MongoDB Basics

- Connection methods (local, remote, with authentication)
- Database operations (creating, switching, viewing, dropping)
- Collection operations (creating, showing, dropping)

CRUD Operations

- Creating documents (insertOne, insertMany)
- Reading data with various query options and operators
- Updating documents (updateOne, updateMany, various update operators)
- Deleting documents (deleteOne, deleteMany)

Query Operations

- Comparison operators ($eq, $ne, $gt, $lt, etc.)
- Logical operators ($and, $or, $not, $nor)
- Element operators ($exists, $type)
- Evaluation operators ($regex, $text, $expr)
- Array operators ($all, $elemMatch, $size)

Cursor Methods

- Pagination with limit() and skip()
- Sorting results
- Counting and iteration techniques

Aggregation Framework

- Basic pipeline structure
- Common stages ($match, $group, $project, $sort, etc.)
- Advanced operations ($lookup, $unwind, $bucket)

Indexes

- Creating various index types (single field, compound, unique, etc.)
- Text indexes and geospatial indexes
- Managing and viewing indexes

Data Management

- Import/Export commands
- Backup/Restore procedures

Administration

- User management
- Database status monitoring
- Replica set operations
- Sharding commands

Schema Validation

- JSON Schema validation examples
- Validation rules and options

Performance & Optimization

- Query explanation and performance analysis
- Statistical tools and monitoring

MongoDB Cheatsheet

MongoDB Basics

Connection

Connect to local MongoDB instance

mongo

Connect to specific host and port

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

Connect with authentication

mongo --host <hostname> --port <port> -u <username> -p <password>
--authenticationDatabase <authDB>

Connection string format (for applications)

mongodb://[username:password@]hostname[:port][/database][?options]

Connection string examples

mongodb://localhost:27017/mydb

mongodb://user:pass@mongodb.example.com:27017/mydb

mongodb+srv://user:pass@cluster0.mongodb.net/mydb # DNS SRV connection (Atlas)

Database Operations

// Show all databases

```
show dbs

// Switch to database (creates it if it doesn't exist)
use <database_name>

// Show current database
db

// Drop current database
db.dropDatabase()
```

Collection Operations

```
// Show collections in current database
show collections

// Create collection explicitly
db.createCollection("collection_name")

// Create collection implicitly (by inserting a document)
db.collection_name.insertOne({key: "value"})

// Drop a collection
db.collection_name.drop()
```

CRUD Operations

Create (Insert)

```
// Insert a single document
db.collection_name.insertOne({
  name: "John Doe",
  age: 30,
  email: "john@example.com",
  created_at: new Date()
})

// Insert multiple documents
db.collection_name.insertMany([
  { name: "Jane Smith", age: 25, email: "jane@example.com" },
  { name: "Bob Johnson", age: 35, email: "bob@example.com" }
])

// Insert with a specific _id
db.collection_name.insertOne({
  _id: ObjectId("60eabc1d3a1cde1234567890"),
  name: "Custom ID Document"
})
```

Read (Query)

```
// Find all documents in a collection
```

```
db.collection_name.find()

// Find with pretty printing
db.collection_name.find().pretty()

// Find one document
db.collection_name.findOne()

// Find with condition
db.collection_name.find({ age: 30 })

// Find with multiple conditions (AND)
db.collection_name.find({ age: 30, name: "John Doe" })

// Find specific fields only (projection)
db.collection_name.find({ age: 30 }, { name: 1, email: 1, _id: 0 })

// Find with OR condition
db.collection_name.find({
  $or: [
    { age: 30 },
    { name: "John Doe" }
  ]
})
```

```
// Find with AND and OR combined
db.collection_name.find({
  status: "active",
  $or: [
    { age: { $lt: 30 } },
    { name: "John Doe" }
  ]
})

// Find with nested objects
db.collection_name.find({
  "address.city": "New York"
})

// Find documents with specific array element
db.collection_name.find({
  tags: "mongodb"
})

// Find with array of specific size
db.collection_name.find({
  tags: { $size: 3 }
})
```

```
// Find with specific array element at position
db.collection_name.find({
  "tags.0": "mongodb"
})

// Count documents
db.collection_name.countDocuments({ age: { $gt: 25 } })

// Distinct values
db.collection_name.distinct("age")
```

