MongoDB - NoSQL database

Alex Libis , Ph.D.

Common Terms in MongoDB

_id – This is a field required in every MongoDB document.
The _id field represents a unique value in the MongoDB document.
The _id field is like the document's primary key.
If you create a new document without an _id field, MongoDB will automatically create the field.
So for example, if we see the example of the above customer table, Mongo DB will add a 24 digit unique identifier to each document in the collection.

_Id	CustomerID	CustomerName	OrderID
563479cc8a8a4246bd27d784	11	Alex	111
563479cc7a8a4246bd47d784	22	Dganit	222
563479cc9a8a4246bd57d784	33	Avi	333

Common Terms in MongoDB

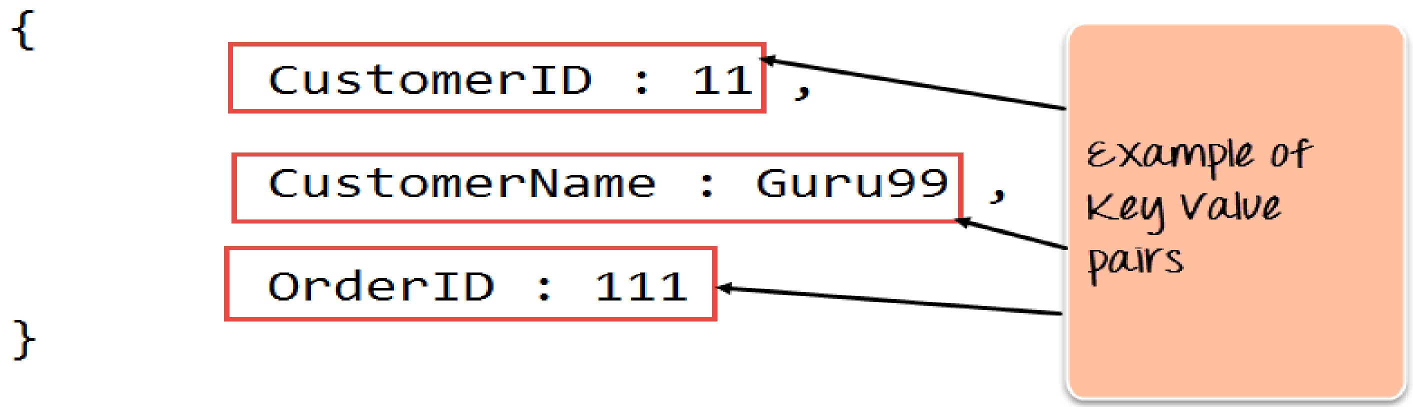
Collection – This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS such as Oracle.
A collection exists within a single database and don't enforce any sort of structure.

Cursor – This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.

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.

Document - A record in a MongoDB collection is basically called a document.
The document in turn will consist of field name and values.

Field - A name-value pair in a document. A document has zero or more fields.
Fields are analogous to columns in relational databases.
The following diagram shows an example of Fields with Key value pairs.
So in the example below CustomerID and 11 is one of the key value pair's defined in the document.



```
{
        CustomerID : 11 ,
        CustomerName : Guru99 ,
        OrderID : 111
}
```

Common Terms in MongoDB

JSON – This is known as JavaScript Object Notation.
This is a human-readable, plain text format for expressing structured data.
JSON is currently supported in many programming languages.

Just a quick note on the key difference between the _id field and a normal collection field.
The _id field is used to uniquely identify the documents in a collection and is automatically added by MongoDB when the collection is created.

