

Reference Cards for MongoDB

What is MongoDB?

MongoDB is an open-source, general purpose database.

Instead of storing data in rows and columns as a relational database does, MongoDB uses a document data model, and stores a binary form of JSON documents called BSON. Documents contain one or more fields, and each field contains a value of a specific data type, including arrays and binary data. Documents are stored in collections, and collections are stored in databases. It may be helpful to think of documents as roughly equivalent to rows in a relational database; fields as equivalent to columns; and collections as tables. There are no fixed schemas in MongoDB, so documents can vary in structure and can be adapted dynamically.

MongoDB provides full index support, including secondary, compound, and geospatial indexes. MongoDB also features a rich query language, atomic update modifiers, text search, the Aggregation Framework for analytics similar to SQL GROUP BY operations, and MapReduce for complex in-place data analysis.

Built-in replication with automated failover provides high availability. Auto-sharding enables horizontal scaling for large deployments. MongoDB also provides native, idiomatic drivers for all popular programming languages and frameworks to make development natural.

Queries

Reference Cards for MongoDB

Queries

Queries and What They Match

<pre>{a: 10}</pre>	Docs where a is 10 , or an array containing the value 10 .
<pre>{a: 10, b: "hello"}</pre>	Docs where a is 10 and b is "hello" .
<pre>{a: {\$gt: 10}}</pre>	Docs where a is greater than 10 . Also available: \$lt (<), \$gte (>=), \$lte (<=), and \$ne (!=).
<pre>{a: {\$in: [10, "hello"]}}</pre>	Docs where a is either 10 or "hello" .
<pre>{a: {\$all: [10, "hello"]}}</pre>	Docs where a is an array containing both 10 and "hello" .
<pre>{"a.b": 10}</pre>	Docs where a is an embedded document with b equal to 10 .
<pre>{a: {\$elemMatch: {b: 1, c: 2}}}</pre>	Docs where a is an array that contains an element with both b equal to 1 and c equal to 2 .
<pre>{\$or: [{a: 1}, {b: 2}]}</pre>	Docs where a is 1 or b is 2 .
<pre>{a: /^m/}</pre>	Docs where a begins with the letter m . One can also use the regex operator: {a: {\$regex: "m"}} .
<pre>{a: {\$mod: [10, 1]}}</pre>	Docs where a mod 10 is 1 .
<pre>{a: {\$type: 2}}</pre>	Docs where a is a string. See bsonspec.org for more.
<pre>{ \$text: { \$search: "hello" } }</pre>	Docs that contain "hello" on a text search. Requires a text index.

Queries and What They Match (continued)

<pre>{a: {\$near: {\$geometry: { type: "Point", coordinates: [-73.98, 40.75] }} }</pre>	Docs sorted in order of nearest to farthest from the given coordinates. <i>For geospatial queries one can also use \$geoWithin and \$geoIntersects operators.</i>
---	---

Not Indexable Queries

The following queries cannot use indexes. These query forms should normally be accompanied by at least one other query term which does use an index.

<pre>{a: {\$nin: [10, "hello"]}}</pre>	Docs where a is anything but 10 or "hello" .
<pre>{a: {\$size: 3}}</pre>	Docs where a is an array with exactly 3 elements.
<pre>{a: {\$exists: true}}</pre>	Docs containing an a field.
<pre>{a: /foo.*bar/}</pre>	Docs where a matches the regular expression foo.*bar .
<pre>{a: {\$not: {\$type: 2}}}</pre>	Docs where a is not a string. \$not negates any of the other query operators.

