 'Create collection' button. A search bar at the top right allows sorting by 'Collection Name'.

# Import Data

The screenshot shows the MongoDB Compass interface with a file import dialog open. The dialog is titled "Import To Collection empdb.Employee". It displays a preview of the "employees.csv" file data:

	Emp_Id	Emp_Name	Emp_Dept	Emp_Salary
1	John Doe	Finance	50000	
2	Mike Russell	HR	45000	
3	Peter Wang	IT	60000	
4	Tom Clancy	Support	40000	
5	Sara Fargo	Operations	35000	

The "Select Input File Type" section has "CSV" selected, highlighted with a red box. The "Options" section includes "Select delimiter [COMMA]" and checkboxes for "Ignore empty strings" (checked) and "Stop on errors" (unchecked). The "Specify Fields and Types" section shows four fields: "Emp\_Id" (Number), "Emp\_Name" (String), "Emp\_Dept" (String), and "Emp\_Salary" (Double), all checked and highlighted with a red box.

# Commands on MongoDB Compass

MongoDB Compass - localhost:27017/empdb.Employee

Connect View Collection Help

Local

HOST localhost:27017

CLUSTER Standalone

EDITION MongoDB 5.0.3 Community

Filter your data

- ClassicModels
- admin
- cmissshoppingcart
- config
- empdb
  - Employee
- local
- notesapp
- test
- vuenodecmissshoppingcart

empdb.Employee

Documents Aggregations Schema Explain Plan Indexes Validation

DOCUMENTS 5 STORAGE SIZE 4.1KB AVG. SIZE 99B INDEXES 1 TOTAL SIZE 4.1KB AVG. SIZE 4.1KB

FILTER { Emp\_Dept : 'HR' } PROJECT { field: 0 } SORT { field: -1 } or [ [ 'field', -1 ] ] COLLATION { Locale: 'simple' }

MAX TIME MS 60000 SKIP 0 LIMIT 0

ADD DATA FIND RESET REFRESH

Displaying documents 1 - 1 of 1

```
_id: ObjectId("633b23a158b82983a489ec42")
Emp_id: 2
Emp_Name: "Mike Russell"
Emp_Dept: "HR"
Emp_Salary: 45000
```

# Basic commands on mongo shell

## What is MongoDB? — MongoDB Manual

The screenshot shows the MongoDB Compass interface connected to the 'empdb.Employee' collection. The collection has 5 documents, each with fields: \_id, Emp\_id, Emp\_Name, Emp\_Dept, and Emp\_Salary. The interface includes tabs for Documents, Aggregations, Schema, Explain Plan, Indexes, and Validation. A query builder is open with the following stages:

- FILTER**: { Emp\_Dept : 'HR' }
- PROJECT**: { field: 0 }
- SORT**: { field: -1 } or [ { 'field': -1 } ]
- COLLATION**: { Locale: 'simple' }

Buttons for FIND, RESET, and ... are present. The results pane displays one document:

```
_id: ObjectId("633b23a158b82983a489ec42")
Emp_id: 2
Emp_Name: "Mike Russell"
Emp_Dept: "HR"
Emp_Salary: 45000
```

The MONGOSH command line interface is highlighted with a red box at the bottom left, showing the following session:

```
MONGOSH
> use empdb;
< already on db empdb'
> db.Employee.find({Emp_Dept: 'HR'});
< { _id: ObjectId("633b23a158b82983a489ec42"),
    Emp_id: 2,
    Emp_Name: 'Mike Russell',
    Emp_Dept: 'HR',
    Emp_Salary: 45000 }
```

# Basic commands on mongo shell

- Show all available databases:

```
> show dbs;  
admin 0.000GB  
config 0.000GB  
local 0.000GB
```

- Select a particular database to access, e.g. mydb. This will create mydb if it does not already exist:

```
> use mydb;  
switched to db mydb
```

# Basic commands on mongo shell

- Show all functions that can be used with the database:

```
> db.mydb.help()
```