Introduction to MongoDB

Below are the few of the reasons as to why one should start using MongoDB

1. Document-oriented – Since MongoDB is a NoSQL type database, instead of having data in a relational type format,
   it stores the data in documents.
   This makes MongoDB very flexible and adaptable to real business world situation and requirements.
2. Ad hoc queries - MongoDB supports searching by field, range queries, and regular expression searches.
   Queries can be made to return specific fields within documents.
3. Indexing - Indexes can be created to improve the performance of searches within MongoDB.
   Any field in a MongoDB document can be indexed.
4. Replication - MongoDB can provide high availability with replica sets. A replica set consists of two or more mongo D
   Each replica set member may act in the role of the primary or secondary replica at any time.
   The primary replica is the main server which interacts with the client and performs all the read/write operations.
   The Secondary replicas maintain a copy of the data of the primary using built-in replication.
   When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the p
5. Load balancing - MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple Mong
   MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system

Difference between MongoDB & RDBMS

RDBMS	MongoDB	Difference
Table	Collection	In RDBMS, the table contains the columns and rows which are used to store the data whereas, in MongoDB, this same structure is known as a collection. The collection contains documents which in turn contains Fields, which in turn are key-value pairs.
Row	Document	In RDBMS, the row represents a single, implicitly structured data item in a table. In MongoDB, the data is stored in documents.
Column	Field	In RDBMS, the column denotes a set of data values. These in MongoDB are known as Fields.
Joins	Embedded documents	In RDBMS, data is sometimes spread across various tables and in order to show a complete view of all data, a join is sometimes formed across tables to get the data. In MongoDB, the data is normally stored in a single collection, but separated by using Embedded documents. So there is no concept of joins in Mongodb.

Practice

.

Create and Drop Database in MongoDB

It’s strange to listen but true that MongoDB doesn’t provide any command to create databases. Then the question is how would we create database ?.
The answer is – We don’t create database in MongoDB, we just need to use database with your preferred name and need to save a single record in database to create it.

List Databases – First check the current databases in our system.

```
# mongo > show dbs;
 admin (empty)
 local 0.078GB
 test 0.078GB
```

Create and Drop Database in MongoDB

Use New Database –

Now if we want to create database with name myfirstdb.

Just run following command and save a single record in database.

After saving your first example, you will see that new database has been created.

```
> use myfirstdb;
>s = { Name : “stam" }
> db.testData.insert( s );
```

List Databases –

Now if you list the databases, you will see the new database will be their with name exampledb.

```
> show dbs;
```

Create and Drop Database in MongoDB

MongoDB provides dropDatabase() command to drop currently used database with their associated data files.

Before deleting make sure what database are you selected using db command.

```
> db myfirstdb
```

Now if you execute dropDatabase() command. It will remove exampledb database.

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

To delete MongoDB database from Linux command line or Shell Script –
 Use following command from

```
# mongo myfirstdb --eval "db.dropDatabase()"
```

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>)

db.collection.find()

db.collection.find(query, projection)

Selects documents in a collection or view and returns a cursor to the selected documents.

RDBMS	MONGO
SELECT * FROM inventory WHERE status = "D"	db.inventory.find({ status: "D" })

SELECT * FROM inventory WHERE status in ("A", "D")
db.inventory.find({ status: { $in: ["A", "D"] } })
	db.inventory.find({ status: { $in: ["A", "D"] } })

SELECT * FROM inventory WHERE status = "A" AND qty < 30
db.inventory.find({ status: "A", qty: { $lt: 30 } })
	db.inventory.find({ status: "A", qty: { $lt: 30 } })

SELECT * FROM inventory WHERE status = "A" OR qty < 30
db.inventory.find({ $or: [{ status: "A" }, { qty: { $lt: 30 } }] })
	db.inventory.find({ $or: [{ status: "A" }, { qty: { $lt: 30 } }] })

SELECT * FROM inventory WHERE status = "A" AND (qty < 30 OR item LIKE "p%")
db.inventory.find({   status: "A", $or: [{ qty: { $lt: 30 } }, { item: /^p/ }]})
	db.inventory.find({ status: "A", $or: [{ qty: { $lt: 30 } }, { item: /^p/ }]})

db.collection.find()

db.collection.find(query, projection)

Selects documents in a collection or view and returns a cursor to the selected documents.