For More Information

<http://docs.mongodb.org/manual/tutorial/query-documents/>

<http://docs.mongodb.org/manual/reference/operator/query/>

Updates

Reference Cards for MongoDB

Updates

Field Update Modifiers

<pre>{ \$inc: { a: 2 } }</pre>	Increments a by 2 .
<pre>{ \$set: { a: 5 } }</pre>	Sets a to the value 5 .
<pre>{ \$unset: { a: 1 } }</pre>	Removes the a field from the document.
<pre>{ \$max: { a: 10 } }</pre>	Sets a to the greater value, either current or 10 . If a does not exist sets a to 10 .
<pre>{ \$min: { a: -10 } }</pre>	Sets a to the lower value, either current or -10 . If a does not exist sets a to -10 .
<pre>{ \$mul: { a: 2 } }</pre>	Sets a to the product of the current value of a and 2 . If a does not exist sets a to 0 .
<pre>{ \$rename: { a: "b" } }</pre>	Renames field a to b .
<pre>{ \$setOnInsert: { a: 1 } }, { upsert: true }</pre>	Sets field a to 1 in case of upsert operation.
<pre>{ \$currentDate: { a: { \$type: "date" } }</pre>	Sets field a with the current date. <i>\$currentDate can be specified as date or timestamp. Note that as of 3.0, date and timestamp are not equivalent for sort order.</i>
<pre>{ \$bit: { a: { and: 7 } } }</pre>	Performs the bitwise and operation over a field. If a is 12: 1100 0111 ----- 0100 Supports and xor or bitwise operators.

Array Update Operators

<pre>{ \$push: { a: 1 } }</pre>	Appends the value 1 to the array a .
<pre>{ \$push: { a: { \$each: [1, 2] } } }</pre>	Appends both 1 and 2 to the array a .
<pre>{ \$addToSet: { a: 1 } }</pre>	Appends the value 1 to the array a (if the value doesn't already exist).
<pre>{ \$addToSet: { a: { \$each: [1, 2] } } }</pre>	Appends both 1 and 2 to the array a (if they don't already exist).
<pre>{ \$pop: { a: 1 } }</pre>	Removes the last element from the array a .
<pre>{ \$pop: { a: -1 } }</pre>	Removes the first element from the array a .
<pre>{ \$pull: { a: 5 } }</pre>	Removes all occurrences of 5 from the array a .
<pre>{ \$pullAll: { a: [5, 6] } }</pre>	Removes multiple occurrences of 5 or 6 from the array a .

For More Information

<http://docs.mongodb.org/manual/reference/operator/update/>

Aggregation Framework

Reference Cards for MongoDB

Aggregation Framework

The aggregation pipeline is a framework for data aggregation modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into aggregated results. Pipeline stages appear in an array. Documents pass through the stages in sequence. Structure an aggregation pipeline using the following syntax:

```
db.collection.aggregate( [ { <stage> }, ... ] )
```

Aggregation Framework Stages

<pre>{ \$match: { a: 10 } }</pre>	Passes only documents where a is 10 .	Similar to <code>find()</code>
<pre>{ \$project: { a: 1, _id: 0 } }</pre>	Reshapes each document to include only field a , removing others.	Similar to <code>find()</code> projection
<pre>{ \$project: { new_a: "\$a" } }</pre>	Reshapes each document to include only _id and the new field new_a with the value of a .	<pre>{ a: 1 } => { new_a: 1 }</pre>
<pre>{ \$project: { a: { \$add: ["\$a", "\$b"] } } }</pre>	Reshapes each document to include only _id and field a , set to the sum of a and b .	<pre>{ a: 1, b: 10 } => { a: 11 }</pre>
<pre>{ \$project: { stats: { value: "\$a", fraction: { \$divide: ["\$a", "\$b"] } } } }</pre>	Reshapes each document to contain only _id and the new field stats which contains embedded fields value , set to the value of a , and fraction , set to the value of a divided by b .	<pre>{ a: 10, b: 2 } => { stats: { value: 10, fraction: 5 } }</pre>
<pre>{ \$group: { _id: "\$a", count: { \$sum: 1 } } }</pre>	Groups documents by field a and computes the count of each distinct a value.	<pre>{ a: "hello" }, { a: "goodbye" }, { a: "hello" } => { _id: "hello", count: 2 }, { _id: "goodbye", count: 1 }</pre>

Aggregation Framework Stages (continued)

