

# *Database Technologies – MongoDB*

```
Enterprise primaryDB> config.set("editor", "notepad++")
```

```
Enterprise primaryDB> config.set("editor", null)
```

NoSQL, which stands for "Not Only SQL" which is an alternative to traditional relational databases in which data is placed in tables and data schema is carefully designed before the database is built.

## NoSQL



**NoSQL** database are primarily called as **non-SQL** or **non-relational** database. MongoDB is Scalable (able to be changed in size or scale), open-source, high-perform, document-oriented database.

### Remember:

- **Horizontal scaling** means that you **scale** by adding more machines into your pool of resources.
- **Vertical scaling** means that you **scale** by adding more power (**CPU**, **RAM**) to an existing machine.

# why NoSQL



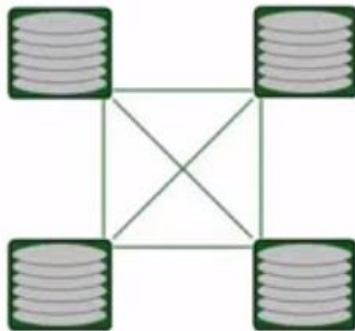
**Next Generation Databases**

**Not Only SQL**

**Not Only SQL**



**Non – Relational**



**Distributed Architecture**



**Open Source**



**Horizontally Scalable**

## When should NoSQL be used:

- When huge amount of data need to be stored and retrieved .
  - The relationship between the data you store is not that important
  - The data changing over time and is not structured.
  - Support of Constraints and Joins is not required at database level.
  - The data is growing continuously and you need to scale the database regular to handle the data.
- 

## Remember:

- Data Persistence on Server-Side via NoSQL.
  - Does not use SQL-like query language.
  - Longer persistence
  - Store massive amounts of data.
  - Systems can be scaled.
  - High availability.
  - Semi-structured data.
  - Support for numerous concurrent connections.
  - Indexing of records for faster retrieval
-

# NoSQL Categories

# NoSQL Categories

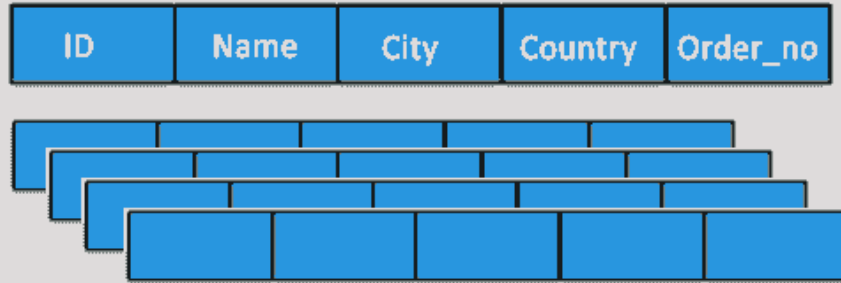
There are 4 basic types of NoSQL databases.

<i>Key-value stores</i>	<p>Key-value stores, or key-value databases, implement a simple data model that pairs a unique key with an associated value.</p> <p>e.g.</p> <ul style="list-style-type: none"><li>• <b>Redis</b></li></ul>
<i>Column-oriented</i>	<p>Wide-column stores organize data tables as columns instead of as rows.</p> <p>e.g.</p> <ul style="list-style-type: none"><li>• <b>hBase, Cassandra</b></li></ul>
<i>Document oriented</i>	<p>Document databases, also called document stores, store semi-structured data and descriptions of that data in document format.</p> <p>e.g.</p> <ul style="list-style-type: none"><li>• <b>MongoDB, CouchDB</b></li></ul>
<i>Graph</i>	<p>Graph data stores organize data as nodes.</p> <p>e.g.</p> <ul style="list-style-type: none"><li>• <b>Neo4j</b></li></ul>

