**Aggregations** operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In SQL count(\*) and with group by is an equivalent of mongodb aggregation.

perform specific operations on the data, like filtering, grouping, and sorting.

db.COLLECTION\_NAME.**aggregate**(AGGREGATE\_OPERATION)

**Common Aggregation Stages**

* **$match**: Filters documents by specified criteria, similar to the find() method.($gt,etc)
* **$group**: Groups documents by a specified field and performs operations (like sum, avg, count) on them.
* **$sort**: Sorts documents by specified field(s).
* **$project**: Shapes the output, allowing you to include, exclude, or compute fields.
* **$limit** and **$skip**: Limits the number of documents and skips a specified number of documents.

Group karte samay hamesha id specify karni padti ki kis field pe group karre(jisko group kiya hai bas vo hi display hoga )

$project me jo jo field dikhane hai vo 1 karo (use when number of field dikhane hai eksath)

([ need to add ‘[‘ bcz no of aggregation pipeline can be implemented hence need to pass an array of pipelines like done in 4

### **1. Query: Find the total number of employees grouped by their positions.**

db.emp.aggregate([

{

$group: {

\_id: "$position",

totalEmployees: { $sum: 1 }

}

}

]);

{ "\_id" : "CEO", "totalEmployees" : 1 }

{ "\_id" : "Manager", "totalEmployees" : 1 }

{ "\_id" : "Developer", "totalEmployees" : 1 }

{ "\_id" : "Designer", "totalEmployees" : 1 }

{ "\_id" : "Analyst", "totalEmployees" : 1 }

{ "\_id" : "HR", "totalEmployees" : 1 }

2. Query: Find the average age of employees grouped by their office locations.

db.emp.aggregate ", "avgAge" : 35 }

{ "\_id" : "Bangalore", "avgAge" : 27 }

{ "\_id" : "Chennai", "avgAge" : 30 }

{ "\_id" : "Hyderabad", "avgAge" : 26 }

{ "\_id" : "Delhi", "avgAge" : 29 }

3. Query: List employees with their positions and ages in descending order of age.

db.emp.aggregate([

{

$project: {

name: 1,

position: 1,

age: 1

}

},

{ $sort: { age: -1 } }

]);

db.pra.aggregate([

{

$project:{

ph:1,

name:1,

office:1

}

},

{$sort:{ph:-1}}])

{ "\_id" : ObjectId("67061e69a40e3d447c2e0595"), "name" : "Rahul Desai", "position" : "Manager", "age" : 35 }

{ "\_id" : ObjectId("67061e69a40e3d447c2e0597"), "name" : "Vikram Singh", "position" : "Designer", "age" : 30 }

{ "\_id" : ObjectId("67061e69a40e3d447c2e0599"), "name" : "Kiran Nair", "position" : "HR", "age" : 29 }

{ "\_id" : ObjectId("67061e69a40e3d447c2e0596"), "name" : "Neha Verma", "position" : "Developer", "age" : 27 }

{ "\_id" : ObjectId("67061e69a40e3d447c2e0598"), "name" : "Anjali Rao", "position" : "Analyst", "age" : 26 }

4. Query: Find how many employees are there in each office.

db.emp.aggregate([

{

$group: {

\_id: "$office",

count: { $sum: 1 }

}

}

]);

{ "\_id" : "pune", "count" : 1 }

{ "\_id" : "Mumbai", "count" : 1 }

{ "\_id" : "Bangalore", "count" : 1 }

{ "\_id" : "Chennai", "count" : 1 }

{ "\_id" : "Hyderabad", "count" : 1 }

{ "\_id" : "Delhi", "count" : 1 }

5. Query: Find employees older than a specific age (e.g., 28) and sort them in ascending order by age.

db.emp.aggregate([

{

$match: {

age: { $gt: 28 }

}

},

{

$sort: { age: 1 }

}

]);

{ "\_id" : ObjectId("67061e69a40e3d447c2e0597"), "name" : "Vikram Singh", "office" : "Chennai", "addr" : "Greenfield Avenue, Chennai", "position" : "Designer", "age" : 30 }

{ "\_id" : ObjectId("67061e69a40e3d447c2e0595"), "name" : "Rahul Desai", "office" : "Mumbai", "addr" : "Sunrise Tower, Mumbai", "position" : "Manager", "age" : 35 }

{ "\_id" : ObjectId("67061e69a40e3d447c2e0599"), "name" : "Kiran Nair", "office" : "Delhi", "addr" : "Connaught Place, Delhi", "position" : "HR", "age" : 29 }

**Indexing** in MongoDB is a way to optimize query performance. Indexes are data structures that store a small portion of the data set in an easy-to-traverse form, allowing MongoDB to quickly find and retrieve relevant documents instead of scanning every document in a collection  
  
 **Single Field Index**: An index on a single field. Often used for simple queries.

javascript

Copy code

db.collection.createIndex({ field: 1 }) // Ascending index on `field`

 **Compound Index**: An index on multiple fields, useful for queries that involve filtering by multiple fields.

javascript

Copy code

db.collection.createIndex({ field1: 1, field2: -1 })

6. Query: Create a simple index on the position field.

db.emp.createIndex({ position: 1 });

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

7. Query: Create a unique index on the name field.

db.emp.createIndex({ name: 1 }, { unique: true });

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 2,

"numIndexesAfter" : 3,

"ok" : 1

}

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 2,

"numIndexesAfter" : 3,

"ok" : 1

}

8. Query: Show indexes present in the emp collection.

db.emp.getIndexes();

[

{ "v" : 2, "key" : { "\_id" : 1 }, "name" : "\_id\_", "ns" : "db.emp" },

{ "v" : 2, "key" : { "position" : 1 }, "name" : "position\_1", "ns" : "db.emp" },

{ "v" : 2, "unique" : true, "key" : { "name" : 1 }, "name" : "name\_1", "ns" : "db.emp" }

]

9. Query: Explain the execution status of finding an employee by name.

db.emp.find({ name: "A.P.J. Abdul Kalam" }).explain("executionStats");

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "db.emp",

"indexFilterSet" : false,

"parsedQuery" : { "name" : { "$eq" : "A.P.J. Abdul Kalam" } },

"winningPlan" : { "stage" : "EOF" },

"rejectedPlans" : []

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 0,

"executionTimeMillis" : 0,

"totalKeysExamined" : 0,

"totalDocsExamined" : 0,

"executionStages" : {

"stage" : "EOF",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 1,

"advanced" : 0,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1

}

},

"serverInfo" : {

"host" : "localhost.localdomain",

"port" : 27017,

"version" : "4.2.13",

"gitVersion" : "82dd40f60c55dae12426c08fd7150d79a0e28e23"

},

"ok" : 1

}

10. Query: Find the count of unique positions.

db.emp.aggregate([

{

$group: {

\_id: "$position"

}

},

{

$count: "distinctPositionsCount"

}

]);

{ "distinctPositionsCount": 6 }