Update

```
// Update a single document
db.collection_name.updateOne(
  { name: "John Doe" },
  { $set: { age: 31, updated_at: new Date() } }
)

// Update multiple documents
db.collection_name.updateMany(
  { age: { $lt: 30 } },
  { $set: { status: "young" } }
)
```

```
// Replace entire document (except _id)
db.collection_name.replaceOne(
  { name: "John Doe" },
  { name: "John Doe Jr", age: 31, email: "johnjr@example.com" }
)

// Update with upsert (insert if doesn't exist)
db.collection_name.updateOne(
  { name: "New User" },
  { $set: { age: 25, status: "active" } },
  { upsert: true }
)

// Increment a field
db.collection_name.updateOne(
  { name: "John Doe" },
  { $inc: { age: 1, logins: 1 } }
)

// Multiply a field
db.collection_name.updateOne(
  { name: "John Doe" },
  { $mul: { score: 2 } }
```

```
)

// Remove a field
db.collection_name.updateOne(
  { name: "John Doe" },
  { $unset: { temporary_field: "" } }
)

// Rename a field
db.collection_name.updateMany(
  {},
  { $rename: { "name": "full_name" } }
)

// Add to array
db.collection_name.updateOne(
  { name: "John Doe" },
  { $push: { tags: "developer" } }
)

// Add multiple values to array
db.collection_name.updateOne(
  { name: "John Doe" },
  { $push: { tags: { $each: ["mongodb", "database"] } } }
```

```
)

// Add unique values to array
db.collection_name.updateOne(
  { name: "John Doe" },
  { $addToSet: { tags: "developer" } }
)

// Remove from array
db.collection_name.updateOne(
  { name: "John Doe" },
  { $pull: { tags: "developer" } }
)

// Remove multiple values from array
db.collection_name.updateOne(
  { name: "John Doe" },
  { $pullAll: { tags: ["developer", "mongodb"] } }
)

// Update array element by position
db.collection_name.updateOne(
  { name: "John Doe" },
  { $set: { "tags.0": "senior-developer" } }
```

```
)

// Update nested element in documents matching array criteria
db.collection_name.updateMany(
  { "comments.author": "Jane" },
  { $set: { "comments.$.approved": true } }
)
```

Delete

```
// Delete a single document
db.collection_name.deleteOne({ name: "John Doe" })

// Delete multiple documents
db.collection_name.deleteMany({ age: { $lt: 18 } })

// Delete all documents
db.collection_name.deleteMany({})
```

Query Operators

Comparison Operators

```
// Equal to
db.collection_name.find({ age: { $eq: 30 } })
// Also: db.collection_name.find({ age: 30 })
```

```
// Not equal to
db.collection_name.find({ age: { $ne: 30 } })

// Greater than
db.collection_name.find({ age: { $gt: 30 } })

// Greater than or equal to
db.collection_name.find({ age: { $gte: 30 } })

// Less than
db.collection_name.find({ age: { $lt: 30 } })

// Less than or equal to
db.collection_name.find({ age: { $lte: 30 } })

// In array of values
db.collection_name.find({ age: { $in: [25, 30, 35] } })

// Not in array of values
db.collection_name.find({ age: { $nin: [25, 30, 35] } })
```

Logical Operators

```
// AND (implicit in MongoDB when using multiple conditions)
db.collection_name.find({ age: 30, status: "active" })
```

```
// Explicit AND
db.collection_name.find({
  $and: [
    { age: { $gt: 25 } },
    { age: { $lt: 35 } }
  ]
})

// OR
db.collection_name.find({
  $or: [
    { age: 30 },
    { status: "active" }
  ]
})

// NOT
db.collection_name.find({
  age: { $not: { $gt: 30 } }
})

// NOR
db.collection_name.find({
```

```
  $nor: [
    { age: 30 },
    { status: "active" }
  ]
})
```