- To check your currently selected database, use the command db

```
> Db  
mydb
```

- command is used to drop a existing database.

```
> db.dropDatabase()  
{ "dropped" : "mydb", "ok" : 1 }
```

# Inserting records to ‘test’ collection

- We add three records to our collection test as:

```
> db.test.insert({"key":"value1","key2":"Val2","key3":"val3"})
WriteResult({ "nInserted" : 1 })
> db.test.insert({"key":"value2","key2":"Val21","key3":"val31"})
WriteResult({ "nInserted" : 1 })
```

- To find our test collections that we have added:

```
> db.test.find()
{ "_id" : ObjectId("5c2f07431cee9c4433924d3b"), "key" : "value1", "key2" : "Val2", "key3" : "val3" }
{ "_id" : ObjectId("5c2f07891cee9c4433924d3c"), "key" : "value2", "key2" : "Val21", "key3" : "val31" }
```

# Find ‘test’ collections

- To find our test collection that we have added and make them pretty readable, use the **pretty()** function.

```
> db.test.find().pretty()
{
    "_id" : ObjectId("5c2f07431cee9c4433924d3b"),
    "key" : "value1",
    "key2" : "Val2",
    "key3" : "val3"
}
{
    "_id" : ObjectId("5c2f07891cee9c4433924d3c"),
    "key" : "value2",
    "key2" : "Val21",
    "key3" : "val31"
}
```

# CRUD Operations : CREATE

- The difference with save is that if the passed document contains an \_id field, if a document already exists with that \_id it will be updated instead of being added as new.

```
> db.people.insert({name: 'Tom', age: 28});
```

Or

```
> db.people.save({name: 'Tom', age: 28});
```

- Use insertOne to insert only one record:

```
> db.people.insertOne({name: 'Tom', age: 28});
```

- Use insertMany to insert multiple records:

```
> db.people.insertMany([{name: 'Tom', age: 28},{name: 'John', age: 25}]);
```

# CRUD Operations : READ

- Query for all the docs in the people collection that have a name field with a value of 'Tom'

```
> db.people.find({name: 'Tom'})
```

- Finding the first record in a collection:

```
> db.people.findOne({name: 'Tom'})
```

- You can also specify which fields to return by passing a field selection parameter. The following will exclude the \_id field and only include the age field:

```
> db.people.find({name: 'Tom'}, {_id: 0, age: 1})
```

# CRUD Operations : READ

- If you want to find sub record like address object contains country, city, etc.

```
> db.people.find({'address.country': 'US'})
```

- Also specify the field too if required

```
> db.people.find({'address.country': 'US'}, {'name': true, 'address.city': true})
```

- Remember that the result has a `pretty()` method that pretty-prints resulting JSON:

```
> db.people.find().pretty()
```

# CRUD Operations : UPDATE

- Update the **entire** object:

```
> db.people.update({name: 'Tom'}, {age: 29, name: 'Tom'});
```

*// New in MongoDB 3.2*

```
> db.people.updateOne({name: 'Tom'}, {$set: {age: 29, name: 'Tom'}});  
//Will replace only first matching document.
```

```
> db.people.updateMany({name: 'Tom'}, {$set: {age: 29, name: 'Tom'}});  
//Will replace all matching documents.
```

- just update a single field of a document. In this case age:

```
> db.people.update({name: 'Tom'}, {$set: {age: 29}});
```

# CRUD Operations : UPDATE

- You can also update multiple documents simultaneously by adding a third parameter. This query will update all documents where the name equals Tom:

```
> db.people.update({name: 'Tom'}, {$set: {age: 29}}, {multi: true})
```

*// New in MongoDB 3.2*

```
> db.people.updateOne({name: 'Tom'}, {$set: {age: 29}});  
//Will updateonly first matching document.
```

```
> db.people.updateMany({name: 'Tom'}, {$set: {age: 31}});  
//Will update all matching documents.
```

```
> db.people.updateMany({name: 'Tom'},{$set:{age: 30, salary:50000}});  
// Document will have `salary` field as well.
```

