#1 MongoDB installation in MacOS
#2 Set up database
#3 working with MongoDB, MQL
#4 CRUD operations
#5 Aggregation Pipelines (like unix pipes)

#1 MongoDB installation in MacOS

npm version
npm install -g m
m latest
mongod --version
brew tap mongodb/brew
brew install mongosh
mongosh --version
brew install mongodb-database-tools
mongoimport

#2 Set up database

mkdir <your dir>
cd <your dir>
mkdir mongod_only
mongod --dbpath mongod_only

note: A mongod is a daemon process for MongoDB. You could say it is the core unit of MongoDB. It handles data requests from the MongoDB Shell or drivers, manages data access, and performs background management operations.

in a 2nd terminal (mongo shell):
mongosh
show databases
db.test.insertOne({"hello": "world"})

.. above deployment is running in only 1 process, ie NOT for production; solution - replica set (a group of multiple mongod instances working together) usually with at least 3 replica set members, choose an odd number of voting replica set members:
(*) redundancy and high availability - In a replica set, your data is stored multiple times and all operations are asynchronously replicated to maintain the same data set. Because a replica set has multiple nodes that can take over as the primary of the deployment, a replica set also provides high availability.

CTRL-C to terminate the mongod process above, CTRL-D to exis the mongod shell above
mkdir replica_set_cmdline
cd replica_set_cmdline
openssl rand -base64 755 > keyfile
chmod 400 keyfile
mkdir -p m{1,2,3}/db //creating m1,m2,m3 folders and sub-folder db
mongod --replSet myReplSet --dbpath ./m1/db --logpath ./m1/mongodb.log --port 27017 --fork --keyFile ./keyfile
mongod --replSet myReplSet --dbpath ./m2/db --logpath ./m2/mongodb.log --port 27018 --fork --keyFile ./keyfile
mongod --replSet myReplSet --dbpath ./m3/db --logpath ./m3/mongodb.log --port 27019 --fork --keyFile ./keyfile

in a different terminal:
mongosh “mongodb://localhost:27017”
rs.initiate()
use admin
db.createUser({user: '<your name>', pwd: passwordPrompt(), roles: ["root"]})
db.getSiblingDB("admin").auth("<your name>", passwordPrompt())
rs.add("localhost:27018")
rs.add("localhost:27019")
rs.status()
db.serverStatus()['repl']
exit
.. back to the 1st terminal and:
killall mongod
cd ..
rm -rf replica_set_cmdline

mkdir replicaset
cd replicaset
openssl rand -base64 755 > keyfile
chmod 400 keyfile
mkdir -p m{1,2,3}/db
touch m1.conf
code . //and edit m1.conf with:
storage:
  dbPath: m1/db
net:
  bindIp: localhost
  port: 27017
security:
  authorization: enabled
  keyFile: keyfile
systemLog:
  destination: file
  path: m1/mongod.log
processManagement:
  fork: true
replication:
  replSetName: mongodb-essentials-rs

cp m1.conf m2.conf
cp m1.conf m3.conf
mongod -f m1.conf
mongod -f m2.conf
mongod -f m3.conf

.. and in a 2nd terminal:
mongosh
use admin

```
config = {
   _id: "mongodb-rs",
   members: [
      { _id: 0, host: "localhost:27017" },
      { _id: 1, host: "localhost:27018" },
      { _id: 2, host: "localhost:27019" }
   ]
};
rs.initiate(config)
db.createUser({user: '<your name>', pwd: passwordPrompt(), roles: ["root"]})
db.getSiblingDB("admin").auth("<your name>")
rs.add("localhost:27018")
rs.add("localhost:27019")
rs.status()
db.serverStatus()['repl']
```

MongoDB db tools –
mongostat
mongodump
mongorestore
mongoexport
mongoimport

```
cd datasets
mongoimport --username="<your name>" --authenticationDatabase="admin" --db=sample_data inventory.json
mongoimport --username="<your name>" --authenticationDatabase="admin" --db=sample_data movies.json
mongoimport --username="<your name>" --authenticationDatabase="admin" --db=sample_data orders.json
```

for debug:
check log files at m*/mongod.log
disable fork option in the config file
check the oplog
increase the log level