# NoSQL Categories

## Column-oriented

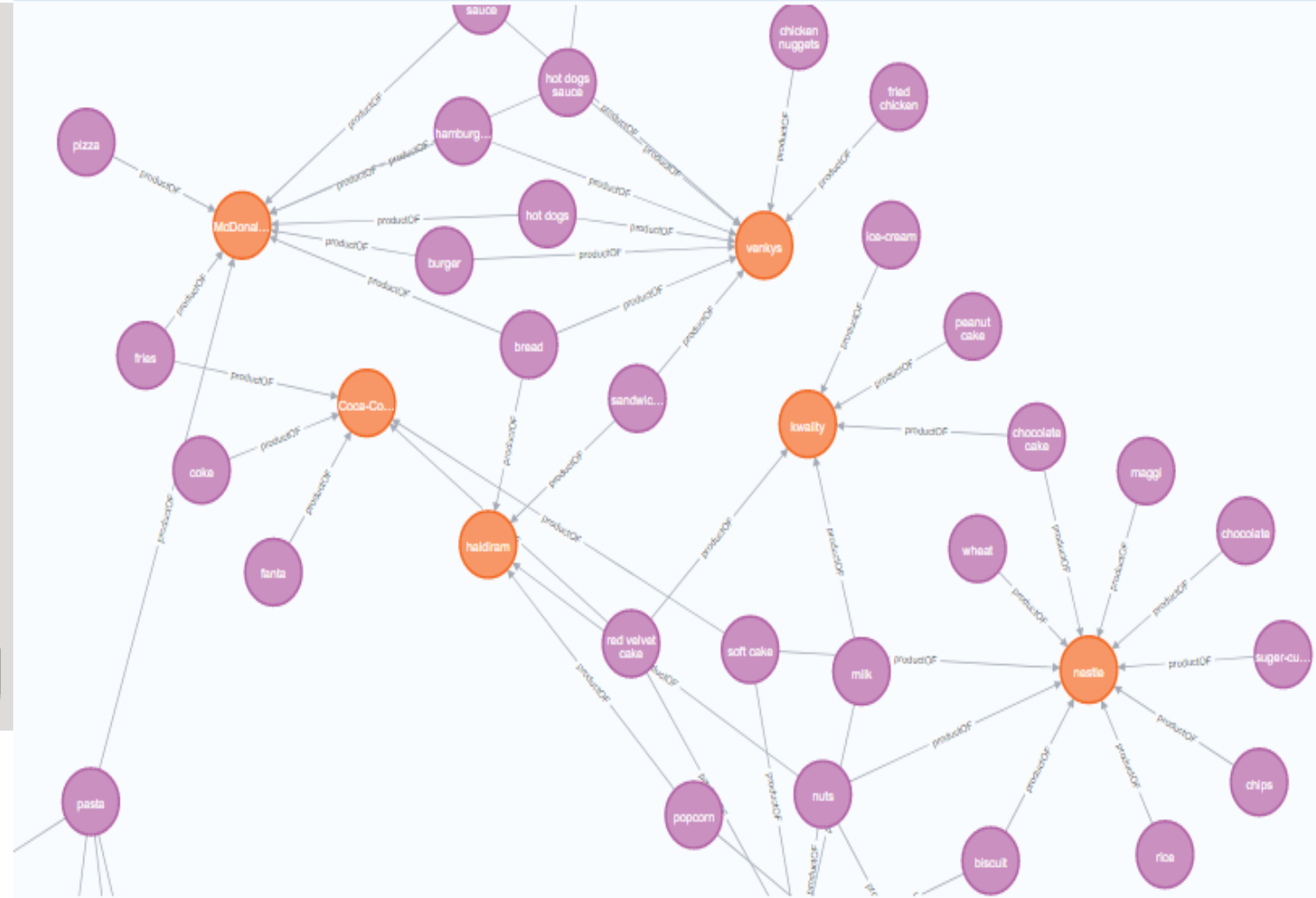
### row-store



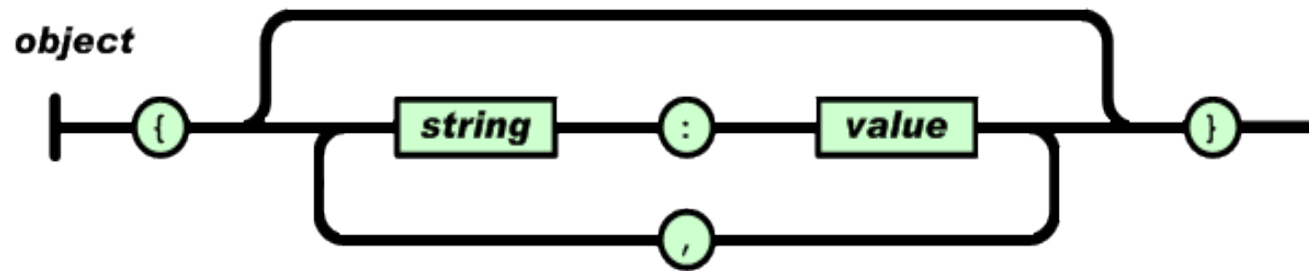
### column-store



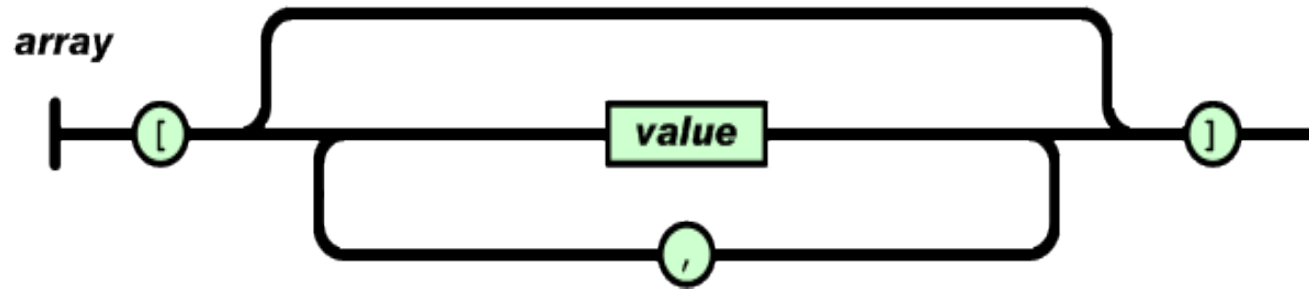
## Graph



An object is an unordered set of name/value pairs.



An array is an ordered collection of values.





# SQL vs NoSQL Database

**Relational databases** are commonly referred to as SQL databases because they use **SQL** (structured query language) as a way of storing and querying the data.

## Difference:

- NoSQL databases are document based, key-value pairs, or wide-column stores. This means that SQL databases represent data in form of tables which consists of  $n$  number of rows of data whereas NoSQL databases are the collection of key-value pair, documents, or wide-column stores which do not have standard schema definitions.
  - SQL databases have predefined schema whereas NoSQL databases have dynamic schema for unstructured data.
  - SQL databases are vertically scalable whereas the NoSQL databases are horizontally scalable.
  - SQL databases uses SQL ( structured query language ) for defining and manipulating the data. In NoSQL database, queries are focused on collection of documents.
-

# Types of Data

## Structured



0.103	0.176	0.387	0.300	0.379
0.333	0.384	0.564	0.587	0.857