- db.bios.find().limit(5)
- var myCursor = db.bios.find( );
  myCursor.forEach(printjson);
- db.bios.find().sort({ name: 1 })
- db.bios.find().skip(5)

Combine Examples:

db.bios.find().sort({ name: 1 }).limit(5)
db.bios.find().limit(5).sort({ name: 1 })

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 loginal 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

https://docs.mongodb.org/manual/reference/operator/query/

Date()

•Date() returns the current date as a string in the mongo shell.
•new Date() returns the current date as a Date object. The mongo shell wraps the Date object with the ISODate helper. The ISODate is in UTC

•new Date("<YYYY-mm-dd>") returns the ISODate with the specified date.
•new Date("<YYYY-mm-ddTHH:MM:ss>") specifies the datetime in the client's local timezone and returns the ISODate with the specified datetime in UTC.
•new Date("<YYYY-mm-ddTHH:MM:ssZ>") specifies the datetime in UTC and returns the ISODate with the specified datetime in UTC.
•new Date(<integer>) specifies the datetime as milliseconds since the Unix epoch (Jan 1, 1970), and returns the resulting ISODate instance.

Return Date as a String

```
var myDateString = Date();
var myDate = new Date();
var myDateInitUsingISODateWrapper = ISODate();
```

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/
})
```

Differences Between Interactive and Scripted mongo

show dbs, show databases	**db.adminCommand('listDatabases')**
use <db>	**db = db.getSiblingDB('<db>')**
show collections	**db.getCollectionNames()**
show users	**db.getUsers()**
show roles	**db.getRoles({showBuiltinRoles: true})**
show log <logname>	**db.adminCommand({ 'getLog' : '<logname>' })**
show logs	**db.adminCommand({ 'getLog' : '*' })**
it	**cursor = db.collection.find() if (cursor.hasNext()){ cursor.next();**

db.collection.insertOne()

Inserts a document into a collection.

The insertOne() method has the following syntax:

```
db.collection.insertOne(
  <document>,
  {
     writeConcern: <document>
  }
)
```

Example:

```
 db.employees.insertOne({
   Employee_id : 100,
    First_name :     "Steven",
     Last_name : "King",
    Job : "AD_PRES",
   Salary: 24000
      }
```

db.collection.insertMany()

```
db.collection.insertMany(
   [ <document 1> , <document 2>, ... ],
   {
      writeConcern: <document>,
      ordered: <boolean>
   })
```

Example:

```
try {
   db.products.insertMany( [
      { item: "card", qty: 15 },
      { item: "envelope", qty: 20 },
      { item: "stamps" , qty: 30 }
   ] );
```

db.collection.update()

db.collection.update(**query**, **update**, **options**)

```
db.collection.update(
  <query>,
  <update>,
  {
    upsert: <boolean>,
    multi: <boolean>,
    writeConcern: <document>,
    collation: <document>,
    arrayFilters: [ <filterdocument1>, ... ]
  }
)
```

Example:

```
db.books.update(
  { _id: 1 },
  {
    $inc: { stock: 5 },
    $set: {
      item: "ABC123",
      "info.publisher": "2222",
      tags: [ "software" ],
      "ratings.1": { by: "xyz", rating: 3 }
    }
  }
)
```

Copy collections

```
db.source.copyTo(target)
```

Copy collections

```
db.source.copyTo(target)
```

Clone DataBase

```
db.adminCommand({
  copydb: 1,
  fromdb: "test",
  todb: "records"
})
```

db.collection.delete()

MongoDB provides three methods for deleting documents:

- db.collection.deleteOne()
- db.collection.deleteMany()
- db.collection.remove()

```
db.collection.deleteMany(
  <filter>,
  {
    writeConcern: <document>,
    collation: <document>
  }
)
 db.collection.deleteOne(
   <filter>,
   {
     writeConcern: <document>,
     collation: <document>
   }
 )
```

Example:

