MONGODB CHEAT SHEET

The document database with scalability and flexibility.

MongoDB version: **5.0.9+** - Date: **June 2022**

MAIN CONCEPTS

Document : A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are **very** similar to JSON objects. Values may include other documents, arrays, and arrays of documents.

```
{
   name: "John",
   age: 77,
   nets: [ "expert", "developer", "superhero" ],
   lastExp: { name: "Full-Stack Lead Developer",
duration: 10.5 }
}
```

Note : each document requires a unique **_id** field (uses the ObjectId BSON type) that acts as a primary key. The MongoDB driver generates it if omitted.

Collection : A grouping of MongoDB documents. A collection is the equivalent of an RDBMS table. Collections do not enforce a schema by default. Documents within a collection can have different fields but should have a similar or related purpose.

BSON TYPES

Each value of a field/value pair is recorded as a BSON type. BSON is a binary serialization format. The most used and useful BSON types in MongoDB are (each BSON type has both integer and string identifiers) :

Double : 1 "double" ; **String** : 2 "string" ; **Object** : 3 "object" ; **Array** : 4 "array" ; **ObjectId** : 7 "objectId" ; **Boolean** : 8 "bool" ; **Date** : 9 "date" ; **Null** : 10 "null" ; **32-bit integer** : 16 "int" ; **64-bit integer** : 18 "long" ; **Decimal128** : 19 "decimal"

Note : **Timestamp** data type is for internal MongoDB use. You will want to use the BSON date type.

TIME SERIES COLLECTIONS

Time series collections came with version 5.0 of MongoDB. They efficiently store sequences of measurements over a period of time. Time series data is any data that is collected over time and is uniquely identified by one or more unchanging parameters. Consider using them for measurements storage!

CRUD OPERATIONS

Create

```
db.collection.insertOne()
db.collection.insertMany()
```

```
db.users.insertOne(
   { name: "John", surname: "Doe" }
)
db.users.insertMany(
   { name: "John", surname: "Doe" },
   { name: "Jean", surname: "Dudule" }
)
```

Note : If the collection does not currently exist, insert operations will create the collection.

Find (see Query operators)

```
db.collection.find()
```

```
db.users.find(
   { age: { $gt: 18 } }
)
```

Update (see Query/Update operators)

```
db.collection.updateOne()
db.collection.updateMany()
db.collection.replaceOne()
```

```
db.users.updateMany(
   { age: { $gt: 60 } },
   { $set: { retirement: true } }
)
```

See **findAndModify** to modify and return a single document. By default, the returned document does not include the modifications made on the update.

Delete

```
db.collection.deleteOne()
db.collection.deleteMany()
```

```
db.users.deleteMany(
   { retirement: true }
)
```

MONGODB CHEAT SHEET

QUERY OPERATORS

Comparison

- **$eq**/**$ne** : Equals/Not equals
- **$gt**/**$gte**/**$lt**/**$lte** : Greater than/g.t. equals/ ...
- **$in**/**$nin** : Any/None of values specified in an array

Logical

- **$and**/**$nor**/**$or** : Joins clauses with AND/NOR/OR
- **$not** : Invert the effect of a query expression

Element

- **$exists** : Documents that have the specified field
- **$type** : Field is of the specified type.

Array

- **$all** : Arrays that contain all elements
- **$elemMatch** : Element in the array field matches all the specified $elemMatch conditions.
- **$size** : Array size

More operators : $regex, $text (text search), $where (javascript expression), geometries operators, ...

UPDATE OPERATORS

Fields

- **$currentDate** : Sets value to current date
- **$inc**/**$mul** : Increments/Multiplies value by specified amount
- **$min**/**$max** : Updates if the specified value is less/more than existing
- **$set**/**$unset** : Sets/Removes the value of a field in a document.

More operators : $rename, $setOnInsert.

Array

- **$/$[]/$[<identifier>]** : Update first/all/all elements that match the array filters
- **$push**/**$addToSet** : Adds/Adds if not exists
- **$pop** : Removes first or last item
- **$pull**/**$pullAll** : Removes all/all from an array

Modifiers : $each, $position, $slice, $sort

INDEXES

Appropriate indexes (and index orderings) are essential for efficient queries execution.

```
db.collection.createIndex()
```

```
db.users.createIndex( { age: -1 } )
db.users.createIndex( { name: 1, surname: 1 } )
```

Note: _id field is a unique index for every collection.

AGGREGATION PIPELINE

Aggregation operations process data records, can perform a variety of operations, and return computed results. Documents pass through the stages of the pipeline in sequence.

```
db.collection.aggregate([ { operation: {  } }, ... ])
```

```
db.users.aggregate([
   { $group: { _id: "$nets", age: { $avg: "$age"} } },
   { $match: { age: { $gt: 35 } } }
])
```

Note : map-reduce function is an alternative way for aggregation. You will want to use aggregation pipeline.

WANNA BECOME EXPERT ?

- **Reporting** : mongostat, and many more utilities
- **Replication** : redundancy and high availability
- **Sharding** : data distribution and horizontal scaling for high query rates and exhausted CPU
- **Bulk Write** : for multiple ordered rows loads
- **Change Streams** : access real-time data changes

TOOLS AND DOCUMENTATION

Studio3T : Useful and powerful tool to build queries through a solid UI. studio3t.com/download .

Official documentation : A "must-have" favorite. Manual : docs.mongodb.com/manual and reference : docs.mongodb.com/manual/reference