# CRUD Operations : DELETE

- Deletes all documents matching the query parameter:

```
// New in MongoDB 3.2  
> db.people.deleteMany({name: 'Tom'})
```

```
// All versions  
> db.people.remove({name: 'Tom'})
```

- Delete one document:

```
// New in MongoDB 3.2  
> db.people.deleteOne({name: 'Tom'})
```

```
// All versions  
> db.people.remove({name: 'Tom'}, true)
```

# CRUD Operations : DELETE

- MongoDB's remove() method. If you execute this command without any argument or without empty argument it will remove all documents from the collection.

```
> db.people.remove({});
```

# Update of embedded documents

- For the following schema:  

```
{name: 'Tom', age: 28, marks: [50, 60, 70]}
```
- Update Tom's marks to 55 where marks are 50 (Use the positional operator \$):  

```
> db.people.update({name: "Tom", marks: 50}, {"$set": {"marks.$": 55}})
```
- For the following schema:  

```
{name: 'Tom', age: 28, marks: [{subject: "English", marks: 90}, {subject: "Maths", marks: 100}, {subject: "Computes", marks: 20}]}  
By using {name: "Tom", "marks.subject": "English"} you will get the position of the object in the marks array, where subject is English. In "marks.$.marks", $ is used to update in that position of the marks array
```

```
> db.people.update({name: "Tom", "marks.subject": "English"}, {"$set": {"marks.$.marks": 85}})
```

# Update of embedded documents

- Update values in an Array
  1. The positional \$ operator identifies an element in an array to update without explicitly specifying the position of the element in the array.
  2. Consider a collection students with the following documents:
    - { "\_id" : 1, "grades" : [ 80, 85, 90 ] }
    - { "\_id" : 2, "grades" : [ 88, 90, 92 ] }
    - { "\_id" : 3, "grades" : [ 85, 100, 90 ] }
  3. To update 80 to 82 in the grades array in the first document

```
> db.students.update({ _id: 1, grades: 80 },{ $set: { "grades.$" : 82 } });
```

# Getting database information

- List all collections in database

```
> show collections  
or  
> show tables  
or  
> db.getCollectionNames()
```

- List all databases

```
> show dbs  
or  
> db.adminCommand('listDatabases')  
or  
>db.getMongo().getDBNames()
```

# Querying for Data

- Retrieve all documents in a collection

```
> db.people.find({});
```

- Retrieve documents in a collection using a condition (similar to WHERE in MYSQL)

```
db.collection.find({key: value});
```

example :

```
> db.people.find({"name":"Tom"});
```

- Retrieve documents in a collection using Boolean conditions (Query Operators)

*//AND operator*

```
db.collection.find({ $and:[ { key:value },{ key: value } ] });
```

example :

```
> db.people.find({$and:[{"name":"Tom"}, {"age": 28}]});
```

# Querying for Data

//OR operator

```
db.collection.find({ $or:[ { key:value },{ key: value } ] });
```

example :

```
> db.people.find({$or:[{"name":"Tom"}, {"age": 28}]});
```

//NOT operator

```
db.inventory.find( { key: { $ne: value } } );
```

example:

```
> db.people.find({ "name": { $ne : "billl" } });
```

//similarly we have following operators

- \$eq: Matches values that are equal to a specified value.
- \$gt: Matches values that are greater than a specified value.
- \$gte: Matches values that are greater than or equal to a specified value.
- \$lt: Matches values that are less than a specified value.
- \$lte: Matches values that are less than or equal to a specified value.
- \$ne: Matches values that are not equal to a specified value.
- \$in: Matches all values that are not equal to a specified value.
- \$nin: Matches none of the values specified in an array.



# Cognixia

A Collabera LEARNING SOLUTIONS COMPANY

## THANK YOU



For more Information or set up an appointment kindly contact us today.