

MongoDB

Learning MongoDB

PRAVEEN NAIR

Introduction to MongoDB

MongoDB is a document database.

MongoDB is a non-relational, non-tabular database.

Relational data is stored differently.

Instead of having multiple tables all the related data are stored together.

In MongoDB, tables are called collections.

MongoDB can be installed locally or in cloud called MongoDB Atlas

Mongosh or Compass can be used to query MongoDB

Advantages of MongoDB

Flexibility: MongoDB is schema-less, meaning you don't need to design a schema for the database.

Scalability: MongoDB can be horizontally scaled by distributing data across multiple servers, a process called sharding.

Performance: MongoDB is fast at inserting or updating large numbers of records. It also supports geospatial efficiently.

MongoDB Community Server

<https://www.mongodb.com/try/download/community>

Choose MSI

MongoDB Shell Download

<https://www.mongodb.com/try/download/shell>

Choose MSI

Connect to local mongodb

Type mongosh –version

Type mongosh to get prompt

show dbs

use myproj to create or access new db

db.dropDatabase("dbname") to delete database (or db.dropDatabase())

show collections

db.createCollection("employees")

db.employees.drop() to delete collection

db.restaurant.renameCollection('restaurants') //rename collection

Inserting Data

```
db.employees.insertOne({  
    name: "John Smith",  
    email: "john@gmail.com",  
    department: "IT",  
    salary: 1456,  
    location: ["FL", "OH"],  
    date: Date()  
})  
  
db.employees.find()
```

Inserting Multiple Data

```
db.employees.insertMany([{
  name: "Mike Joseph",
  email: "mike@gmail.com",
  department: "IT",
  salary: 2456,
  location: ["FL", "TX"],
  date: Date()
},
{ name: "Cathy G",
  email: "cathy@gmail.com",
  department: "IT",
  salary: 3456,
  location: ["AZ", "TX"],
  date: Date()
}])
```

Find Data – part 1

```
db.employees.find() //returns first 20, then type it for more documents  
db.employees.find().skip(2)  
db.employees.findOne()  
db.users.find().sort({name:1}) //sorting -1 for reverse  
db.users.find().limit(1) //returns 1 document sort by object id  
db.users.find().sort({name:1}).limit(3)  
db.employees.find( {department: "IT"} )  
db.users.find({name:"Cathy",pass:"1234"}) //two condition  
db.employees.find({}, {_id: 0, salary: 1, date: 1}) //cannot give 0  
db.users.find({},{_id:false,name:true}) //cannot give false  
db.employees.find({}, {_id: 0, salary: 0, date: 1}) //either use 0 or 1, can't use both
```

Find Data – part 2

```
db.users.find({'address.city':'Gwenborough'}) //query nested documents  
db.users.find({address.geo.lat:-37.3159})  
db.employees.find({'location':'TX'}) //where location : ['FL','TX']  
db.users.find().count()  
db.employees.find({}, {"dept": "$department", email:1,salary:1}) //dept is alias  
db.users.find({'address.city':'Gwenborough'}) //query nested documents  
db.users.find({address.geo.lat:-37.3159})  
db.employees.find({'location':'TX'}) //where location : ['FL','TX']
```

returns first 20, then type it for more documents

Query Operators – part 1

```
db.employees.find({department:{$eq:'HR'}})
```

```
db.users.find({email:{$ne:'cathy@gmail.com'}})
```

```
db.employees.find({salary:{$gt:3000}})
```

```
db.employees.find({salary:{$gte:3000}})
```

```
db.employees.find({salary:{$gte:3000,$lt:5000}})
```

```
db.employees.find({salary:{$gt:1000},department:{$eq:'HR'}})
```

```
db.employees.find({salary:{$gt:2000},department:{$in:['HR','IT']}})
```

Query Operators – part 2

```
db.employees.find({salary:{$gt:2000},department:{$nin:['HR','IT']}})  
db.employees.find({$or:[{salary:{$gt:2000}},{department:{$eq:'HR'}}]})  
db.employees.find({$and:[{salary:{$gt:2000}},{department:{$eq:'HR'}}]})  
db.employees.find({$nor:[{salary:{$gt:2000}},{department:{$eq:'HR'}}]})  
//like and but both should be false  
db.employees.find({department:{$not:{$eq:'HR'}}})  
db.users.find({email1:{$exists:false}})
```