```
db.inventory.deleteMany({ status : "A" })

db.inventory.deleteOne( { status: "D" } )

db. inventory.remove( { } )

db.products.remove( { qty: { $gt: 20 } } )
```

$TEXT

$text performs a text search on the content of the fields indexed with a text index.
A $text expression has the following syntax:

```
{  $text:   {
      $search: <string>,
      $language: <string>,
      $caseSensitive: <boolean>,
      $diacriticSensitive: <boolean>
    } }
```

$TEXT

```
db.articles.insert(
  [
    { _id: 1, subject: "coffee", author: "xyz", views: 50 },
    { _id: 2, subject: "Coffee Shopping", author: "efg", views: 5 },
    { _id: 3, subject: "Baking a cake", author: "abc", views: 90  },
    { _id: 4, subject: "baking", author: "xyz", views: 100 },
    { _id: 5, subject: "Café Con Leche", author: "abc", views: 200 },
    { _id: 6, subject: "Сырники", author: "jkl", views: 80 },
    { _id: 7, subject: "coffee and cream", author: "efg", views: 10 },
    { _id: 8, subject: "Cafe con Leche", author: "xyz", views: 10 }
  ]
)
```

db.articles.createIndex({ subject: "text" })

db.articles.find({ $text: { $search: "coffee" } })

$sort

Sorts all input documents and returns them to the pipeline in sorted order.

Sort order to 1 or -1 to specify an ascending or descending sort

```
db.users.aggregate(
  [
    { $sort : { age : -1, posts: 1 } }
  ]
)
```

$lookup

```
{
  $lookup:
    {
      from: <collection to join>,
      localField: <field from the input documents>,
      foreignField: <field from the documents of the "from" collection>,
      as: <output array field>
    }
}
```

$lookup

SQL

```
SELECT *, <output
 field> FROM collection WHERE <output
 field> IN (SELECT * FROM <collection to join>
WHERE <foreignField>= <collection.localField>);
```

MongoDb

```
{
  $lookup:
   {
     from: <collection to join>,
     let: { <var_1>: <expression>, …, <var_n>: <expression> },
     pipeline: [ <pipeline to execute on the collection to join> ],
     as: <output array field>
   }
}
```

$count (aggregation)

- db.collection.aggregate([
-   { $group: { _id: null, myCount: { $sum: 1 } } },
-   { $project: { _id: 0 } }
-])

A collection named `scores` has the following documents:

```
{ "_id" : 1, "subject" : "History", "score" : 88 }
{ "_id" : 2, "subject" : "History", "score" : 92 }
{ "_id" : 3, "subject" : "History", "score" : 97 }
{ "_id" : 4, "subject" : "History", "score" : 71 }
{ "_id" : 5, "subject" : "History", "score" : 79 }
{ "_id" : 6, "subject" : "History", "score" : 83 }
```

```
db.scores.aggregate(  [
   {
     $match: {
       score: {
         $gt: 80
       }
     }
   },
   {
     $count: "passing_scores"    }  ])
```

$match

Filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.

```
db.articles.aggregate(
   [ { $match : { author : "dave" } } ]
);
```

SQL vs MONGO

SQL Terms, Functions, and Concepts	MongoDB Aggregation Operators
WHERE	`$match`
GROUP BY	`$group`
HAVING	`$match`
SELECT	`$project`
ORDER BY	`$sort`
LIMIT	`$limit`
SUM()	`$sum`
COUNT()	`$sum` `$sortByCount`
join	`$lookup`

Interacting with cursors

```
var people = db.people.find();
while (people.hasNext())
{print(tojson(people.next()));}

db.people.find().forEach(function(person)
{print(person.name);});

var array = db.people.find().map(function(person) {
return person.name;
});
```

Grouping with $group

$group allows us to group a collection according to criteria.

We can group by fields that exist in the data, or we can group by expressions that create new fields.

db.people.aggregate(

{$group: {_id: 1}})

____

db.people.aggregate(

{$group: {_id: '$name'}})

____

Grouping by multiple fields

db.people.aggregate({$group: {

_id: {name: '$name',age: '$age'}}}

)

$EXIST

We can use exists to filter on the existence of non-existence of a field. We might find all the breakfasts with eggs:

```
db.breakfast.find({
eggs: {
$exists: true
}
})
```

$WHERE

We can even filter using an arbitrary JavaScript expression using $where. This will allow us to compare two fields in a single document.