Element Operators

```
// Field exists
db.collection_name.find({ email: { $exists: true } })

// Field doesn't exist
db.collection_name.find({ email: { $exists: false } })

// Field is of specific type
db.collection_name.find({ age: { $type: "number" } })
db.collection_name.find({ age: { $type: 16 } })  // 16 is the BSON type code for int
```

Evaluation Operators

```
// Regex match
db.collection_name.find({ name: { $regex: /john/i } })

// Text search (requires text index)
db.collection_name.find({ $text: { $search: "mongodb tutorial" } })
```

```
// Expression evaluation
db.collection_name.find({
  $expr: { $gt: ["$field1", "$field2"] }
})

// Where JavaScript expression (use with caution - performance impact)
db.collection_name.find({
  $where: function() { return this.credits - this.debits < 0; }
})
```

Array Operators

```
// Array contains all elements
db.collection_name.find({ tags: { $all: ["mongodb", "database"] } })

// Array element matches criteria
db.collection_name.find({ "scores.0": { $gte: 90 } })

// Array size
db.collection_name.find({ tags: { $size: 3 } })

// Array element matching criteria
db.collection_name.find({
  scores: { $elemMatch: { type: "quiz", score: { $gt: 80 } } }
})
```

Cursor Methods

```
// Limit results

db.collection_name.find().limit(5)


// Skip results

db.collection_name.find().skip(5)


// Skip and limit (pagination)

db.collection_name.find().skip(10).limit(5)  // page 3, 5 items per page


// Sort results (1 for ascending, -1 for descending)

db.collection_name.find().sort({ age: 1 })  // ascending

db.collection_name.find().sort({ age: -1 })  // descending

db.collection_name.find().sort({ age: -1, name: 1 })  // multiple fields


// Count results

db.collection_name.find({ age: { $gt: 30 } }).count()


// Get only specific number of results

db.collection_name.find().limit(5).toArray()


// Check if cursor has more results

var cursor = db.collection_name.find();
```

```
cursor.hasNext()  // returns true or false

// Get next result
cursor.next()

// Iterate through cursor
var cursor = db.collection_name.find();
while (cursor.hasNext()) {
  printjson(cursor.next());
}

// For each on cursor
db.collection_name.find().forEach(function(doc) {
  print(doc.name);
})

// Map reduce on cursor
db.collection_name.find().map(function(doc) {
  return doc.name;
})
```

Aggregation

```
// Basic aggregation pipeline
db.collection_name.aggregate([
```

```
  { $match: { status: "active" } },
  { $group: { _id: "$category", total: { $sum: 1 } } }
])

// Match stage (filtering)
db.collection_name.aggregate([
  { $match: { age: { $gte: 30 } } }
])

// Group stage
db.collection_name.aggregate([
  { $group: {
    _id: "$department",
    averageSalary: { $avg: "$salary" },
    totalEmployees: { $sum: 1 }
  } }
])

// Project stage (reshaping)
db.collection_name.aggregate([
  { $project: {
    _id: 0,
    name: 1,
    age: 1,
```

```
    adultCategory: {
      $cond: { if: { $gte: ["$age", 18] }, then: "adult", else: "minor" }
    }
  } }
])

// Sort stage
db.collection_name.aggregate([
  { $sort: { age: -1 } }
])

// Limit stage
db.collection_name.aggregate([
  { $limit: 5 }
])

// Skip stage
db.collection_name.aggregate([
  { $skip: 5 }
])

// Unwind stage (flatten arrays)
db.collection_name.aggregate([
  { $unwind: "$tags" }
```

```
])

// Lookup stage (JOIN)
db.orders.aggregate([
  { $lookup: {
    from: "customers",
    localField: "customer_id",
    foreignField: "_id",
    as: "customer_info"
  } }
])

// Add fields
db.collection_name.aggregate([
  { $addFields: {
    totalScore: { $sum: "$scores" }
  } }
])

// Replace root
db.collection_name.aggregate([
  { $replaceRoot: { newRoot: "$details" } }
])
```

```
// Count
db.collection_name.aggregate([
  { $count: "total_documents" }
])

// Bucket (group by ranges)
db.collection_name.aggregate([
  { $bucket: {
    groupBy: "$age",
    boundaries: [18, 30, 50, 80],
    default: "other",
    output: {
      "count": { $sum: 1 },
      "averageSalary": { $avg: "$salary" }
    }
  } }
])

// Sort by count
db.collection_name.aggregate([
  { $sortByCount: "$category" }
])

// Output to collection
```

```
db.collection_name.aggregate([
  { $match: { status: "active" } },
  { $out: "active_users" }
])
```