0.421	0.309	0.654	0.729	0.228
0.266	0.750	1.056	0.936	0.911
0.225	0.326	0.643	0.337	0.721
0.187	0.586	0.529	0.340	0.829
0.153	0.485	0.560	0.428	0.628

## Semi-Structured

```
{
  "_id" : 1001,
  "Name" : "Saleel Bagde",
  "canVote" : true
},
{
  "_id" : 1002,
  "Name" : "Sharmin Bagde",
  "canVote" : true,
  "canDrive" : false
}
```

## Unstructured



**MongoDB** stores **documents** (objects) in a format called **BSON**.  
**BSON** is a binary serialization of JSON

- *Structured*

The data that can be stored and processed in a fixed format is called as Structured Data. Data stored in a relational database management system (RDBMS) is one example of 'structured' data. It is easy to process structured data as it has a fixed schema. Structured Query Language (SQL) is often used to manage such kind of Data.

- *Semi-Structured*

Semi-Structured Data is a type of data which does not have a formal structure of a data model, i.e. a table definition in a relational DBMS, XML files or JSON documents are examples of semi-structured data.

- *Unstructured*

The data which have unknown form and cannot be stored in RDBMS and cannot be analyzed unless it is transformed into a structured format is called as unstructured data. Text Files and multimedia contents like images, audios, videos are example of unstructured data.