#3 working with MongoDB, MQL

Document model – natively works with JSON documents, actually uses BSON; each doc is at most 16MB

```
use blog
show collections
db.authors.insertOne({"name": "<your name>"})

db.authors.insertOne({"name": "Joe Nash"})
db.authors.insertMany([
   {"name": "Eliot Horowitz"},
   {"name": "Dwight Merriman"},
   {"name": "Kevin P. Ryan"}
])
db.authors.find()
```

```
db.authors.find({ "name": "<your name>"})
db.authors.find({ name: "<your name>"})

db.authors.updateOne(
   { name: "<your name>"},
   { $set: { website: "<your website URL> " } }
)
db.authors.find({ name: "<your name>"})
db.authors.updateMany(
   { },
   { $set: { books: [] } }
)
db.authors.find()
db.authors.deleteOne({ name: "Joe Nash" })
db.authors.deleteMany({}) // this will delete ALL documents!!!

db.authors.find()
db.createIndex({ name: 1 }) // there are different types of indexes, here: single field

database for a phone number lookup app by name or phone#:
use lookup
db.records.insertOne({
   "name": "<your name>l",
   "number": "1234567890",
   "profession": "<best whatever>",
   "website": "<your website URL>"
})
db.records.findOne()
db.records.createIndex({name: 1})
db.records.createIndex({number: 1})
```

#4 CRUD operations
writeConcern, readConcern, sort/skip/limit, $set/$unset/$inc/$mul/$max/$min,
arrays {query: exact match - [], $all, $elemMatch; update: $push, $addToSet, $pop},
startSession, endSession, startTransaction, commitTransaction

```
show dbs
use sample_data
show collections //and in this case, they are: inventory, movies, orders
db.inventory.findOne()
db.orders.findOne()
db.movies.findOne()
db.movies.findOne({"ratings.mndb": 10})
db.movies.findOne({"genres.0": "Musical"})

comparison operators:
use sample_data
db.inventory.findOne()
db.inventory.findOne({ "variations.quantity": { $gte: 8 } })
```

```
db.inventory.findOne({ "price": { $lt: 1700 } })
db.inventory.findOne({ "variations.variation": { $in: [ "Blue", "Red" ] } })
db.inventory.findOne({ "variations.variation": { $nin: [ "Blue", "Red" ] } })
```

logical operators:

```
use sample_data
db.inventory.findOne()
db.inventory.findOne(
   {
      $and: [
         { "variations.quantity": { $ne: 0} },
         { "variations.quantity": { $exists: true } }
      ]
   }
)
db.inventory.findOne(
   {
      $or: [
         { "variations.variation": "Blue" },
         { "variations.variation": "Green" }
         { "variations.variation": "Teal" },
      ]
   }
)
db.inventory.findOne(
   {
      $nor: [
         { price: { $gt: 8000 } },
         { "variations.variation": "Blue" }
      ]
   }
)
db.inventory.findOne(
   {
      $not: { price: { $gt: 20000 } },
   }
)
```

sort, skip, limit (MongoDB performs these in this order)[*]:

```
db.movies.find().sort({title: 1})
db.movies.find({}, { title: 1, genres: 1 }).sort( {title: 1})
db.movies.find({}, { title: 1, genres: 1 }).sort( {title: -1})
db.movies.find({}, { title: 1, genres: 1 }).sort( {director: 1, title: 1})
db.movies.find({}, { title: 1, genres: 1 }).sort( {title: 1}).skip(100)
db.movies.find({}, { title: 1, genres: 1 }).sort( {title: 1}).skip(100).limit(5)
```

