**Learn MongoDB Queries**

**To clear the screen**

cls

**To create a database “library” if it does not exist or switch to the existing “library” database**

use library

**To check currently selected/active database**

db

**To remove the database**

db.dropDatabase()

**To view collections**

show collections

**To create “books” collection**

db.createCollection(“books”)

**To create “books” collection (if it does not exist) and insert single document**

db.books.insertOne([

{

title: "The Pragmatic Programmer",

author: "Andrew Hunt",

genres: ["Programming", "Software Engineering"],

year: 1999,

price: 45.99,

stock: 12,

ratings: [5, 4, 5, 5, 4]

},

])

**To create “books” collection (if it does not exist) and insert multiple documents**

db.books.insertMany([

{

title: "The Pragmatic Programmer",

author: "Andrew Hunt",

genres: ["Programming", "Software Engineering"],

year: 1999,

price: 45.99,

stock: 12,

ratings: [5, 4, 5, 5, 4]

},

{

title: "Clean Code",

author: "Robert C. Martin",

genres: ["Programming"],

year: 2008,

price: 40.00,

stock: 5,

ratings: [5, 5, 4]

},

{

title: "The Clean Coder",

author: "Robert C. Martin",

genres: ["Programming", "Career"],

year: 2011,

price: 38.00,

stock: 8,

ratings: [4, 3, 4]

},

{

title: "Deep Work",

author: "Cal Newport",

genres: ["Productivity", "Self-help"],

year: 2016,

price: 25.00,

stock: 3,

ratings: [5, 5, 5, 5]

},

{

title: "Atomic Habits",

author: "James Clear",

genres: ["Self-help", "Psychology"],

year: 2018,

price: 22.50,

stock: 10,

ratings: [5, 4, 5, 5, 5]

}

])

### **Count number of documents in the specified collection**

db.books.countDocuments()

### 

### **Find All Books**

db.books.find()

### **Find Books by Author**

db.books.find({ author: "Robert C. Martin" })

### **Find Books Published After 2010**

db.books.find({ year: { $gt: 2010 } })

### **Find Books with price $38 and published in the year 2011**

db.books.find({

$and: [

{year: 2011},

{price: 38}

]

})

### **Find Books either published in the year 2011 or 2017**

db.books.find({

$or: [

{year: 2011},

{year: 2017}

]

})

### **Find Books Published between 2010 and 2017 years**

db.books.find({

$and: [

{year: { $gte: 2010}},

{year: { $lte: 2017}}

]

})

### **Books in Stock Less Than 5**

db.books.find({ stock: { $lt: 5 } })

db.books.find( {stock: { $lt: 5 } }, {title:1, stock:1, author:1, \_id:0})

### **Find Books in the “Programming” Genre**

db.books.find({ genres: "Programming" })

**Find documents where genres includes both "Programming" and "Career":**

db.books.find({ genres: { $all: ["Programming", "Career"] } })

## **Projections (Show Specific Fields)**

### **Show Title and Price Only**

db.books.find({}, { title: 1, price: 1, \_id: 0 })

**Update the price of the book titled "Clean Code" to $42.00:**

db.books.find({title: "Clean Code"}, {title:1, price: 1})

db.books.updateOne(

{ title: "Clean Code" },

{ $set: { price: 42.00 } }

)

### **Increase Stock of All Programming Books by 5**

db.books.find({genres: "Programming"}, {title:1, stock: 1})

db.books.updateMany(

{ genres: "Programming" },

{ $inc: { stock: 5 } }

)

### **Rename Field from “stock” to “inventory”**

db.books.updateMany(

{},

{ $rename: { "stock": "inventory" } }

)

db.books.find()

db.books.updateMany(

{},

{ $rename: { "inventory": "stock" } }

)

**Add a bestseller: true field to all books published after 2015:**

db.books.find({year: {$gt: 2015}}, {title: 1})

db.books.updateMany(

{ year: { $gt: 2015 } },

{ $set: { bestseller: true } }

)

**Remove the ratings field from all books:**

db.books.updateMany(

{},

{ $unset: { ratings: "" } }

)