---

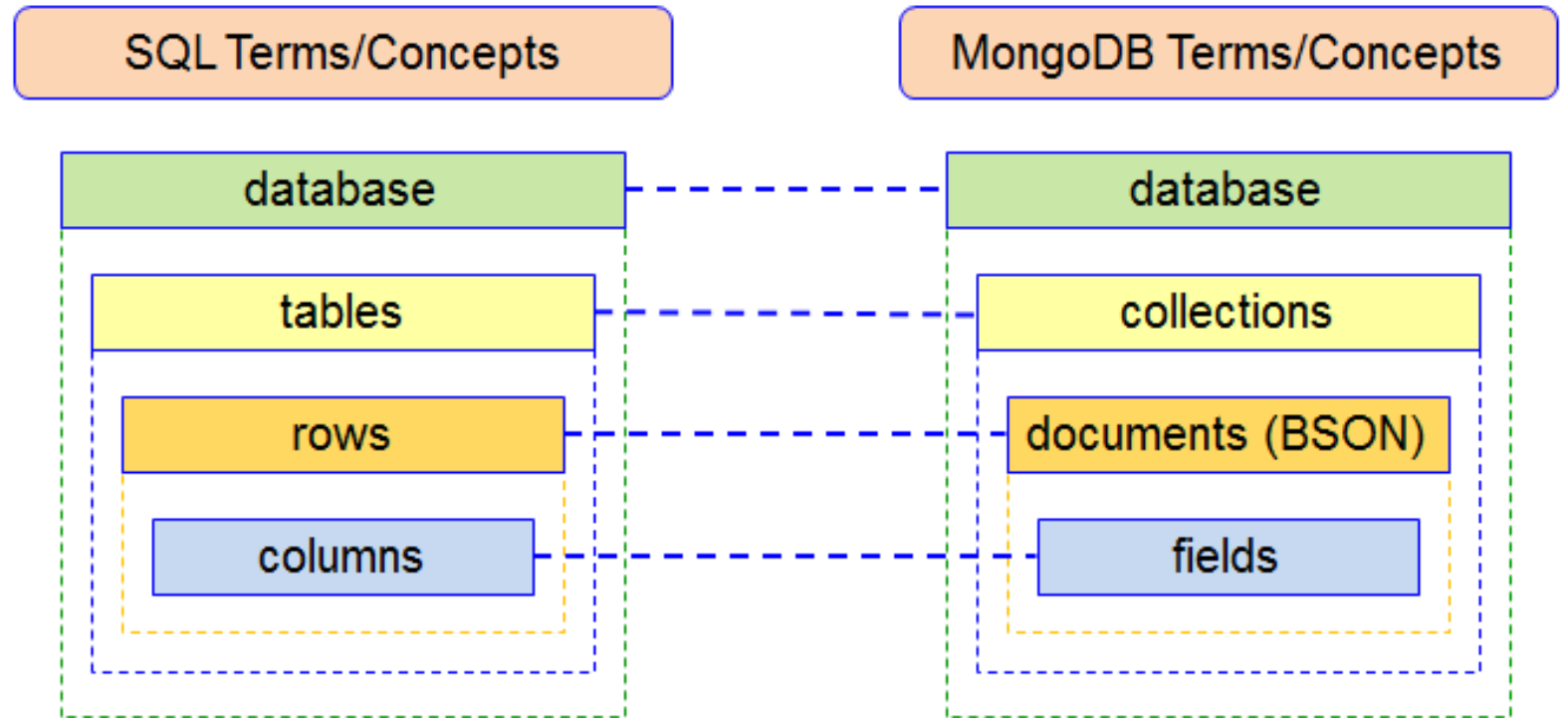
MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database.

## Remember:

- MongoDB documents are similar to JSON (key/fields and value pairs) objects.
- The values of fields may include other documents, arrays, or an arrays of documents.

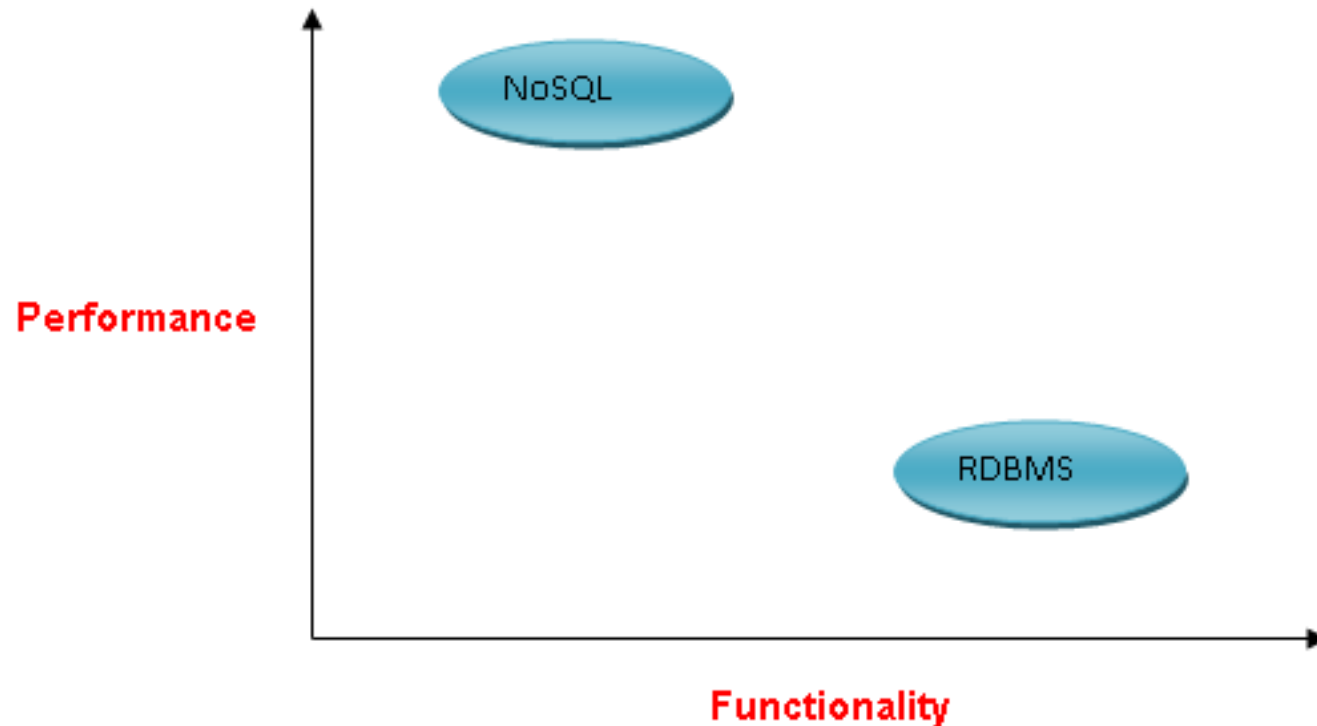
Core MongoDB Operations (CRUD), stands for **create**, **read**, **update**, and **delete**.