[*] when the sort is a common query pattern, use an index; if no index, sort/limit is faster

updateOne and updateMany:

```
use blog
```

```
show collections
db.authors.find()
db.authors.updateOne(
   { name: "<your name>" },
   { $set: { message: "Hello World!" } }
)
db.authors.find()
db.authors.updateMany(
   {},
   { $set: { message: "Hello" } }
)
db.authors.find()
db.authors.updateMany(
   {},
   { $unset: { message: "" } }
)
db.authors.find()
```

Arrays:

```
use sample_data
db.movies.findOne()
db.movies.find({genres: "Comedy"})
db.movies.find({genres: ["Comedy", "Drama", "Thriller"]})
db.movies.find({genres: { $all: ["Comedy", "Drama"] } })
db.inventory.findOne()
db.inventory.find(
   {
      variations: {
         $elemMatch: {
            variation: "Blue",
            quantity: { $gt: 8  }
         }
      }
   }
)
db.movies.findOne()
db.movies.updateOne(
   {
      title: 'The Adventures of Tom Thumb & Thumbelina'
   },
   {
      $push: { genres: "Test" }
   }
)
db.movies.findOne( {title: 'The Adventures of Tom Thumb & Thumbelina'} )
db.movies.updateOne(
   {
      title: 'The Adventures of Tom Thumb & Thumbelina'
   },
```

```
  {
    $addToSet: { genres: "Test" }
  }
)
db.movies.findOne( {title: 'The Adventures of Tom Thumb & Thumbelina'} )
db.movies.updateOne(
  {
    title: 'The Adventures of Tom Thumb & Thumbelina'
  },
  {
    $addToSet: { genres: "Green" }
  }
)
db.movies.findOne( {title: 'The Adventures of Tom Thumb & Thumbelina'} )
db.movies.updateOne(
  {
    title: 'The Adventures of Tom Thumb & Thumbelina'
  },
  {
    $pop: { genres: 1 }
  }
)
db.movies.findOne( {title: 'The Adventures of Tom Thumb & Thumbelina'} )
db.movies.updateOne(
  {
    title: 'The Adventures of Tom Thumb & Thumbelina'
  },
  {
    $pop: { genres: 1 }
  }
)
db.movies.findOne( {title: 'The Adventures of Tom Thumb & Thumbelina'} )

Transactions(atomic update to multiple documents, either all writes happen or none do):
use blog
session = db.getMongo().startSession( { readPreference: { mode: "primary" } } )
session.startTransaction()
session.getDatabase("blog").updateMany( {}, {$set: {message: "Transaction occurred"} } )
session.commitTransaction()
session.endSession()
db.authors.find()

$expr:
use sample_data
db.movies.findOne()
db.movies.find({ title: 1, ratings: 1 })
db.movies.find(
  { title: 1, ratings: 1 }
)
```

db.movies.find({ $expr: { $gt: [{$multiple: [“ratings.mndb”, 10]}, “ratings.soft_avocados”] } })

Arithmetic Expression Operators
$abs
$add
$ceil
$divide
$exp
$floor
$ln
$log
$log10
$mod
$multiply
$pow
$sqrt
$subtract
$trunc

Array Expression Operators
$arrayElemAt
$arrayToObject
$concatArrays
$filter
$in
$indexOfArray
$isArray
$last
$map
$objectToArray
$range
$reduce
$reverseArray
$size
$slice
$zip

Boolean Expression Operators
$and
$not
$or

Comparison Expression Operators
$cmp
$eq
$gt
$gte
$lt
$lte
$ne

Conditional Expression Operators
$cond
$ifNull
$switch

Custom Aggregation Expression Operators
$accumulator
$function

Data Size Operators
$binarySize
$bsonSize

Date Expression Operators
$datetAdd
$dateDiff
$dateFromParts
$dateFromString
$dateSubtract
$dateToParts
$dateToString
$dateTrunc
$dayOfMonth
$dayOfWeek
$dayOfYear
$hour
$isoDayOfWeek
$isoWeek
$isoWeekYear
$millisecond
$minute
$month
$second
$toDate
$week
$year