Indexes

```
// Create single field index
db.collection_name.createIndex({ field_name: 1 })  // 1 for ascending, -1 for descending

// Create compound index
db.collection_name.createIndex({ field1: 1, field2: -1 })

// Create unique index
db.collection_name.createIndex({ email: 1 }, { unique: true })

// Create sparse index (only includes documents with the field)
db.collection_name.createIndex({ field_name: 1 }, { sparse: true })

// Create TTL index (automatically remove documents after seconds)
db.collection_name.createIndex({ createdAt: 1 }, { expireAfterSeconds: 3600 })

// Create text index
db.collection_name.createIndex({ content: "text" })
```

```
// Create multiple text index fields
db.collection_name.createIndex({ title: "text", content: "text" })

// Create geospatial index
db.collection_name.createIndex({ location: "2dsphere" })

// Create hashed index
db.collection_name.createIndex({ field_name: "hashed" })

// Create background index
db.collection_name.createIndex({ field_name: 1 }, { background: true })

// View all indexes
db.collection_name.getIndexes()

// Drop specific index
db.collection_name.dropIndex("index_name")
db.collection_name.dropIndex({ field_name: 1 })

// Drop all indexes
db.collection_name.dropIndexes()
```

Data Management

Import/Export

Export data to JSON

mongoexport --db=dbname --collection=collname --out=data.json

Export data to CSV

mongoexport --db=dbname --collection=collname --type=csv --fields=field1,field2 --out=data.csv

Import JSON data

mongoimport --db=dbname --collection=collname --file=data.json

Import CSV data

mongoimport --db=dbname --collection=collname --type=csv --headerline --file=data.csv

Backup/Restore

Backup database (mongodump)

mongodump --db=dbname --out=/backup/directory

Backup specific collection

mongodump --db=dbname --collection=collname --out=/backup/directory

Backup with compression

mongodump --db=dbname --out=/backup/directory --gzip

Restore database (mongorestore)

mongorestore --db=dbname /backup/directory/dbname

```
# Restore specific collection
mongorestore --db=dbname --collection=collname /backup/directory/dbname/collname.bson

# Restore with different database name
mongorestore --db=newdbname /backup/directory/dbname
```

Administration

User Management

```
// Create user
db.createUser({
  user: "username",
  pwd: "password",
  roles: [
    { role: "readWrite", db: "database_name" },
    { role: "dbAdmin", db: "database_name" }
  ]
})

// Grant additional roles to user
db.grantRolesToUser("username", [
  { role: "readWrite", db: "another_database" }
])
```

```
// Revoke roles from user
db.revokeRolesFromUser("username", [
  { role: "readWrite", db: "another_database" }
])

// Change user password
db.changeUserPassword("username", "newpassword")

// Get user info
db.getUser("username")

// Get all users
db.getUsers()

// Delete user
db.dropUser("username")
```

Database Status

```
// Server status
db.serverStatus()

// Database stats
db.stats()
```

```
// Collection stats
db.collection_name.stats()

// Current operations
db.currentOp()

// Kill specific operation
db.killOp(opId)

// Repair database
db.repairDatabase()
```