```
db.sandwiches.find({
$where: "this.jam && this.peanutButter &&
this.jam > this.peanutButter"
})
```

Pivot Data

```
{ "_id" : 8751, "title" : "The Banquet", "author" : "Dante", "copies" : 2 }
{ "_id" : 8752, "title" : "Divine Comedy", "author" : "Dante", "copies" : 1 }
{ "_id" : 8645, "title" : "Eclogues", "author" : "Dante", "copies" : 2 }
{ "_id" : 7000, "title" : "The Odyssey", "author" : "Homer", "copies" : 10 }
{ "_id" : 7020, "title" : "Iliad", "author" : "Homer", "copies" : 10 }
```

`db.books.aggregate(   [    { $group : { _id : "$author", books: { $push: "$title" } } }   ])`

```
{ "_id" : "Homer", "books" : [ "The Odyssey", "Iliad" ] }
{ "_id" : "Dante", "books" : [ "The Banquet", "Divine Comedy", "Eclogues" ] }
```

Pivot Data

```
{ "_id" : 8751, "title" : "The Banquet", "author" : "Dante", "copies" : 2 }
{ "_id" : 8752, "title" : "Divine Comedy", "author" : "Dante", "copies" : 1 }
{ "_id" : 8645, "title" : "Eclogues", "author" : "Dante", "copies" : 2 }
{ "_id" : 7000, "title" : "The Odyssey", "author" : "Homer", "copies" : 10 }
{ "_id" : 7020, "title" : "Iliad", "author" : "Homer", "copies" : 10 }
```

```
db.books.aggregate(   [     { $group : { _id : "$author", books: { $push: "$title" } } }   ])
```

```
{ "_id" : "Homer", "books" : [ "The Odyssey", "Iliad" ] }
{ "_id" : "Dante", "books" : [ "The Banquet", "Divine Comedy", "Eclogues" ] }
```

$ROOT

The $$ROOT variable contains the source documents for the group.

If you'd like to just pass them through unmodified, you can do this by $pushing $$ROOT into the output from the group.

```
db.entrycodes.aggregate([{
$group: {
_id: "$email",
name: "$firstName"
count: {$sum: 1},
entries: { $push: "$$ROOT" }}}])
```

$ROOT

The $$ROOT variable contains the source documents for the group.

If you'd like to just pass them through unmodified, you can do this by $pushing $$ROOT into the output from the group.

```
db.entrycodes.aggregate([{
$group: {
_id: "$email",
name: "$firstName"
count: {$sum: 1},
entries: { $push: "$$ROOT" }}}])
```

$PROJECT

The Project takes a document that can specify the inclusion of fields, the suppression of the `_id` field,
the addition of new fields, and the resetting of the values of existing fields.
Alternatively, you may specify the *exclusion* of fields.

$PROJECT

```
{
  "_id" : 1,
  title: "abc123",
  isbn: "0001122223334",
  author: { last: "zzz", first: "aaa" },
  copies: 5}

db.books.aggregate( [ { $project : { title : 1 , author : 1 } } ] )

{ "_id" : 1, "title" : "abc123", "author" : { "last" : "zzz", "first" : "aaa" }
}
```

Aggregation by date

```
db.competitionentries.aggregate({
$project: {
year: {$year: date},
month: {$month: date},
dayOfMonth: {$dayOfMonth: date}}},
{
$group: {_id: {
year: '$year',
month: '$month',
dayOfMonth: '$dayOfMonth'},
count: {
$sum: 1}}})
```