Literal Expression Operator
$literal

Miscellaneous Operators
$getField
$rand
$sampleRate

Object Expression Operators
$mergeObjects
$objectToArray
$setField

Set Expression Operators
$allElementsTrue
$anyElementTrue
$setDifference
$setEquals
$setIntersection
$setIsSubset
$setUnion

String Expression Operators
$concat
$dateFromString
$dateToString
$indexOfBytes
$indexOfCP
$ltrim
$regexFindall
$regexMatch
$replaceOne
$replaceAll
$rtrim
$split
$strLenBytes
$strLenCP
$strcasecmp
$substr
$substrBytes
$substrCP
$toLower
$toString
$trim
$toUpper

Text Expression Operator
$meta

Trigonometry Expression Operators
$sin
$cos
$tan
$asin
$acos
$atan
$atan2
$asinh
$acosh
$atanh
$sinh
$cosh
$tanh
$degreesToRadians
$radiansTtoDegrees

Type Expression Operators
$convert
$isNumber
$toBool
$toDate
$toDecimal
$toDouble
$toInt
$toLong
$toObjectId
$toString
$type

Accumulators ($group)
$accumulator
$addToSet
$avg
$count
$first
$last
$max
$mergeObjects
$min
$push
$stdDevPop
$stdDevSamp
$sum

Variable Expression Operator
$let

Window Operators
$addToSet
$avg
$count
$covariancePip
$covarianceSamp
$denseRank
$derivative
$documentNumber
$expMovingAvg
$first
$integral
$last
$max
$min
$push
$rank
$shift
$stdDevPop
$stdDevSamp
$sum

#5 Aggregation Pipelines (like unix pipes)

Pipeline stages:

$addFields	$graphLookup	$merge	$setWindowFields
$bucket	$group	$out	$skip
$bucketAuto	$indexStats	$planCacheStats	$sort
$collStats	$limit	$project	$sortByCount
$count	$listLocalSessions	$redact	$unionWith
$currentOp	$listSessions	$replaceRoot	$unset
$facet	$lookup	$replaceWith	$unwind
$geoNear	$match	$set	

db.collection.aggregate([

Arithmetic Expression Operators
$abs
$add
$ceil
$divide
$exp
$floor
$ln
$log
$log10
$mod
$multiply
$pow
$sqrt
$subtract
$trunc

Array Expression Operators
$arrayElemAt
$arrayToObject
$concatArrays
$filter
$in
$indexOfArray
$isArray
$last
$map
$objectToArray
$range
$reduce
$reverseArray
$size
$slice
$zip

Boolean Expression Operators
$and
$not
$or

Comparison Expression Operators
$cmp
$eq
$gt
$gte
$lt
$lte
$ne

Conditional Expression Operators
$cond
$ifNull
$switch

Custom Aggregation Expression Operators
$accumulator
$function

Data Size Operators
$binarySize
$bsonSize

Date Expression Operators
$datetAdd
$dateDiff
$dateFromParts
$dateFromString
$dateSubtract
$dateToParts
$dateToString
$dateTrunc
$dayOfMonth
$dayOfWeek
$dayOfYear
$hour
$isoDayOfWeek
$isoWeek
$isoWeekYear
$millisecond
$minute
$month
$second
$toDate
$week
$year

Literal Expression Operator
$literal

Miscellaneous Operators
$getField
$rand
$sampleRate

Object Expression Operators
$mergeObjects
$objectToArray
$setField

Set Expression Operators
$allElementsTrue
$anyElementTrue
$setDifference
$setEquals
$setIntersection
$setIsSubset
$setUnion

String Expression Operators
$concat
$dateFromString
$dateToString
$indexOfBytes
$indexOfCP
$ltrim
$regexFindall
$regexMatch
$replaceOne
$replaceAll
$rtrim
$split
$strLenBytes
$strLenCP
$strcasecmp
$substr
$substrBytes
$substrCP
$toLower
$toString
$trim
$toUpper