### **Delete One Book by Title**

db.books.find({}, {title:1})

db.books.deleteOne({ title: "The Clean Coder" })

db.books.find({}, {title:1})

db.books.insertOne([

{

title: "The Clean Coder",

author: "Robert C. Martin",

genres: ["Programming", "Career"],

year: 2011,

price: 38.00,

stock: 8,

ratings: [4, 3, 4]

}

])

**Delete books priced under $30.00:**

db.books.deleteMany({ price: { $lt: 30.00 } })

### **Delete All Books from a Specific Author**

db.books.deleteMany({ author: "Cal Newport" })

### **Drop the Entire Collection**

**Warning**: This removes all documents and indexes:

db.books.drop()

## **Sorting and Limiting**

### **Sort by Price (Descending)**

db.books.find().sort({ price: -1 })

db.books.find({}, {author:1, price:1, \_id:0}).sort({ price: -1 })

### **Top 3 Cheapest Books**

db.books.find({}, {title:1, price: 1, \_id:0})

db.books.find({}, {title:1, price: 1, \_id:0}).sort({ price: 1 }).limit(3)

db.books.find().sort({ price: 1 }).limit(3)

The **aggregation pipeline** is a **framework for data processing** in MongoDB. It processes documents through a sequence of stages, where each stage transforms the documents in some way.

Each stage takes in documents, processes them, and passes the result to the next stage — like an assembly line.

db.collection.aggregate([

{ stage1 },

{ stage2 },

{ stage3 },

...

])

| **Stage** | **Purpose** | **Example Use** |
| --- | --- | --- |
| $match | Filters documents (like find) | Match by author, genre, etc. |
| $project | Includes/excludes fields, computes new ones | Show only title and price; compute stockValue |
| $group | Groups by a field and performs aggregation | Group by author and count books |
| $sort | Sorts the documents | Sort by price or rating |
| $limit | Limits number of documents | Return top 5 books |
| $skip | Skips a number of documents | Pagination |
| $unwind | Flattens arrays into separate documents | Handle genres or tags |
| $lookup | Joins documents from another collection (like JOIN) | Combine books with authors |
| $count | Counts number of documents | Total books in a genre |
| $addFields | Adds new computed fields (like $project, but additive) | Add discountedPrice field |

## 

### 

### **Find books published after 2010:**

db.books.aggregate([

{ $match: { year: { $gt: 2010 } } }

])

### **Show only title and price of all the books:**

db.books.aggregate([

{ $project: { title: 1, price: 1, \_id: 0 } }

])

### **Show total books in the library:**

db.books.aggregate([

{ $count: "TotalBooks" }

])

### In MongoDB, the **$group stage** in the aggregation pipeline is used to group documents by a specified identifier and perform aggregation operations such as summing, averaging, counting, etc. It is similar to SQL's **GROUP BY**.

### Here are the most commonly used operators in the **$group** stage:

### **Accumulator Operators**

These operators compute values from multiple documents in each group:

| **Operator** | **Description** |
| --- | --- |
| **$sum** | Calculates the sum of numeric values. You can use 1 to count documents. |
| **$avg** | Calculates the average of numeric values. |
| **$min** | Returns the minimum value. |
| **$max** | Returns the maximum value. |
| **$first** | Returns the first value in the group (based on pipeline order). |
| **$last** | Returns the last value in the group (based on pipeline order). |
| **$push** | Appends values to an array. |
| **$addToSet** | Adds unique values to a set (no duplicates). |
| **$stdDevPop** | Calculates population standard deviation. |
| **$stdDevSamp** | Calculates sample standard deviation. |

### **Example**

db.sales.aggregate([

{

$group: {

\_id: "$region",

totalSales: { $sum: "$amount" },

avgSales: { $avg: "$amount" },

minSale: { $min: "$amount" },

maxSale: { $max: "$amount" },

salesList: { $push: "$amount" },

uniqueProducts: { $addToSet: "$product" },

count: { $sum: 1 }

}

}

])

### **Notes**

* The **\_id** field is **mandatory** in $group. It specifies the field(s) to group by.
* You can use compound fields like { \_id: { year: "$year", month: "$month" } } to group by multiple keys.
* **$group** does **not preserve the order** of documents.