Replica Set Operations

```
// Check replica set status
rs.status()

// Initialize a replica set
rs.initiate({
  _id: "myReplicaSet",
  members: [
    { _id: 0, host: "mongodb0.example.net:27017" },
    { _id: 1, host: "mongodb1.example.net:27017" },
    { _id: 2, host: "mongodb2.example.net:27017" }
  ]
```

```
})

// Add member to replica set
rs.add("mongodb3.example.net:27017")

// Remove member from replica set
rs.remove("mongodb3.example.net:27017")

// Reconfigure replica set
rs.reconfig(config)

// Step down primary
rs.stepDown()

// Force a member to become primary (use with caution)
rs.freeze(0)
```

Sharding Operations

```
// Enable sharding for a database
sh.enableSharding("database_name")

// Shard a collection
sh.shardCollection("database_name.collection_name", { "shard_key": 1 })
```

```
// Add shard
sh.addShard("rs1/mongodb0.example.net:27017")

// Check sharding status
sh.status()
```

Data Modeling & Schema Validation

```
// Create a collection with validation
db.createCollection("users", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name", "email", "created_at"],
      properties: {
        name: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        email: {
          bsonType: "string",
          pattern: "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$",
          description: "must be a valid email address and is required"
        },
        phone: {
```

```
        bsonType: "string",
        description: "must be a string if provided"
      },
      age: {
        bsonType: "int",
        minimum: 18,
        maximum: 120,
        description: "must be an integer between 18 and 120 if provided"
      },
      status: {
        enum: ["active", "inactive", "pending"],
        description: "can only be one of the enum values if provided"
      },
      created_at: {
        bsonType: "date",
        description: "must be a date and is required"
      }
    }
  }
 },
 validationLevel: "strict",
 validationAction: "error"
})
```

```
// Update validation rules for existing collection
db.runCommand({
  collMod: "users",
  validator: { $jsonSchema: { /* schema definition */ } },
  validationLevel: "moderate",  // strict|moderate|off
  validationAction: "warn"      // error|warn
})
```

Performance & Optimization

```
// Explain query execution plan
db.collection_name.find({ age: { $gt: 30 } }).explain()

// Explain with verbose info (executionStats)
db.collection_name.find({ age: { $gt: 30 } }).explain("executionStats")

// Explain with all info (allPlansExecution)
db.collection_name.find({ age: { $gt: 30 } }).explain("allPlansExecution")

// Get collection statistics
db.collection_name.stats()

// Check index usage statistics
db.collection_name.aggregate([
  { $indexStats: {} }
```

])

// Analyze query performance

db.collection_name.find({ age: { $gt: 30 }
}).explain("executionStats").executionStats.executionTimeMillis

// Get top time-consuming operations

db.currentOp({ "active": true, "microsecs_running": { $gt: 1000000 } })

// Hint specific index

db.collection_name.find({ age: { $gt: 30 } }).hint({ age: 1 })

MongoDB Shell Tips

// Open MongoDB shell with specific config file

mongo --config /path/to/mongo.conf

// Run JavaScript file in MongoDB shell

mongo myScript.js

// Execute a single command and exit

mongo --eval "db.getSiblingDB('mydb').collection_name.find().limit(5)"

// Set profiling level (0=off, 1=slow queries, 2=all)

db.setProfilingLevel(1, 100) // Level 1, threshold 100ms

```
// Get profiling info
db.getProfilingStatus()

// View profiled operations
db.system.profile.find().pretty()

// Print in a formatted way
printjson(obj)

// Store variables
var result = db.collection_name.findOne()
var count = db.collection_name.countDocuments()

// Helper functions
show dbs
show collections
show users
show roles
show profile

// Format output
DBQuery.shellBatchSize = 10  // Show only 10 results at a time
```

```
// Enable autocompletion

DBQuery.prototype._prettyShell = true
```