Update Document

```
db.employees.updateOne({email:'cathy@gmail.com'},{$set:{department:'HR'}})

db.employees.updateOne(
  { email: "ria@gmail.com" },
  {
    $set:
    {
      name: "Ria K",
      email: "ria@gmail.com",
      department: "HR",
      salary: 5000,
      location: ["FL", "LA"],
      date: Date()
    }
  },
  { upsert: true }
)

db.employees.updateMany({}, { $set: { date: Date() } })
```

Delete Document

```
db.employees.deleteOne({email:'ria@gmail.com'})
```

```
db.employees.deleteMany({email:'ria@gmail.com'})
```

Update Operators(fields)

```
db.employees.updateOne({email:'cathy@gmail.com'},{$set:{email:'cathy@hotmail.com'}})
```

```
db.employees.updateMany({},{$set:{points:0}}) -- new field
```

```
db.employees.updateMany({},{$inc:{points:70}})
```

```
db.employees.updateMany({},{$rename:{points:'score'}})
```

```
db.employees.updateMany({},{$unset:{score:""}}) //deletes the field
```

Misc – skip and limit

```
db.employees.find().skip(2)
```

```
db.employees.find().skip(2).limit(1)
```

Used for pagination

Connect to local mongodb

Type mongosh –version

Type mongosh to get prompt

show dbs

use myproj to create or access new db

db.dropDatabase("dbname") to delete database (or db.dropDatabase())

show collections

db.createCollection("employees")

db.employees.drop() to delete collection

db.restaurant.renameCollection('restaurants') //rename collection

Query Operators - 3

```
db.employees.find(  
  {department:{$in:["HR","Admin"]}}  
)
```

```
db.employees.find(  
  {department:{$nin:["HR","Admin"]}}  
)
```

Update Operators (arrays)

```
db.employees.updateOne({email:'cathy@hotmail.com'},{$addToSet:{location:'F  
L'}}) //duplicates won't be added, use push instead  
  
db.employees.updateOne({email:'cathy@hotmail.com'},{$pop:{location:1}}) -try  
-1  
  
db.employees.updateMany({email:'cathy@hotmail.com'},{$pull:{points:{$gt:1}}}  
)  
  
db.employees.updateMany({email:'cathy@hotmail.com'},{$push:{points:5}})
```

Indexes (improves search but slows insert, update)

```
db.users.find({email:'cathy@gmail.com'}).explain("executionStats")
totalDocsExamined: 13,
```

```
db.users.createIndex({email:1}) //ascending
totalDocsExamined: 3,
```

```
db.users.getIndexes()
```

```
db.users.createIndex({'email':1},{unique:true})
```

```
db.users.dropIndex("email_1")
```

Aggregation pipeline

```
db.employees.aggregate([  
    {pipeline1 or stage 1 },  
    {pipeline2 or stage 2},  
])
```

Aggregation - \$match

```
db.employees.aggregate([
  {
    $match: {} //stage 1
  },
  {
    $group: { _id: "$department", total: { $sum: "$salary" } } //stage 2
  },
  {
    $sort: { "department": -1 }
  },
])
```

Sorting - collation

Product.find()

```
.collation({ locale: 'en', strength: 2 }) // strength: 2 = case-insensitive  
.sort({ name: 1 });
```

Aggregation - \$sort

```
db.employees.aggregate([
  {
    $sort: { "name": -1 }
  },
  {
    $project: {
      "name": 1,
      "email": 1,
      "salary":1
    }
  },
  {
    $limit: 5
  }
])
```

Aggregation - \$match

```
db.employees.aggregate([
  {
    $match: { salary: { $gt: 1000 } } //state 1
  },
  {
    $group: { _id: "$department", total: { $sum: "$salary" } } //stage 2
  }
])
```

Aggregation - \$group

An aggregation pipeline return results for groups of documents. For example, return the total, average, maximum, and minimum values.

```
db.employees.aggregate([
  {
    $group: {
      _id: "$department",
      Total: { $sum: "$salary" },
      Highest: { $max: "$salary" },
      Lowest: { $min: "$salary" },
      Average: { $avg: "$salary" },
    },
  },
]);
```