## SQL/MongoDB Terms:



MongoDB stores data as BSON documents. BSON is a binary representation of JSON documents.

*JSON* (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write.



## \* MongoDB does not support duplicate field names

The maximum size an individual document can be in MongoDB is **16MB with a nested depth of 100 levels**.

- `cls`
- `console.clear();`

## document

MongoDB stores data as BSON documents. BSON is a binary representation of JSON documents.

MongoDB's **collections**, by default, do not require their **documents** to have the same schema. That is:

- The documents in a single collection do not need to have the same set of fields and the data type for a field can differ across documents within a collection.
- To change the structure of the documents in a collection, such as add new fields, remove existing fields, or change the field values to a new type, update the documents to the new structure.

## PERSON

Pers_ID	First_Name	Last_Name	City
0	Paul	Miller	London
1	Alvaro	Ortega	Valencia
2	Bianca	Bertolini	Rome
3	Aurieles	Jackson	Paris
4	Urs	Huber	Zurich

## CAR

Car_ID	Model	Year	Value	Pers_ID
101	Bentley	1973	100000	0
102	Renault	1993	2000	3
103	Smart	1999	2000	2
104	Ferrari	2005	150000	4
105	Rolls Royce	1965	350000	0
106	Renault	2001	7000	3
107	Peugeot	1993	500	3

## People Collection

```
{
  id: 0,
  first_name: 'Paul',
  last_name: 'Miller',
  city: 'London',
  cars: [
    {
      model: 'Bentley',
      year: 1973,
      color: 'gold',
      value: NumberDecimal('100000.00'),
      currency: 'USD',
      owner: 0
    },
    {
      model: 'Rolls Royce',
      year: 1965,
      color: 'brewster green',
      value: NumberDecimal('350000.00'),
      currency: 'USD',
      owner: 0
    }
  ]
}
```

MongoDB documents are composed of *field-and-value* pairs. The value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents.

The *field name* **\_id** is reserved for use as a primary key; its value must be unique in the collection, is immutable, and may be of any type other than an array.

```
{
  field1: value,
  field2: value,
  field3: [],
  field4: {},
  field5: [ {}, {}, ... ]
  ...
  fieldN: valueN
}
```

The primary key **\_id** is automatically added, if **\_id** field is not specified.

## Note:

- The **\_id** field is always the first field in the documents.
- MongoDB does not support duplicate field names.



# db

In the mongo shell, **db** is the variable that references the current database. The variable is automatically set to the default database **test** or is set when you use the **use <db\_name>** to switch current database.

	MongoDB	Redis	MySQL	Oracle
Database Server	mongod	./redis-server	mysqld	oracle
Database Client	mongo	./redis-cli	mysql	sqlplus

start db server

# start server and client

To start **MongoDB server**, execute **mongod.exe**.

