MongoDB Aggregation Pipeline - Complete Example

The **Aggregation Pipeline** in MongoDB is a powerful framework for data processing and analysis. It allows you to process documents step by step using multiple stages.

Step 1: Create a Sample Collection

Let's create a collection called `employees` with some sample documents.

Insert Sample Data

```
db.employees.insertMany([
  { _id: 1, name: "Alice", department: "HR", salary: 50000, joinDate:
ISODate("2020-01-15") },
  { _id: 2, name: "Bob", department: "IT", salary: 75000, joinDate:
ISODate("2019-07-23") },
  { _id: 3, name: "Charlie", department: "Finance", salary: 62000, joinDate:
ISODate("2021-06-12") },
  { _id: 4, name: "David", department: "IT", salary: 80000, joinDate:
ISODate("2018-09-30") },
  { _id: 5, name: "Emma", department: "HR", salary: 52000, joinDate:
ISODate("2019-12-05") }
]);
```

Step 2: Understanding the Aggregation Pipeline

The Aggregation Pipeline consists of **multiple stages**, each performing an operation on the data. Some common stages are:

1. **$match** – Filters documents based on conditions.
2. **$group** – Groups documents together for calculations.
3. **$sort** – Sorts documents.
4. **$project** – Reshapes the document by including/excluding fields.
5. **$limit** – Limits the number of documents.
6. **$skip** – Skips a certain number of documents.
7. **$lookup** – Joins data from another collection.
8. **$unwind** – Deconstructs an array into multiple documents.

Step 3: Aggregation Pipeline Examples

Let's go through several real-world examples.

1. Calculate the Average Salary per Department

```
db.employees.aggregate([
  { $group: { _id: "$department", avgSalary: { $avg: "$salary" } } }
]);
```

Explanation:

- `$group`: Groups documents by the **department** field.
- `$avg`: Computes the average salary for each department.

Output:

```
[
  { "_id": "HR", "avgSalary": 51000 },
  { "_id": "IT", "avgSalary": 77500 },
  { "_id": "Finance", "avgSalary": 62000 }
]
```

---

2. Find Employees Who Joined After 2020

```
db.employees.aggregate([
  { $match: { joinDate: { $gt: ISODate("2020-01-01") } } }
]);
```

Explanation:

- `$match`: Filters employees who joined **after January 1, 2020**.

Output:

```
[
  { "_id": 3, "name": "Charlie", "department": "Finance", "salary": 62000,
"joinDate": "2021-06-12T00:00:00Z" }
]
```

---

3. Sort Employees by Salary (Descending)

```
db.employees.aggregate([
  { $sort: { salary: -1 } }
]);
```

Explanation:

- `$sort`: Sorts documents by **salary** in descending order.

Output:

```
[
  { "_id": 4, "name": "David", "department": "IT", "salary": 80000 },
  { "_id": 2, "name": "Bob", "department": "IT", "salary": 75000 },
  { "_id": 3, "name": "Charlie", "department": "Finance", "salary": 62000 },
  { "_id": 5, "name": "Emma", "department": "HR", "salary": 52000 },
  { "_id": 1, "name": "Alice", "department": "HR", "salary": 50000 }
]
```

---

4. Get the Highest Paid Employee in Each Department

```
db.employees.aggregate([
  { $sort: { salary: -1 } },
  { $group: { _id: "$department", highestPaid: { $first: "$$ROOT" } } }
]);
```

Explanation:

- `$sort`: Sorts employees by **salary** in descending order.
- `$group`: Groups by **department** and selects the first document (highest salary).

Output:

```
[
  { "_id": "HR", "highestPaid": { "name": "Emma", "salary": 52000 } },
  { "_id": "IT", "highestPaid": { "name": "David", "salary": 80000 } },
  { "_id": "Finance", "highestPaid": { "name": "Charlie", "salary": 62000 } }
]
```

---

5. Count Employees per Department

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

Explanation:

- `$group`: Groups documents by **department**.
- `$sum`: Counts the number of employees in each department.

Output:

```
[
  { "_id": "HR", "totalEmployees": 2 },
  { "_id": "IT", "totalEmployees": 2 },
```

```
  { "_id": "Finance", "totalEmployees": 1 }
]
```

---

6. Reshape the Output (Only Name and Salary)

```
db.employees.aggregate([
  { $project: { _id: 0, name: 1, salary: 1 } }
]);
```

Explanation:

- `$project`: Includes only **name** and **salary**, excluding `_id`.

Output:

```
json
CopyEdit
[
  { "name": "Alice", "salary": 50000 },
  { "name": "Bob", "salary": 75000 },
  { "name": "Charlie", "salary": 62000 },
  { "name": "David", "salary": 80000 },
  { "name": "Emma", "salary": 52000 }
]
```

---

7. Join with Another Collection (Using `$lookup`)

Assume we have another collection `departments`:

```
db.departments.insertMany([
  { dept: "HR", location: "New York" },
  { dept: "IT", location: "San Francisco" },
  { dept: "Finance", location: "Chicago" }
]);
```

Now, let's **join** `employees` with `departments`:

```
db.employees.aggregate([
  {
    $lookup: {
      from: "departments",
      localField: "department",
      foreignField: "dept",
      as: "deptInfo"
    }
  }
]);
```

Explanation:

- `$lookup`: Joins `employees` with `departments` on the `department` field.

Output:

```
[
  { "name": "Alice", "department": "HR", "deptInfo": [{ "dept": "HR",
"location": "New York" }] },
  { "name": "Bob", "department": "IT", "deptInfo": [{ "dept": "IT", "location":
"San Francisco" }] }
]
```

Summary

MongoDB's **Aggregation Pipeline** is a powerful tool for analyzing and transforming data.
We covered: ✅ Creating a collection
✅ Filtering (`$match`)
✅ Grouping (`$group`)
✅ Sorting (`$sort`)
✅ Reshaping (`$project`)
✅ Counting (`$sum`)
✅ Joining (`$lookup`)