### **Average Price of All Books**

db.books.find({}, {price: 1, \_id:0})

db.books.aggregate([

{ $group: { \_id: null, avgPrice: { $avg: "$price" } } }

])

### **Total Price of All Books**

db.books.find({}, {price: 1, \_id:0})

db.books.aggregate([

{ $group: { \_id: null, TotalPrice: { $sum: "$price" } } }

])

### **Total Price of All the Books written by Robert C. Martin**

db.books.find({author:'Robert C. Martin'},{title:1, author:1, price:1, \_id:0})

db.books.aggregate([

{ $match: { author: 'Robert C. Martin'} },

{ $group: { \_id: null, TotalPrice: { $sum: "$price" } } }

])

### **Number of Books per Author**

db.books.aggregate([

{ $group: { \_id: "$author", count: { $sum: 1 } } }

])

### **Average Rating per Book**

db.books.aggregate([

{

$project: {

title: 1,

avgRating: { $avg: "$ratings" }

}

}

])

## **Purpose of $project**

* Shape the output documents by specifying:  
  + Which fields to **include/exclude**
  + How to **rename fields**
  + Create **computed fields**
  + Apply **expressions** and **operators**

## **Operators Available in $project**

### **1. Arithmetic Operators**

| **Operator** | **Description** |
| --- | --- |
| **$add** | Adds numbers or dates |
| **$subtract** | Subtracts two numbers or dates |
| **$multiply** | Multiplies numbers |
| **$divide** | Divides numbers |
| **$mod** | Computes remainder of division |

**Example:**

**{ totalPrice: { $multiply: ["$price", "$quantity"] } }**

### **2. String Operators**

| **Operator** | **Description** |
| --- | --- |
| **$concat** | Concatenates strings |
| **$substr / $substrBytes** | Extracts substring |
| **$toUpper, $toLower** | Converts case |
| **$trim, $ltrim, $rtrim** | Trims whitespace |
| **$strLenBytes, $strLenCP** | Length of strings |

**Example:**

**{ fullName: { $concat: ["$firstName", " ", "$lastName"] } }**

### **3. Date Operators**

| **Operator** | **Description** |
| --- | --- |
| **$year, $month, $dayOfMonth, $dayOfWeek, $hour, $minute, $second** | Extract parts of a date |
| **$dateToString** | Format date as string |
| **$subtract / $add** | Can be used to add/subtract dates |

**Example:**

**{ borrowYear: { $year: "$borrowDate" } }**

### **4. Conditional and Comparison Operators**

| **Operator** | **Description** |
| --- | --- |
| **$cond** | If-then-else logic |
| **$ifNull** | If value is null, replace it |
| **$eq, $ne, $gt, $gte, $lt, $lte** | Comparison expressions |

**Example:**

**{ lateFee: { $cond: [ { $gt: ["$daysLate", 0] }, 50, 0 ] } }**

### **5. Array Operators**

| **Operator** | **Description** |
| --- | --- |
| **$arrayElemAt** | Access element by index |
| **$size** | Length of an array |
| **$slice** | Get a portion of an array |
| **$filter** | Filter items in array |
| **$map** | Transform each item in array |

**Example:**

**{ firstBorrower: { $arrayElemAt: ["$borrowers", 0] } }**

### **6. Type Conversion Operators (MongoDB 4.0+)**

| **Operator** | **Description** |
| --- | --- |
| **$toInt, $toDouble, $toString, $toDate, $toBool** | Convert between types |
| **$convert** | General type conversion |

**Example:**

**{ priceAsString: { $toString: "$price" } }**

### **7. Field Inclusion/Exclusion**

{

$project: {

title: 1, // include

author: 1,

internalNotes: 0 // exclude

}

}

* You **cannot mix** inclusion (1) and exclusion (0) in the same stage, **except** for \_id.