Text Expression Operator
$meta

Trigonometry Expression Operators
$sin
$cos
$tan
$asin
$acos
$atan
$atan2
$asinh
$acosh
$atanh
$sinh
$cosh
$tanh
$degreesToRadians
$radiansTtoDegrees

Type Expression Operators
$convert
$isNumber
$toBool
$toDate
$toDecimal
$toDouble
$toInt
$toLong
$toObjectId
$toString
$type

Accumulators ($group)
$accumulator
$addToSet
$avg
$count
$first
$last
$max
$mergeObjects
$min
$push
$stdDevPop
$stdDevSamp
$sum

Variable Expression Operator
$let

Window Operators
$addToSet
$avg
$count
$covariancePip
$covarianceSamp
$denseRank
$derivative
$documentNumber
$expMovingAvg
$first
$integral
$last
$max
$min
$push
$rank
$shift
$stdDevPop
$stdDevSamp
$sum

$group:

```
use sample_data
db.inventory.findOne()
db.inventory.aggregate( [
  {
    $group: {
      _id: "$source",
    }
  }
] )
db.inventory.aggregate( [
  {
    $group: {
      _id: "$source",
      count: { $sum: 1 }
    }
  }
] )
db.inventory.aggregate( [
  {
    $group: {
      _id: "$source",
      count: { $sum: 1 },
      items: { $push: "$name" }
    }
  }
] )
db.inventory.aggregate( [
  {
```

```
      $group: {
         _id: "$source",
         count: { $sum: 1 },
         items: { $push: "$name" },
         avg_price: { $avg: "$price"}
      }
   }
] )
```

$bucket vs $bucketAuto

$unwind:

The primary use of `$unwind` is to "flatten" array fields, making it easier to perform operations on individual array elements.

```
db.inventory.findOne()
db.inventory.aggregate( [
   {
      $unwind: "$variations"
   }
] )
db.inventory.findOne()
db.inventory.aggregate( [
   {
      $unwind: "$variations",
   },
   {
      $match: { "variations.variation": "Purple" }
   }
] )
.. more performant:
db.inventory.aggregate( [
   {
      $match: { "variations.variation": "Purple" }
   },
   {
      $unwind: "$variations",
   },
   {
      $match: { "variations.variation": "Purple" }
   }
] )
```

$merge/$out:

```
db.inventory.aggregate( [
   {
      $match: { "variations.variation": "Purple" }
   },
```

```
  {
    $unwind: "$variations",
  },
  {
    $match: { "variations.variation": "Purple" }
  },
  {
    $out: { db: "sample_data", coll: "purple" }
  }
] )
show collections
db.purple.find()
db.inventory.aggregate( [
  {
    $match: { "variations.variation": "Purple" }
  },
  {
    $unwind: "$variations",
  },
  {
    $match: { "variations.variation": "Purple" }
  },
  {
    $merge: {
      into: "purple",
      on: "_id",
      whenMatched: "keepExisting"
      whenNotMatched: "insert"
    }
  }
] )

$function:
db.movies.findOne()
db.movies.aggregate([
  {
    $project: {
      title: 1,
      actors: {
        $function: {
          body: 'function(actors) { return actors.sort(); }',
          args: [ "$actors" ],
          lang: "js"
        }
      }
    }
  }
])
```

$lookup (for: join):

```
db.orders.aggregate([
    { $lookup: {
          from: "inventory",
          localField: "car_id",
          foreignField: "_id",
          as: "car_id"
    } }
])
```

Performance:

```
db.movies.explain("executionStats").aggregate( [
    { $project: {
        release_year: {$year: "$release_year"},
        title: 1
    } },
    { $lookup: {
        from: "inventory",
        localField: "release_year",
        foreignField: "year",
        as: "year"
    } }
] )
```

.. then check totalKeysExamined and collectionScans, etc .. mongod.log too ..

Use Profiler: db.setProfilingLevel(1, { slowms: 20 }) . db[“system.profile”].find()

Common optimizations:

$sort + $limit, $project as the final stage, Hinting, Analytics nodes