https://www.mongodb.com/cloud/atlas/register -> Signup

https://www.mongodb.com/library/search/intro-monogdb-atlas?lb-mode=overlay -> Tutorial

Sure! Here's a structured, concise guide that covers basic MongoDB queries, documents, collections, indexes, and aggregation. This should serve as a quick reference for writing MongoDB programs from scratch.

**MongoDB Quick Reference Guide**

**Basic Definitions**

**Cluster**

* **Definition**: A cluster is a collection of MongoDB databases managed together. It ensures high availability and scalability.
* **Example**: A MongoDB Atlas cluster hosting multiple databases.

**Database**

* **Definition**: A logical container for collections.
* **Example**: use sports; switches to the sports database.

**Collection**

* **Definition**: A group of MongoDB documents, similar to a table in a relational database.
* **Example**: players and stadiums are collections within the sports database.

**Document**

* **Definition**: A record in a MongoDB collection, stored in BSON format (Binary JSON).
* **Example**:

json

Copy code

{

"\_id": ObjectId("..."),

"name": "John Doe",

"age": 25,

"team": "Team A",

"position": "Forward"

}

**Index**

* **Definition**: A data structure that improves the speed of data retrieval operations on a collection.
* **Example**: db.players.createIndex({ name: 1 })

**What is an Accumulator?**

**Imagine you have a big jar where you keep all your candies. Each time you add more candies to the jar, you can do different things to find out more about your candies. This jar and what you do with it is similar to what an accumulator does in MongoDB.**

**Accumulator Explained Like You’re 5**

1. **Counting Candies ($sum)**
   * **Example**: You want to count how many candies you have in your jar.
   * **MongoDB Example**:

javascript

Copy code

db.candies.aggregate([

{ $group: { \_id: null, totalCandies: { $sum: 1 } } }

]);

* + **Explanation**: The $sum accumulator counts the number of candies (documents) in your jar (collection).

1. **Finding the Biggest Candy ($max)**
   * **Example**: You want to find the biggest candy you have.
   * **MongoDB Example**:

javascript

Copy code

db.candies.aggregate([

{ $group: { \_id: null, biggestCandy: { $max: "$size" } } }

]);

* + **Explanation**: The $max accumulator finds the biggest candy (maximum value in a field).

1. **Finding the Smallest Candy ($min)**
   * **Example**: You want to find the smallest candy you have.
   * **MongoDB Example**:

javascript

Copy code

db.candies.aggregate([

{ $group: { \_id: null, smallestCandy: { $min: "$size" } } }

]);

* + **Explanation**: The $min accumulator finds the smallest candy (minimum value in a field).

1. **Finding the Average Candy Size ($avg)**
   * **Example**: You want to find out the average size of your candies.
   * **MongoDB Example**:

javascript

Copy code

db.candies.aggregate([

{ $group: { \_id: null, averageCandySize: { $avg: "$size" } } }

]);

* + **Explanation**: The $avg accumulator calculates the average size of your candies.

1. **Putting Similar Candies Together ($push)**
   * **Example**: You want to list all candies of the same color together.
   * **MongoDB Example**:

javascript

Copy code

db.candies.aggregate([

{ $group: { \_id: "$color", candies: { $push: "$name" } } }

]);

* + **Explanation**: The $push accumulator gathers all the candies of the same color into a group.

**Summary**

* **Accumulator**: A tool in MongoDB that helps you count, find the biggest or smallest, calculate the average, or group similar things together.
* **Think of it like**: A way to manage and understand all your candies (data) in different useful ways.