Aggregation - \$limit

```
db.employees.aggregate([
  { $group: { _id: "$department", Total: { $sum: "$salary" } } },
  { $limit: 1 },
]);
```

Aggregation - \$project

```
db.employees.aggregate([
  {
    $project: {
      "name": 1,
      "email": 1,
      "salary": 1
    }
  },
  {
    $limit: 2
  }
])
```

\$project – remove field

```
db.employees.aggregate([{ $project: { _id: 0, name: 0 } }]);
```

\$project – rename & add calc

```
db.employees.aggregate([
  {
    $project: {
      empname: "$name",
      email: 1,
      salary: 1,
      AnnualSalary: { $multiply: [12, "$salary"] },
    },
  },
]);
```

\$rename field (object)

```
db.profile.insertOne({  
  _id:ObjectId("6980326a71d308ebb027caa2"),  
  address:{line1:"3356 Lane 1",city:"Columbus",State:"OH"},  
  phone:7564555  
})  
  
db.profile.find(  
  {},  
  {address1:"$address.line1",city:"$address.city"}  
)
```

\$unwind – Convert Array to Object

```
db.empSkills.insertOne({  
  skills:["Java","Python",".NET"]  
})  
  
db.empSkills.aggregate([  
  {$unwind:"$skills"}  
])
```

Aggregation -\$cond

```
{ $cond: [ <boolean-expression>, <true-case>, <false-case> ] }
```

```
.....
```

```
db.employees.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      salary: 1,
      grade: { $cond: [{ $gte: ["$salary", 2000] }, "Grade A", "Grade B"] },
    },
  },
]);
```

Aggregation - \$cond-if

```
{ $cond: { if: <boolean-expression>, then: <true-case>, else: <false-case> } }
```

```
.....
```

```
db.employees.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      salary: 1,
      grade: {
        $cond: {
          if: { $gte: ["$salary", 2000] },
          then: "Grade A",
          else: "Grade B",
        },
      },
    },
  },
]);
```

Switch case - syntax

```
Grade: {  
  $switch: {  
    branches: [  
      { case: <boolean-expression>, then: <result> },  
      { case: <boolean-expression>, then: <result> }  
      // more branches...  
    ],  
    default: <result>  
  }  
}
```

Switch case

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      level: {
        $switch: {
          branches: [
            { case: { $gte: ["$score", 90] }, then: "A" },
            { case: { $gte: ["$score", 75] }, then: "B" },
            { case: { $gte: ["$score", 60] }, then: "C" }
          ],
          default: "Fail"
        }
      }
    }
  }
])
```

Aggregation - \$lookup prep

```
db.createCollection("orders")
```

```
db.orders.insertOne({'empid':ObjectId('65fc6dd2198f1b870853d26e'), 'date':  
Date(), 'orderValue':5000})
```

Aggregation - \$lookup

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

Aggregation - \$lookup – pipeline

```
db.orders.aggregate([
  {$lookup: {
    from:"employees",
    let:{uid:"$empid"},
    pipeline:[
      {$match:{$expr:{$eq:[ "$_id", "$$uid" ]}}}
    ],
    as:"users"
  }}
])
```

Aggregation - \$lookup – pipeline - project

```
db.orders.aggregate([
  {
    $lookup: {
      from: "employees",
      let: { uid: "$empid" },
      pipeline: [
        { $match: { $expr: { $eq: ["$_id", "$$uid"] } } },
        {
          $project: {
            _id: 0,
            name: 1,
          },
        },
      ],
      as: "users",
    },
    { $unwind: "$users" },
    {$project:{
      name:"$users.name",
      orderValue:1
    }}
  });
})
```

Aggregation - \$out (creates ratingbydep collection)

```
db.employees.aggregate([
  {
    $project: {
      name: 1,
      department: 1,
      rating:{$convert:{input:"$rating",to:"int"}}
    },
  },
  { $group: { _id: "$department", avg: { $avg: "$rating" } } },
  {$out:"ratingByDep"}
]);
```

Views

```
db.createView(  
    "activeUsers",  
    "users",  
    [  
        { $match: { isActive: true } },  
    ]  
)  
db.activeUsers.find()  
db.activeUsers.drop()
```

LMS case study - 1