<pre>{ \$group: { _id: "\$a", names: { \$addToSet: "\$b" } } }</pre>	Groups documents by field a with new field names consisting of a set of b values.	<pre>{a:1, b:"John"}, {a:1, b:"Mary"} => { _id:1, names: ["John", "Mary"] }</pre>
<pre>{ \$unwind: "\$a" }</pre>	Deconstructs array field a into individual documents of each element.	<pre>{a: [2,3,4]} => {a:2}, {a:3}, {a:4}</pre>
<pre>{ \$limit: 10 }</pre>	Limits the set of documents to 10 , passing the first 10 documents.	
<pre>{ \$sort: {a:1} }</pre>	Sorts results by field a ascending.	
<pre>{ \$skip: 10 }</pre>	Skips the first 10 documents and passes the rest.	
<pre>{ \$out: "myResults" }</pre>	Writes resulting documents of the pipeline into the collection "myResults" .	Must be the last stage of the pipeline.

For More Information

<http://docs.mongodb.org/master/core/aggregation-introduction/>

Indexing

Reference Cards for MongoDB

Indexing

Index Creation Syntax

```
db.coll.createIndex(<key_pattern>, <options>)
```

Creates an index on collection `coll` with given key pattern and options.

Indexing Key Patterns

<code>{a:1}</code>	Simple index on field a .
<code>{a:1, b:-1}</code>	Compound index with a ascending and b descending.
<code>{"a.b": 1}</code>	Ascending index on embedded field " a.b ".
<code>{a: "text"}</code>	Text index on field a . A collection can have at most one text index.
<code>{a: "2dsphere"}</code>	Geospatial index where the a field stores GeoJSON data. See documentation for valid GeoJSON formatting.
<code>{a: "hashed"}</code>	Hashed index on field a . Generally used with hash-based sharding.

Index options

<pre>{unique: true}</pre>	Creates an index that requires all values of the indexed field to be unique.
<pre>{background: true}</pre>	Creates this index in the background; useful when you need to minimize index creation performance impact.
<pre>{name: "foo"}</pre>	Specifies a custom name for this index. If not specified, the name will be derived from the key pattern.
<pre>{sparse: true}</pre>	Creates entries in the index only for documents having the index key.
<pre>{expireAfterSeconds: 360}</pre>	Creates a time to live (TTL) index on the index key. This will force the system to drop the document after 3600 seconds expire. <i>Only works on keys of date type.</i>
<pre>{default_language: 'portuguese'}</pre>	Used with text indexes to define the default language used for stop words and stemming.

Examples

<pre>db.products.createIndex({ 'supplier': 1 }, { unique: true })</pre>	Creates ascending index on supplier assuring unique values.
<pre>db.products.createIndex({ 'description': 'text', 'default_language': 'spanish' })</pre>	Creates text index on description key using Spanish for stemming.
<pre>db.products.createIndex({ 'regions': 1 }, { sparse: true })</pre>	Creates ascending sparse index on regions key. If regions is an array – e.g., <code>regions: ['EMEA', 'NA', 'LATAM']</code> – will create a multikey index.
<pre>db.stores.createIndex({ location: "2dsphere" })</pre>	Creates a 2dsphere geospatial index on location key .

Administration

<code>db.products.getIndexes()</code>	Gets a list of all indexes on the products collection.
<code>db.products.reIndex()</code>	Rebuilds all indexes on this collection.
<code>db.products.dropIndex({x: 1, y: -1})</code>	Drops the index with key pattern {x: 1, y: -1} . Use db.products.dropIndex('index_a') to drop index named index_a . Use db.products.dropIndexes() to drop all indexes on the products collection.

For More Information

<http://docs.mongodb.org/master/core/indexes-introduction/>

Replication

Reference Cards for MongoDB

Replication

What is a Majority?

If your set consists of...

1 server, 1 server is a majority.

2 servers, 2 servers are a majority.

3 servers, 2 servers are a majority.

4 servers, 3 servers are a majority.

...

