MongoDB Cheat Sheet

A quick reference guide to MongoDB commands, syntax, and operations for developers and database administrators.

Basic Operations

Connecting to MongoDB

Connect to a MongoDB instance using the `mongo` shell:

```
mongo
```

To connect to a specific host and port:

```
mongo --host <hostname> --port <port>
```

Connect to a MongoDB instance with authentication:

```
mongo -u <username> -p <password> --authenticationDatabase <db>
```

Database Operations

`show dbs`	Lists all the databases available on the server.
`use <database_name>`	Switches to the specified database. Creates the database if it doesn't exist.
`db`	Displays the current database you are using.
`db.dropDatabase()`	Deletes the current database.

Collection Operations

`show collections`	Lists all collections in the current database.
`db.createCollection('<collection_name>')`	Creates a new collection in the current database.
`db.<collection_name>.drop()`	Deletes the specified collection.

CRUD Operations

Insert Operations

Insert a single document:

```
db.<collection_name>.insertOne({ key: 'value' })
```

Insert multiple documents:

```
db.<collection_name>.insertMany([{ key: 'value' }, { key: 'value2' }])
```

Query Operations

Find all documents in a collection:

```
db.<collection_name>.find()
```

Find documents with a specific condition:

```
db.<collection_name>.find({ key: 'value' })
```

Find a single document that matches the condition:

```
db.<collection_name>.findOne({ key: 'value' })
```

Using operators:

```
db.<collection_name>.find({age: {$gt: 25}})
```

Update Operations

Update a single document:

```
db.<collection_name>.updateOne({ query }, { $set: { key: 'new_value' } })
```

Update multiple documents:

```
db.<collection_name>.updateMany({ query }, { $set: { key: 'new_value' } })
```

Replace a single document:

```
db.<collection_name>.replaceOne({ query }, { key: 'new_value' })
```

Delete Operations

Delete a single document:

```
db.<collection_name>.deleteOne({ query })
```

Delete multiple documents:

```
db.<collection_name>.deleteMany({ query })
```

Querying with Operators

Comparison Operators

`$eq`	Matches values that are equal to a specified value. `db.inventory.find( { qty: { $eq: 20 } } )`
`$gt`	Matches values that are greater than a specified value. `db.inventory.find( { qty: { $gt: 20 } } )`
`$gte`	Matches values that are greater than or equal to a specified value. `db.inventory.find( { qty: { $gte: 20 } } )`
`$lt`	Matches values that are less than a specified value. `db.inventory.find( { qty: { $lt: 20 } } )`
`$lte`	Matches values that are less than or equal to a specified value. `db.inventory.find( { qty: { $lte: 20 } } )`
`$ne`	Matches all values that are not equal to a specified value. `db.inventory.find( { qty: { $ne: 20 } } )`
`$in`	Matches any of the values specified in an array. `db.inventory.find( { qty: { $in: [ 5, 15 ] } } )`
`$nin`	Matches none of the values specified in an array. `db.inventory.find( { qty: { $nin: [ 5, 15 ] } } )`

Logical Operators

`$and`	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses. `db.inventory.find( { $and: [ { price: { $ne: 1.99 } }, { qty: { $lt: 20 } }, { sale: true } ] } )`
`$or`	Joins query clauses with a logical OR returns all documents that match the conditions of either clause. `db.inventory.find( { $or: [ { qty: { $lt: 20 } }, { price: { $gt: 10 } } ] } )`
`$not`	Inverts the effect of a query expression and returns documents that do not match the query expression. `db.inventory.find( { price: { $not: { $gt: 1.99 } } } )`
`$nor`	Joins query clauses with a logical NOR returns all documents that fail to match both clauses. `db.inventory.find( { $nor: [ { price: 1.99 }, { qty: { $lt: 20 } } ] } )`

Element Operators

`$exists`	Matches documents that have the specified field. `db.inventory.find( { size: { $exists: true } } )`
`$type`	Selects documents where values match a specified BSON type. `db.inventory.find( { qty: { $type: "number" } } )`

Indexes and Aggregation

Index Operations

Create an index on a field:

```
db.<collection_name>.createIndex({ field: 1 })
```

(1 for ascending, -1 for descending)

List all indexes for a collection:

```
db.<collection_name>.getIndexes()
```

Drop an index:

```
db.<collection_name>.dropIndex('<index_name>')
```

Aggregation Pipeline

An example aggregation pipeline:

```
db.<collection_name>.aggregate([
  { $match: { status: "A" } },
  { $group: { _id: "$cust_id", total: { $sum: "$amount" } } },
  { $sort: { total: -1 } }
])
```

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

`$group` : Groups documents that share the same _id.

`$sort` : Sorts all input documents and returns them to the pipeline in sorted order.

`$project` : Passes along the documents with the requested fields to the next stage in the pipeline. The specified fields can be existing fields or newly computed fields.

`$limit` : Limits the number of documents passed to the next stage in the pipeline.

`$skip` : Skips over the specified number of documents and passes the remaining documents to the next stage in the pipeline.