```
//use lms  
db.users.insertOne({  
    _id: "u1",  
    name: "Rahul Dev",  
    email: "rahul@gmail.com",  
    password: "1234",  
    role: "student", // student | instructor | admin  
});
```

LMS case study - 2

```
db.users.insertOne({  
  _id: "u2",  
  name: "Aryan",  
  email: "aryan@gmail.com",  
  password: "1234",  
  role: "instructor", // student | instructor | admin  
});
```

LMS case study - 3

```
db.users.insertOne({  
  _id: "u3",  
  name: "admin",  
  email: "admin@gmail.com",  
  password: "1234",  
  role: "admin", // student | instructor | admin  
});
```

LMS case study - 4

```
//course collection  
db.courses.insertOne({  
    _id: "c1",  
    title: "MongoDB for Beginners",  
    description: "Learn MongoDB from scratch",  
    instructorId: "u2",  
    price: 1999,  
});
```

LMS case study - 5

```
db.courses.insertOne({  
  _id: "c2",  
  title: "Nodejs for Beginners",  
  description: "Learn Nodejs",  
  instructorId: "u2",  
  price: 2000,  
});
```

LMS case study - 6

```
//modules collection  
db.modules.insertOne({  
    _id: "m1",  
    courseId: "c1",  
    title: "Introduction to MongoDB",  
    order: 1,  
});
```

LMS case study - 7

```
db.modules.insertOne({  
  _id: "m2",  
  courseld: "c1",  
  title: "CRUD Operation",  
  order: 2,  
});
```

LMS case study - 8

```
db.modules.insertOne({  
  _id: "m3",  
  courseId: "c1",  
  title: "Aggregate Pipelines",  
  order: 3,  
});
```

LMS case study - 9

```
//lesson collection  
db.lessons.insertOne({  
    _id: "l1",  
    moduleId: "m1",  
    title: "What is MongoDB?",  
    description: "MongoDB is a document database.",  
    order: 1,  
});
```

LMS case study - 10

```
//enrollment collection  
db.enrollments.insertOne({  
    studentId: "u1",  
    courseId: "c1",  
});
```

LMS case study - 11

```
//lesson progress collection  
db.lessonProgress.insertOne({  
    studentId: "u1",  
    lessonId: "l1",  
    isCompleted: true,  
});
```

LMS case study - 12

```
//quizzes collection  
  
db.quizzes.insertOne({  
  lessonId: "l1",  
  questions: [  
    {  
      question: "MongoDB is?",  
      options: ["SQL DB", "NoSQL DB", "File System"],  
      correctAnswer: "NoSQL DB",  
    },  
  ],  
});
```

Aggregation - \$convert

```
db.employees.aggregate([
  {
    $project: {
      name: 1,
      department: 1,
      salaryStr: { $convert: { input: "$salary", to: "int" } },
    },
  },
  { $group: { _id: "$department", total: { $sum: "$salaryStr" } } },
]);
```

Aggregation - \$convert, group, total:0

```
db.employees.aggregate([
  {
    $project: {
      name: 1,
      department: 1,
      salaryStr: { $convert: { input: "$salary", to: "string" } },
    },
  },
  { $group: { _id: "$department", total: { $sum: "$salaryStr" } } },
]);
```

Backup and Restore - Tool

Download MSI version using below link:

<https://www.mongodb.com/try/download/database-tools>

Click on the downloaded file and install

Setup environment variables to add path

C:\Program Files\MongoDB\Tools\100\bin

Backup Steps

//backup of a particular database

mongodump -d mydb -o d:/bck //d means data

//backup of a particular collection

mongodump -d mydb -c employees -o d:/bck //c means collection

//backup of all the databases

mongodump -o d:/bck //o means output

Restore Steps

```
//to restore a particular database  
mongorestore -d mydb d:/bck/mydb
```

```
//to restore a particular collection  
mongorestore -d mydb -c employees d:\bck\mydb\employees.bson
```

```
//to restore all the databases  
mongorestore --dir d:\bck\
```

```
//creates a new database and then restores  
mongorestore -d mydbnew -c employees d:\bck\mydb\employees.bson
```

```
//creates a new collection and then restores  
mongorestore -d mydbnew -c employees d:\bck\mydb\employees.bson
```