Setup

To initialize a three-node replica set including one arbiter, start three **mongod** instances, each using the **--replSet** flag followed by a name for the replica set. For example:

```
mongod --replSet cluster-foo
```

Next, connect to one of the **mongod** instances and run the following:

```
rs.initiate()  
rs.add("host2:27017")  
rs.add("host3:27017", true)
```

rs.add() can also accept a document parameter, such as **rs.add({"_id": 4, "host": "host4:27017"})**. The document can contain the following options:

<code>priority: n</code>	Members will be elected primary in order of priority, if possible. Higher values make a member more eligible to become a primary. n=0 means this member will never be a primary.
<code>votes: n</code>	Assigns a member voting privileges (n=1 for voting, n=0 for nonvoting).
<code>slaveDelay: n</code>	This member will always be a secondary and will lag n seconds behind the master.

Setup (continued)

<code>arbiterOnly: true</code>	This member will be an arbiter.
<code>hidden: true</code>	Do not show this member in isMaster output. Use this option to hide this member from clients.
<code>tags: [...]</code>	Member location description. See docs.mongodb.org/manual/data-center-awareness .

Administration

<code>rs.initiate()</code>	Creates a new replica set with one member.
<code>rs.add("host:port")</code>	Adds a member.
<code>rs.addArb("host:port")</code>	Adds an arbiter.
<code>rs.remove("host:port")</code>	Removes a member.
<code>rs.status()</code>	Returns a document with information about the state of the replica set.
<code>rs.conf()</code>	Returns the replica set configuration document.
<code>rs.reconfig(newConfig)</code>	Re-configures a replica set by applying a new replica set configuration object.
<code>rs.isMaster()</code>	Indicates which member is primary.
<code>rs.stepDown(n)</code>	Forces the primary to become a secondary for n seconds, during which time an election can take place.

Administration (continued)

<code>rs.freeze(n)</code>	Prevents the current member from seeking election as primary for n seconds. n=0 means <i>unfreeze</i> .
<code>rs.printSlaveReplicationInfo()</code>	Prints a report of the status of the replica set from the perspective of the secondaries.

For More Information

<http://docs.mongodb.org/master/core/replication-introduction/>

Sharding

Reference Cards for MongoDB

Sharding

<pre>sh.enableSharding('products')</pre>	Enables sharding on products database.
<pre>sh.shardCollection('products.catalog', { sku:1, brand:1})</pre>	Shards collection catalog of products database with shard key consisting of the sku and brand fields.
<pre>sh.status()</pre>	Prints a formatted report of the sharding configuration and the information regarding existing chunks in a sharded cluster.
<pre>sh.addShard('REPLICA1/ host:27017')</pre>	Adds existing replica set REPLICA1 as a shard to the cluster.

For More Information

<http://docs.mongodb.org/master/core/sharding-introduction/>

Mapping SQL to MongoDB

Reference Cards for MongoDB

Mapping SQL to MongoDB

Converting to MongoDB Terms

MYSQL Executable	Oracle Executable	MongoDB Executable
mysqld	oracle	mongod
mysql	sqlplus	mongo

SQL Term	MongoDB Term
database (schema)	database
table	collection
index	index
row	document
column	field
joining	linking & embedding
partition	shard

Queries and other operations in MongoDB are represented as documents passed to **find()** and other methods. Below are examples of SQL statements and the analogous statements in MongoDB JavaScript shell syntax.

SQL	MongoDB
CREATE TABLE users (name VARCHAR(128), age NUMBER)	db.createCollection("users")
INSERT INTO users VALUES ('Bob', 32)	db.users.insert({name: "Bob", age: 32})
SELECT * FROM users	db.users.find()
SELECT name, age FROM users	db.users.find({}, {name: 1, age: 1, _id:0})
SELECT name, age FROM users WHERE age = 33	db.users.find({age: 33}, {name: 1, age: 1, _id:0})
SELECT * FROM users WHERE age > 33	db.users.find({age: {\$gt: 33}})
SELECT * FROM users WHERE age <= 33	db.users.find({age: {\$lte: 33}})
SELECT * FROM users WHERE age > 33 AND age < 40	db.users.find({age: {\$gt: 33, \$lt: 40}})
SELECT * FROM users WHERE age = 32 AND name = 'Bob'	db.users.find({age: 32, name: "Bob"})
SELECT * FROM users WHERE age = 33 OR name = 'Bob'	db.users.find({\$or:[{age:33}, {name:"Bob"}]})
SELECT * FROM users WHERE age = 33 ORDER BY name ASC	db.users.find({age: 33}).sort({name: 1})
SELECT * FROM users ORDER BY name DESC	db.users.find().sort({name: -1})
SELECT * FROM users WHERE name LIKE '%Joe%'	db.users.find({name: /Joe/})