**Note:** Always give --dbpath in ""

- The --dbpath option points to your database directory.
- The --bind\_ip\_all option : bind to all ip addresses.
- The --bind\_ip arg option : comma separated list of ip addresses to listen on, localhost by default.

```
--bind_ip <hostnames | ipaddresses>
```

```
mongod --dbpath "c:\database" --bind_ip_all --journal
mongod --dbpath "c:\database" --bind_ip stp10 --journal
mongod --dbpath "c:\database" --bind_ip 192.168.100.20 --journal
mongod --dbpath="c:\database" --bind_ip=192.168.100.20 -journal
mongod -auth --dbpath="c:\database" --bind_ip=192.168.100.20 --journal
mongod --storageEngine inMemory --dbpath="d:\tmp" --bind_ip=192.168.100.20
```

To start **MongoDB client**, execute **mongo.exe**.

```
mongo "192.168.100.20:27017/primaryDB"
mongo --host 192.168.100.20 --port 27017
mongo --host 192.168.100.20 --port 27017 primaryDB
mongo --host=192.168.100.20 --port=27017 primaryDB
mongo --host=192.168.100.20 --port=27017 -u user01 -p user01 --
authenticationDatabase primaryDB
```



*must be empty folder*

- `db.version();`      # version number
- `db.getMongo();`      # connection to 192.168.100.20:27017
- `db.hostInfo();`      # Returns a document with information about the mongoDB is runs on.
- `db.stats();`      # Returns DB status
- `getHostName();`      # stp5

# comparison operator

## comparison operator

\$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 all 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.

# comparison operator

\$eq

```
{ field: { $eq: value } }
```

\$ne

```
{ field: { $ne: value } }
```

\$gt

```
{ field: { $gt: value } }
```

\$gte

```
{ field: { $gte: value } }
```

\$lt

```
{ field: { $lt: value } }
```

\$lte

```
{ field: { $lte: value } }
```

\$in

```
{ field: { $in: [ <value1>, <value2>, ..., <valueN> ] } }
```

\$nin

```
{ field: { $nin: [ <value1>, <value2>, ..., <valueN> ] } }
```

logical operator

# logical operator

\$or	Joins query clauses with a logical <b>OR</b> returns all documents that match the conditions of either clause.
\$and	Joins query clauses with a logical <b>AND</b> returns all documents that match the conditions of both clauses.
\$not	Inverts the effect of a query expression and returns documents that do not match the query expression.

- ```
db.emp.find( { $or:[ { job: 'manager' }, { job: 'salesman' } ], $and: [ { sal: { $gt: 3000 } } ] }, { _id: false, ename: true, job: true, sal: true } )
```



# logical operator

## \$or

{ \$or: [ { <expr1> }, { <expr2> }, ... , { <exprN> } ] }

- db.emp.find({\$or: [{job: 'manager'}, {job: 'salesman'}]}))

## \$and

{ \$and: [ { <expr1> }, { <expr2> }, ... , { <exprN> } ] }

- db.emp.find({\$and: [{job: 'manager'}, {sal: 3400}]}))

## \$not

{ field: { \$not: { <operator-expression> } } }

- db.emp.find({ job: { \$not: { \$eq: 'manager' } } } )

# arithmetic expression operators

- `db.movies.aggregate([ { $project: { _id: false, "Title": true, R: { $round: { $multiply: [ { $rand: {} }, 800 ] } } } } ])`

# arithmetic expression operators

Arithmetic expressions perform mathematic operations on numbers. Some arithmetic expressions can also support date arithmetic.

| Arithmetic expressions |                                                                      |
|------------------------|----------------------------------------------------------------------|
| \$abs                  | x: { \$abs: '\$<number>' }                                           |
| \$add                  | x: { \$add: ['\$<expression1>', '\$<expression2>', ... ] }           |
| \$subtract             | x: { \$subtract: ['\$<expression1>', '\$<expression2>' ] }           |
| \$multiply             | x: { \$multiply: ['\$<expression1>', '\$<expression2>', ...] }       |
| \$divide               | x: { \$divide: ['\$<expression1>', '\$<expression2>' ] }             |
| \$mod                  | x: { \$mod: ['\$<expression1>', '\$<expression2>' ] }                |
| \$round                | x: { \$round: [ <number>, <place> ] }                                |
| \$trunc                | x: { \$trunc: '\$<number>' } x: { { \$round: [ <number>, <place> ] } |
| \$rand                 | x: { \$rand: { } }                                                   |

\$ifNull(), \$toUpper, \$toLower,  
\$concat, . . .