## **Complete Example**

db.books.aggregate([

{

$project: {

\_id: 0,

title: 1,

author: 1,

fullInfo: { $concat: ["$title", " by ", "$author"] },

borrowYear: { $year: "$borrowDate" },

lateFee: { $cond: [ { $gt: ["$daysLate", 0] }, 50, 0 ] }

}

}

])

## **$project – Shape the output**

### **Show only title and stock value (price × stock):**

db.books.aggregate([

{

$project: {

title: 1,

stockValue: { $multiply: ["$price", "$stock"] }

}

}

])

## **$group – Group and aggregate**

### **Count books by author:**

db.books.aggregate([

{

$group: {

\_id: "$author",

bookCount: { $sum: 1 }

}

}

])

### 

## **$sort – Sort the documents**

### **Sort books by price (descending):**

db.books.aggregate([

{ $sort: { price: -1 } }

])

### 

## **$addFields – Add computed fields**

### **Add a field discountedPrice with 10% discount:**

db.books.aggregate([

{

$addFields: {

discountedPrice: { $multiply: ["$price", 0.9] }

}

}

])

## **$count – Count documents**

### **Count total number of books:**

db.books.aggregate([

{ $count: "totalBooks" }

])

## **$unwind – Flatten array values**

### **Get one document per genre per book:**

db.books.aggregate([

{ $unwind: "$genres" },

{ $project: { title: 1, genre: "$genres" } }

])

## **$lookup – Join with another collection**

**Let’s create a Publisher collection:**

db.publishers.insertMany([

{

bookTitle: "Clean Code",

publisher: "Pearson",

country: "USA",

yearPublished: 2008

},

{

bookTitle: "The Pragmatic Programmer",

publisher: "Addison-Wesley",

country: "USA",

yearPublished: 1999

},

{

bookTitle: "Deep Work",

publisher: "Grand Central Publishing",

country: "USA",

yearPublished: 2016

},

{

bookTitle: "Atomic Habits",

publisher: "Avery",

country: "USA",

yearPublished: 2018

},

{

bookTitle: "The Clean Coder",

publisher: "Pearson",

country: "USA",

yearPublished: 2011

}

])

### **Join books with publishers:**

db.books.aggregate([

{

$lookup: {

from: "publishers",

localField: "title",

foreignField: "bookTitle",

as: "publisherInfo"

}

}

])

db.books.aggregate([

{

$lookup: {

from: "publishers",

localField: "title",

foreignField: "bookTitle",

as: "publisherInfo"

}

},

{$limit: 1}

])

db.books.aggregate([

{

$lookup: {

from: "publishers",

localField: "title",

foreignField: "bookTitle",

as: "publisherInfo"

}

},

{

$project: {

\_id: 0,

title: 1,

author: 1,

price: 1,

publisher: { $arrayElemAt: ["$publisherInfo.publisher", 0] },

yearPublished: { $arrayElemAt: ["$publisherInfo.yearPublished", 0] }

}

},

{

$limit: 1

}

])

## 

## 

## **$limit and $skip – Pagination**

### **Get the first 3 most expensive books:**

db.books.aggregate([

{ $sort: { price: -1 } },

{ $limit: 3 }

])

db.books.aggregate([

{ $project: { title: 1, price: 1, \_id:0 } },

{ $sort: { price: -1 } },

{ $limit: 4 }

])

### 

### **Skip the first 2 and return the next 2:**

db.books.aggregate([

{ $project: { title: 1, price: 1, \_id:0 } },

{ $sort: { price: -1 } },

{ $skip: 2 },

{ $limit: 2 }

])

## **Combine Multiple Stages**

### **For books published after 2010, compute stock value, group by author, and sort by total value:**

db.books.find({year: {$gt: 2000}}, {title:1, year:1, price:1, stock:1, \_id:0})

db.books.aggregate([

{ $match: { year: { $gt: 2000 } } },

{

$project: {

author: 1,

stockValue: { $multiply: ["$price", "$stock"] }

}

},

{

$group: {

\_id: "$author",

totalStockValue: { $sum: "$stockValue" }

}

},

{ $sort: { totalStockValue: -1 } }

])

### **Average Price of All Books**

db.books.aggregate([

{ $group: { \_id: null, avgPrice: { $avg: "$price" } } }

])

### **Total Stock Value (price \* stock) of each book**

db.books.aggregate([

{

$project: {

title: 1,

stockValue: { $multiply: ["$price", "$stock"] }

}

}

])

## **Total stock value of all the books in the library**

## db.books.aggregate([

## {

## $project: {

## stockValue: { $multiply: ["$price", "$stock"] }

## }

## },

## {

## $group: {

## \_id: null,

## totalLibraryStockValue: { $sum: "$stockValue" }

## }

## }

## ])

## 