SQL	MongoDB
<code>SELECT * FROM users WHERE name LIKE 'Joe%'</code>	<code>db.users.find({name: /^Joe/})</code>
<code>SELECT * FROM users LIMIT 10 SKIP 20</code>	<code>db.users.find().skip(20).limit(10)</code>
<code>SELECT * FROM users LIMIT 1</code>	<code>db.users.findOne()</code>
<code>SELECT DISTINCT name FROM users</code>	<code>db.users.distinct("name")</code>
<code>SELECT COUNT(*) FROM users</code>	<code>db.users.count()</code>
<code>SELECT COUNT(*) FROM users WHERE AGE > 30</code>	<code>db.users.find({age: {\$gt: 30}}).count()</code>
<code>SELECT COUNT(AGE) FROM users</code>	<code>db.users.find({age: {\$exists: true}}).count()</code>
<code>UPDATE users SET age = 33 WHERE name = 'Bob'</code>	<code>db.users.update({name: "Bob"}, {set: {age: 33}}, {multi: true})</code>
<code>UPDATE users SET age = age + 2 WHERE name = 'Bob'</code>	<code>db.users.update({name: "Bob"}, {inc: {age: 2}}, {multi: true})</code>
<code>DELETE FROM users WHERE name = 'Bob'</code>	<code>db.users.remove({name: "Bob"})</code>
<code>CREATE INDEX ON users (name ASC)</code>	<code>db.users.createIndex({name: 1})</code>
<code>CREATE INDEX ON users (name ASC, age DESC)</code>	<code>db.users.createIndex({name: 1, age: -1})</code>
<code>EXPLAIN SELECT * FROM users WHERE age = 32</code>	<code>db.users.find({age: 32}).explain() (db.users.explain().find({age: 32}) for 3.0)</code>
<code>SELECT age, SUM(1) AS counter FROM users GROUP BY age</code>	<code>db.users.aggregate([{\$group: {'_id': '\$age', counter: {\$sum:1}} }])</code>

SQL	MongoDB
<pre>SELECT age, SUM(1) AS counter FROM users WHERE country = "US" GROUP BY age</pre>	<pre>db.users.aggregate([{\$match: {country: 'US'} }, {\$group: {'_id': '\$age', counter: {\$sum:1}} }])</pre>
<pre>SELECT age AS "how_old" FROM users</pre>	<pre>db.users.aggregate([{\$project: {"how_old": "\$age"}}])</pre>

For More Information

<http://docs.mongodb.org/manual/reference/sql-comparison/>

Resources

Learn

Downloads - mongodb.org/downloads

MongoDB Enterprise Advanced - mongodb.com/enterprise

MongoDB Manual - docs.mongodb.org

Free Online Education - university.mongodb.com

Presentations - mongodb.com/presentations

In-person Training - university.mongodb.com/training

Support

Stack Overflow - stackoverflow.com/questions/tagged/mongodb

Google Group - groups.google.com/group/mongodb-user

Bug Tracking - jira.mongodb.org

Commercial Support - mongodb.com/support

Community

MongoDB User Groups (MUGs) - mongodb.com/user-groups

MongoDB Events - mongodb.com/events

Social

Twitter - [@MongoDB](https://twitter.com/MongoDB), [@MongoDB_Inc](https://twitter.com/MongoDB_Inc)

Facebook - facebook.com/mongodb

LinkedIn - linkedin.com/groups/MongoDB-2340731

Contact

Contact MongoDB - mongodb.com/contact