# ***\$ifNull(), \$toUpper(), \$toLower(), \$concat***

Evaluates an expression and returns the value of the expression if the expression evaluates to a non-null value. If the expression evaluates to a null value, including instances of undefined values or missing fields, returns the value of the replacement expression.

```
x: { $ifNull:[ '$<expression>', <replacement-expression-if-null> ] }
```

```
x: { $toUpper: '$<expression>' }
```

```
x: { $toLower: '$<expression>' }
```

```
x: { $strlenCP: '$<expression>' } // string expression
```

```
x: { $concat:[ '$<expression1>', '$<expression2>', ... ] }
```

```
x: { $substr: [ <string>, <start>, <length> ] }
```

```
x: { $size: '$<expression>' }
```

```
x: { $arrayElemAt: ['$<array>', <idx> ] } // -1 will get last element from array
```

```
x: { $split: ['$<expression>', <delimiter> ] }
```

# ObjectId()

The ObjectId class is the default primary key for a MongoDB document and is usually found in the `_id` field in an inserted document.

The **`_id`** field must have a unique value. You can think of the **`_id`** field as the document's primary key.

MongoDB uses ObjectIds as the default value of `_id` field of each document, which is auto generated while the creation of any document.

`ObjectId()`

- `x = ObjectId()`

# show databases and get current db name

Print a list of all available databases.



# *show database*

Print a list of all databases on the server.

```
show { dbs | databases }
```

- `show dbs`
- `show databases`      `# returns: all database name.`

```
db.getName()
```

- `db`
- `db.getName()`      `# returns: the current database name.`

# use database

Switch current database to `<db>`. The mongo shell variable `db` is set to the current database.

# *use database*

Switch current database to <db>.The mongo shell variable db is set to the current database.

```
use <db>
```

- use db1



# *db.dropDatabase()*

Removes the current database, deleting the associated data files.

```
db.dropDatabase()
```

- `use db1`
- `db.dropDatabase()`

**If not working then do changes in *my.ini* file.**

secure\_file\_priv = ""

- `SELECT * FROM emp INTO OUTFILE "d:/emp.csv" FIELDS TERMINATED BY ',';`

## mongoimport

`mongoimport` tool imports content from an Extended JSON, CSV, or TSV export created by `mongoexport`, or another third-party export tool.

# mongoimport - JSON

The *mongoimport* tool imports content from an Extended JSON, CSV, or TSV export created by *mongoexport*.

```
mongoimport < --host > < --port > < --db > < --collection > < --type >  
< --file > < --fields "Field-List" > < --mode { insert | upsert | merge  
} > < --jsonArray > < --drop >
```

```
< --jsonArray > # if the documents are in array i.e. in [] brackets  
< --drop >      # drops the collection if exists
```

- C:\> mongoimport --host 192.168.0.3 --port 27017 --db db1 --collection emp --type json --file "d:\emp.json"
- C:\> mongoimport --host 192.168.0.6 --port 27017 --db db1 --collection movies --type json --file "d:\movies.json" --jsonArray --drop

# mongoimport - CSV

The *mongoimport* tool imports content from an Extended JSON, CSV, or TSV export created by *mongoexport*.

```
mongoimport < --host > < --port > < --db > < --collection > < --type > < --file >  
< --fields "<field1>[,<field2>]*" > < --headerline > < --useArrayIndexFields >
```

- C:\> mongoimport --host 192.168.100.20 --port 27017 --db db1 --collection emp --type csv --file d:\emp.csv --headerline
- C:\> mongoimport --host 192.168.100.20 --port 27017 --db db1 --collection emp --type csv --file d:\emp.csv --fields  
"EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO,BONUSID,USERNAME,PWD"
- C:\> mongoimport --db db1 --collection o --type csv --file d:\emp.csv --fields  
"EMPNO.int(32),ENAME.string(),JOB.string(),MGR.int32(),HIREDATE.date(2006-01-02),SAL.int32(),COMM.int32(),DEPTNO.int32(),BONUSID.int32(),USERNAME.string(),PWD.string())"

## Note:

- There should be no blank space in the field list.

e.g.

\_id, ename, salary    *#this is an error*



# mongoimport - CSV

The *mongoimport* tool imports content from an Extended JSON, CSV, or TSV export created by *mongoexport*.

```
mongoimport < --host > < --port > < --db > < --collection > < --type > < --file >  
< --fields "<field1>[,<field2>]*" > < --headerline > < --useArrayIndexFields >
```

```
_id,course,duration,modules.0,modules.1,modules.2,modules.3  
1,course1,6 months,c++,database,java,.net  
2,course2,6 months,c++,database,python,R  
3,course3,6 months,c++,database,awp,.net
```

- C:\> mongoimport --host 192.168.100.20 --port 27017 --db db1 --collection course --type csv --file d:\course.csv --headerline --useArrayIndexFields

# mongoexport

`mongoexport` is a utility that produces a JSON or CSV export of data stored in a MongoDB instance.

