MongoDB

Documents, CRUD and queries in mongosh.

01 Run MongoDB locally

```
docker run --name mongodb \
  -p 27017:27017 -d \
  mongodb/mongodb-community-server:8.0-ubi8
docker exec -it mongodb mongosh
docker stop mongodb    # docker start resumes
```

Data lasts until you remove the container; a named volume keeps it longer. mongosh connects to localhost:27017.

02 Insert documents

```
db.users.insertOne({
  name: "John Doe",
  email: "john@example.com",
  created: new Date()
})
db.users.insertMany([
  { name: "Alice", age: 25 },
  { name: "Bob", age: 35 }
])
```

MongoDB adds an _id if you leave it out and returns insertedId(s). One insertMany() beats many insertOne() calls.

03 Find, project & sort

```
db.users.findOne({ name: "John Doe" })
db.users.find(
  { age: { $gte: 21 }, status: "active" },
  { name: 1, email: 1, _id: 0 }
).sort({ created: -1 }).skip(20).limit(10)
db.users.countDocuments({ age: 30 })
db.users.distinct("address.city")
```

findOne() returns the first match only. The second argument is a projection: 1 keeps a field, 0 drops it.

04 Update & upsert

```
db.users.updateMany({ status: "inactive" },
  { $set: { status: "archived" } })
db.users.updateOne(
  { email: "new@example.com" },
  { $set: { name: "New User" } },
  { upsert: true }   // insert if no match
)
```

$set changes only listed fields. findOneAndUpdate() returns the doc, the new one with returnDocument: "after".

05 Delete documents

```
db.users.deleteOne({ _id: userId })
db.users.deleteMany({ status: "archived" })
db.users.deleteMany({})   // every document!
db.users.findOneAndDelete({ _id: userId })
```

Deletes are permanent. Soft delete instead: $set deleted: true, then query { deleted: { $ne: true } }.

06 Comparison operators

$eq / $ne	Equal / not equal
$gt / $gte	Greater than (or equal)
$lt / $lte	Less than (or equal)
$in / $nin	In / not in a list

```
db.products.find({
  price: { $gt: 20, $lte: 100 },
  category: { $in: ["books", "games"] }
})
```

{ price: 29.99 } is an implicit $eq. Dates compare the same way: { $gte: lastWeek }.

07 Logical operators

```
db.products.find({ $or: [
  { category: "sale" },
  { price: { $lt: 20 } }
] })
db.products.find({ $nor: [
  { discontinued: true }, { stock: 0 }
] })
```

Listing several fields is an implicit AND. $nor matches when none match. $not inverts an operator: { $not: { $eq: "x" } }.

08 Array queries

```
db.posts.find({ tags: "mongodb" })
db.users.find({ hobbies: { $size: 3 } })
db.users.find({ "addresses.0.city": "NYC" })
db.orders.find({ items: { $elemMatch: {
  product: "laptop",
  quantity: { $gte: 2 }
} } })
```

A plain value matches any element; $all needs every listed value. $size takes an exact count, not a range.

09 Text & regex search

```
db.articles.createIndex(
  { title: "text", content: "text" })
db.articles.find(
  { $text: { $search: "mongodb -sql" } },
  { score: { $meta: "textScore" } }
).sort({ score: { $meta: "textScore" } })
db.users.find({ name: /^john/i })
```

One text index per collection. Quote a phrase to match it exactly; -word excludes. Regex needs no text index.

MongoDB

Update operators, aggregation, indexes and schema design.

10 Field update operators

`$set / $unset`	Set a field / remove it
`$inc / $mul`	Add to or multiply a number
`$rename`	Rename a field
`$min / $max`	Set only if lower / higher
`$setOnInsert`	Set only when upserting

Dot notation reaches nested fields: "address.city". $inc is atomic, so it is safe for concurrent updates.

11 Array update operators

```
db.posts.updateOne({ _id: postId }, {
  $addToSet: { tags: "news" },  // no dupes
  $push: { comments: {
    $each: [c1, c2], $sort: { date: -1 },
    $slice: 10   // keep the 10 newest
  } }
})
```

$pull removes every match; $pop drops the first (-1) or last (1). "items.$" is the matched element, "scores.$[]" all.

12 Aggregation pipeline

```
db.orders.aggregate([
  { $match: { status: "completed" } },
  { $group: { _id: "$customerId",
    totalSpent: { $sum: "$total" } } },
  { $sort: { totalSpent: -1 } },
  { $limit: 10 }
])
```

Stages run in order, so $match early. $sum: 1 counts; also $avg, $min, $max, $push and $addToSet.

13 $lookup & $unwind

```
db.sales.aggregate([
  { $lookup: { from: "products",
    localField: "productId",
    foreignField: "_id", as: "product" } },
  { $unwind: "$product" },
  { $group: { _id: { $month: "$date" },
    revenue: { $sum: "$total" } } }
])
```

$lookup joins matches into an array; $unwind outputs one document per element.

14 Expression operators

```
db.users.aggregate([{ $project: {
  code: { $substrCP: ["$name", 0, 3] },
  grade: { $cond: {
    if: { $gte: ["$score", 90] },
    then: "A", else: "B"
  } }
} }])
```

$switch takes many branches. $size, $filter, $first and $last work on arrays; $$ marks a variable.

15 Indexes

```
db.users.createIndex({ email: 1 },
  { unique: true })
db.orders.createIndex(
  { customerId: 1, createdAt: -1 })
db.sessions.createIndex({ createdAt: 1 },
  { expireAfterSeconds: 3600 })   // TTL
```

1 is ascending, -1 descending. Too many indexes slow writes. Prefer partial indexes to sparse ones.

16 Explain & profiling

```
db.users.find({ age: { $gte: 18 } })
  .explain("executionStats")
db.users.find({ email: "john@example.com" })
  .hint({ email: 1 })   // force an index
db.users.getIndexes()
db.setProfilingLevel(1, { slowms: 100 })
db.system.profile.find().sort({ ts: -1 })
```

COLLSCAN means no index was used. A customerId + createdAt index does not help a query on createdAt alone.

17 Schema validation

```
db.createCollection("users", {
  validator: { $jsonSchema: {
    bsonType: "object",
    required: ["name", "email"],
    properties: {
      age: { bsonType: "int", minimum: 18 }
    }
  } }
})
```

Change rules with collMod. validationAction: error or warn.

18 Schema design patterns

Embed	1:1 or 1:few, read together
Reference	1:many or many:many by id
Hybrid	Embed hot fields, keep the id
Bucket	Group time-series readings
Polymorphic	One collection, a type field

Pick from your access patterns. Avoid unbounded arrays.