**Just like you use different ways to count or sort your candies, MongoDB uses accumulators to count, sort, and analyze data.**

4o

**Basic Queries**

**Insert Documents**

* **Insert One**:

javascript

Copy code

db.players.insertOne({

"name": "John Doe",

"age": 25,

"team": "Team A",

"position": "Forward"

});

* **Insert Many**:

javascript

Copy code

db.players.insertMany([

{ "name": "John Doe", "age": 25, "team": "Team A", "position": "Forward" },

{ "name": "Jane Smith", "age": 22, "team": "Team B", "position": "Midfielder" }

]);

**Find Documents**

* **Find All**:

javascript

Copy code

db.players.find();

* **Find with Criteria**:

javascript

Copy code

db.players.find({ team: "Team A" });

**Update Documents**

* **Update One**:

javascript

Copy code

db.players.updateOne(

{ name: "John Doe" },

{ $set: { age: 26 } }

);

* **Update Many**:

javascript

Copy code

db.players.updateMany(

{ team: "Team A" },

{ $set: { team: "Team C" } }

);

**Delete Documents**

* **Delete One**:

javascript

Copy code

db.players.deleteOne({ name: "John Doe" });

* **Delete Many**:

javascript

Copy code

db.players.deleteMany({ team: "Team A" });

**Aggregation Framework**

**Aggregation Pipeline**

* **Definition**: A multi-stage pipeline that processes documents and returns computed results.
* **Example**:

javascript

Copy code

db.players.aggregate([

{ $match: { age: { $gt: 20 } } },

{ $group: { \_id: "$team", averageAge: { $avg: "$age" } } },

{ $sort: { averageAge: -1 } }

]);

**Aggregate Functions**

* **$sum**: Calculates the sum of numeric values.

javascript

Copy code

db.stadiums.aggregate([

{ $group: { \_id: null, totalCapacity: { $sum: "$capacity" } } }

]);

**Common Aggregation Operators**

* **$match**: Filters documents to pass only the ones that match the specified condition(s).
* **$group**: Groups input documents by a specified identifier expression and applies the accumulator expressions.
* **$sort**: Sorts all input documents and returns them in the order specified by the sort key.

 1 for ascending order.

*  -1 for descending order.

**Embedded Documents**

**Definition**

* **Embedded Documents**: Documents nested within other documents to store related data together.
* **Example**:

json

Copy code

{

"name": "John Doe",

"age": 25,

"team": "Team A",

"position": "Forward",

"statistics": {

"matchesPlayed": 30,

"goalsScored": 10,

"assists": 5

}

}

**Example Use Cases**

1. **Find All Players in a Specific Team**:

javascript

Copy code

db.players.find({ team: "Team A" });

1. **Calculate Total Capacity of All Stadiums**:

javascript

Copy code

db.stadiums.aggregate([

{ $group: { \_id: null, totalCapacity: { $sum: "$capacity" } } }

]);

1. **Get Average Age of Players by Team**:

javascript

Copy code

db.players.aggregate([

{ $group: { \_id: "$team", avgAge: { $avg: "$age" } } }

]);

1. **Update Player Information**:

javascript

Copy code

db.players.updateOne(

{ name: "Jane Smith" },

{ $set: { age: 23, team: "Team C" } }

);

1. **Delete All Players from a Specific Team**:

javascript

Copy code

db.players.deleteMany({ team: "Team B" });

This guide should help you quickly reference MongoDB basics, ensuring you're prepared for both practical applications and interview questions.

## Aggregation Framework

### Aggregation Pipeline

* **Definition**: A multi-stage pipeline that processes documents and returns computed results.
* **Example**:

javascript

Copy code

db.players.aggregate([

{ $match: { age: { $gt: 20 } } },

{ $group: { \_id: "$team", averageAge: { $avg: "$age" } } },

{ $sort: { averageAge: -1 } }

]);

### Aggregation Operators Syntax

#### 1. $match

* **Purpose**: Filters documents to pass only those that match the specified condition(s).
* **Syntax**:

javascript

Copy code

{ $match: { <query> } }

* **Example**:

javascript

Copy code

db.players.aggregate([

{ $match: { team: "Team A" } }

]);

#### 2. $group

* **Purpose**: Groups input documents by the specified identifier expression and applies the accumulator expressions.
* **Syntax**:

javascript

Copy code

{

$group: {

\_id: <expression>, // Grouping key

<field1>: { <accumulator1>: <expression1> },

<field2>: { <accumulator2>: <expression2> },

...