# mongoexport

*mongoexport* is a utility that produces a JSON or CSV export of data stored in a MongoDB instance..

```
mongoexport < --host > < --port > < --db > < --collection > < --type > < --file >  
< --out >
```

- C:\> mongoexport --host 192.168.0.6 --port 27017 --db db1 --collection emp --type JSON --out "d:\emp.json"
- C:\> mongoexport --host 192.168.0.6 --port 27017 --db db1 --collection emp --type JSON --out "d:\emp.json" --fields "empno,ename,job"
- C:\> mongoexport --host 192.168.0.6 --port 27017 --db db1 --collection emp --type CSV --out "d:\emp.csv" --fields "empno,ename,job"

## Note:

- there should be no space in the field list.  
e.g.  
\_id, ename, salary #this is an error

# db.getCollectionNames() / db.getCollectionInfos()

Returns an array containing the names of all collections and views in the current database.

# *db.getCollectionNames() / db.getCollectionInfos()*

*getCollectionNames()* returns an array containing the names of all collections in the current database.

```
show collection
db.getCollectionNames()
db.getCollectionInfos()
```

- `show collections`
- `db.getCollectionNames()`
- `db.getCollectionInfos()`

# db.createCollection()

Creates a new collection or view.

# db.createCollection()

*Capped* collections have maximum size or document counts that prevent them from growing beyond maximum thresholds. All capped collections must specify a maximum size and may also specify a maximum document count. **MongoDB removes older documents if a collection reaches the maximum size limit before it reaches the maximum document count.**

```
db.createCollection(name, { options1, options2, ... })
```

The options document contains the following fields:

- `capped` : boolean
- `size` : number
- `max` : number

The size parameter specifies the size of the capped collection in bytes.

- `db.createCollection("log")`
- `db.createCollection("log", { capped: true, size: 1, max: 2})` // This command creates a collection named log with a maximum size of 1 byte and a maximum of 2 documents.

# db.getCollection()

Returns a collection or a view object that is in the DB.



## TODO

```
db.getCollection('name')
```

- ```
db.getCollection('emp').find()
```
- ```
const auth = db.getCollection("author")
const doc = {
  usrName : "John Doe",
  usrDept : "Sales",
  usrTitle : "Executive Account Manager",
  authLevel : 4,
  authDept : ["Sales", "Customers"]
}
```

```
auth.insertOne( doc )
```

# db.getSiblingDB()

To access another database without switching databases.

## ***db.getSiblingDB()***

Used to return another database without modifying the db variable in the shell environment.

```
db.getSiblingDB (<database>)
```

- `db.getSiblingDB ('db1') .getCollectionNames ()`

# db.collection.renameCollection()

Renames a collection.

# *db.collection.renameCollection()*

TODO

```
db.collection.renameCollection(target, dropTarget)
```

- `db.emp.renameCollection("employee", false)`

**dropTarget** : If true, mongod drops the target of renameCollection prior to renaming the collection. The default value is false.

# db.collection.drop()

Removes a collection or view from the database. The method also removes any indexes associated with the dropped collection.

# ***db.collection.drop()***

*drop()* removes a collection or view from the database. The method also removes any indexes associated with the dropped collection.

```
db.collection.drop()
```

- `db.emp.drop()`

# db.collection.find()

The `find()` method always returns the `_id` field unless you specify `_id: 0/false` to suppress the field.

**By default, mongo prints the first 20 documents.** The mongo shell will prompt the user to **Type "it" to continue** iterating the next 20 results.



# db.collection.find()

## TODO

```
db['collection'].find({ query }, { projection })
db.collection.find({ query }, { projection })
db.getCollection('name').find({ query }, { projection })
```

**query**: Specifies selection filter using query operators. To return all documents in a collection, omit this parameter or pass an empty document ({}).

```
{ "<Field Name>": { "<Comparison Operator>": <Comparison Value> } }
```

**projection**: Specifies the fields to return in the documents that match the query filter. To return all fields in the matching documents, omit this parameter.

```
{ "<Field Name>": <Boolean Value> } }
```

## Remember

- 1 or true to include the field in the return documents. Non-zero integers are also treated as true.
- 0 or false to exclude the field.