}

}

* **Example**:

javascript

Copy code

db.players.aggregate([

{ $group: { \_id: "$team", totalPlayers: { $sum: 1 } } }

]);

#### 3. $sort

* **Purpose**: Sorts all input documents and returns them in the order specified by the sort key.
* **Syntax**:

javascript

Copy code

{ $sort: { <field>: <sort order> } }

* **Sort Order**:
  + 1 for ascending order.
  + -1 for descending order.
* **Example**:

javascript

Copy code

db.players.aggregate([

{ $sort: { age: 1 } } // Sort by age in ascending order

]);

#### 4. $project

* **Purpose**: Reshapes each document in the stream by adding, removing, or updating fields.
* **Syntax**:

javascript

Copy code

{ $project: { <field1>: <expression1>, <field2>: <expression2>, ... } }

* **Project Inclusion/Exclusion**:
  + 1 to include a field.
  + 0 to exclude a field.
* **Example**:

javascript

Copy code

db.players.aggregate([

{ $project: { name: 1, team: 1, age: 1, \_id: 0 } } // Include only name, team, and age fields, exclude \_id

]);

#### 5. $limit

* **Purpose**: Limits the number of documents passed to the next stage in the pipeline.
* **Syntax**:

javascript

Copy code

{ $limit: <number> }

* **Example**:

javascript

Copy code

db.players.aggregate([

{ $limit: 5 } // Limit the result to 5 documents

]);

#### 6. $skip

* **Purpose**: Skips the first N documents and passes the remaining documents to the next stage in the pipeline.
* **Syntax**:

javascript

Copy code

{ $skip: <number> }

* **Example**:

javascript

Copy code

db.players.aggregate([

{ $skip: 10 } // Skip the first 10 documents

]);

#### 7. $lookup

* **Purpose**: Performs a left outer join to another collection in the same database to filter in documents from the "joined" collection for processing.
* **Syntax**:

javascript

Copy code

{

$lookup: {

from: <collection to join>,

localField: <field from the input documents>,

foreignField: <field from the documents of the "from" collection>,

as: <output array field>

}

}

* **Example**:

javascript

Copy code

db.players.aggregate([

{

$lookup: {

from: "teams",

localField: "team",

foreignField: "name",

as: "teamDetails"

}

}

]);

#### 8. $unwind

* **Purpose**: Deconstructs an array field from the input documents to output a document for each element of the array.
* **Syntax**:

javascript

Copy code

{ $unwind: { path: <field path>, includeArrayIndex: <string>, preserveNullAndEmptyArrays: <boolean> } }

* **Example**:

javascript

Copy code

db.players.aggregate([

{ $unwind: "$statistics" } // Deconstruct the statistics array field

]);

### Accumulators in $group

* **Accumulators** are operators used in the $group stage to perform calculations on grouped data.
* **Common Accumulators**:
  + $sum: Calculates the sum of numeric values.

javascript

Copy code

totalCapacity: { $sum: "$capacity" }

* + $avg: Calculates the average of numeric values.

javascript

Copy code

averageAge: { $avg: "$age" }

* + $max: Finds the maximum value.

javascript

Copy code

maxAge: { $max: "$age" }

* + $min: Finds the minimum value.

javascript

Copy code

minAge: { $min: "$age" }

* + $push: Pushes values into an array.

javascript

Copy code

playerNames: { $push: "$name" }

* + $addToSet: Adds values to an array, ensuring no duplicates.

javascript

Copy code

uniqueTeams: { $addToSet: "$team" }

### Practical Examples

1. **Find Average Age of Players by Team**:

javascript

Copy code

db.players.aggregate([

{ $group: { \_id: "$team", avgAge: { $avg: "$age" } } }

]);

1. **Sort Players by Age and Limit Results**:

javascript

Copy code

db.players.aggregate([

{ $sort: { age: -1 } }, // Sort by age in descending order

{ $limit: 5 } // Limit to top 5 oldest players

]);

1. **Project Specific Fields**:

javascript

Copy code

db.players.aggregate([

{ $project: { name: 1, age: 1, \_id: 0 }