

MongoDB Aggregation Framework – Beginner to Advanced

This guide explains the **MongoDB Aggregation Framework** using **only MongoDB concept**

1. Aggregation Framework Overview

The Aggregation Framework processes documents through a **pipeline of stages**. Each stage:

- Receives documents as input
- Transforms them
- Passes the result to the next stage

```
db.collection.aggregate([
  { stage1 },
  { stage2 },
  { stage3 }
])
```

Sample Database (Used in All Examples)

users collection

```
db.users.insertMany([
  { _id: 1, name: "Alice", age: 25, city: "New York", isActive: true },
  { _id: 2, name: "Bob", age: 30, city: "London", isActive: false },
  { _id: 3, name: "Charlie", age: 35, city: "New York", isActive: true },
  { _id: 4, name: "David", age: 28, city: "Paris", isActive: true }
])
```

orders collection

```
db.orders.insertMany([
  { _id: 101, userId: 1, product: "Laptop", amount: 1200, status: "completed" },
  { _id: 102, userId: 1, product: "Mouse", amount: 50, status: "completed" },
  { _id: 103, userId: 2, product: "Keyboard", amount: 100, status: "pending" },
  { _id: 104, userId: 3, product: "Monitor", amount: 300, status: "completed" }
])
```

2. `$match` - Filter Documents

Purpose

Filters documents based on conditions. Reduces data early in the pipeline.

Problem Statement

Retrieve specific subsets of documents based on conditions.

Example 1: Active users only

```
db.users.aggregate([
  { $match: { isActive: true } }
])
```

Explanation: Returns only users whose `isActive` value is true.

Example 2: Users older than 30

```
db.users.aggregate([
  { $match: { age: { $gt: 30 } } }
])
```

Explanation: Filters users where age is greater than 30.

Example 3: Users from New York

```
db.users.aggregate([
  { $match: { city: "New York" } }
])
```

Example 4: Active users from New York

```
db.users.aggregate([
  { $match: { city: "New York", isActive: true } }
])
```

Example 5: Users between age 25 and 35

```
db.users.aggregate([
  { $match: { age: { $gte: 25, $lte: 35 } } }
])
```

3. `$project` - Shape Documents

Purpose

Select, exclude, rename, or compute fields.

Problem Statement

Control what fields appear in the output.

Example 1: Show name and city only

```
db.users.aggregate([
  { $project: { name: 1, city: 1, _id: 0 } }
])
```

Example 2: Add ageNextYear field

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      ageNextYear: { $add: ["$age", 1] }
    }
  }
])
```

Example 3: Rename fields

```
db.users.aggregate([
  {
    $project: {
      userName: "$name",
      location: "$city"
    }
  }
])
```

Example 4: Conditional field

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      status: {
```

```
        $cond: [{ $eq: ["$isActive", true] }, "ACTIVE", "INACTIVE"]
    }
}
])
])
```

Example 5: Exclude age

```
db.users.aggregate([
  { $project: { age: 0 } }
])
```

4. **\$group** - Aggregate Data

Purpose

Group documents and perform calculations.

Problem Statement

Generate summarized results from multiple documents.

Example 1: Count users per city

```
db.users.aggregate([
  {
    $group: {
      _id: "$city",
      totalUsers: { $sum: 1 }
    }
  }
])
```

Example 2: Average age per city

```
db.users.aggregate([
  {
    $group: {
      _id: "$city",
      averageAge: { $avg: "$age" }
    }
  }
])
```

Example 3: Total users

```
db.users.aggregate([
  { $group: { _id: null, count: { $sum: 1 } } }
])
```

Example 4: Active users per city

```
db.users.aggregate([
  { $match: { isActive: true } },
  {
    $group: {
      _id: "$city",
      activeUsers: { $sum: 1 }
    }
  }
])
```

Example 5: Maximum age

```
db.users.aggregate([
  { $group: { _id: null, maxAge: { $max: "$age" } } }
])
```

5. \$sort, \$skip, \$limit

Purpose

Order and paginate results.

Example 1: Sort by age descending

```
db.users.aggregate([
  { $sort: { age: -1 } }
])
```

Example 2: Oldest two users

```
db.users.aggregate([
  { $sort: { age: -1 } },
  { $limit: 2 }
])
```

Example 3: Skip first user

```
db.users.aggregate([
  { $sort: { age: 1 } },
  { $skip: 1 }
])
```

Example 4: Pagination (page 2)

```
db.users.aggregate([
  { $sort: { age: 1 } },
  { $skip: 2 },
  { $limit: 2 }
])
```

Example 5: Sort by city then age

```
db.users.aggregate([
  { $sort: { city: 1, age: -1 } }
])
```

6. `$lookup` – Combine Collections

Purpose

Combine related documents from another collection.

Problem Statement

Attach related data stored in a separate collection.

Example 1: Attach orders to users

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

Example 2: Users with completed orders only

```
db.users.aggregate([
  {
    $lookup: {
      from: "orders",
      localField: "_id",
      foreignField: "userId",
      as: "orders"
    }
  },
  { $match: { "orders.status": "completed" } }
])
```

Example 3: Orders count per user

```
db.users.aggregate([
  { $lookup: { from: "orders", localField: "_id", foreignField: "userId",
    as: "orders" } },
  { $project: { name: 1, orderCount: { $size: "$orders" } } }
])
```

Example 4: Users without orders

```
db.users.aggregate([
  { $lookup: { from: "orders", localField: "_id", foreignField: "userId",
    as: "orders" } },
  { $match: { orders: { $eq: [] } } }
])
```

Example 5: Lookup with pipeline

```
db.users.aggregate([
  {
    $lookup: {
      from: "orders",
      let: { userId: "$_id" },
      pipeline: [
        { $match: { $expr: { $eq: ["$userId", "$$userId"] } } },
        { $match: { status: "completed" } }
      ],
      as: "completedOrders"
    }
  }
])
```

7. `$unwind` - Deconstruct Arrays

Purpose

Convert array elements into individual documents.

Example: One document per order

```
db.users.aggregate([
  { $lookup: { from: "orders", localField: "_id", foreignField: "userId",
    as: "orders" } },
  { $unwind: "$orders" }
])
```

8. Real-World Pipeline Example

Problem Statement

Calculate total completed order amount per user.

```
db.users.aggregate([
  { $lookup: { from: "orders", localField: "_id", foreignField: "userId",
    as: "orders" } },
  { $unwind: "$orders" },
  { $match: { "orders.status": "completed" } },
  {
    $group: {
      _id: "$name",
      totalSpent: { $sum: "$orders.amount" }
    }
  }
])
```

9. Accumulator & Date Expression Operators (Core Reference)

This section consolidates **commonly used accumulator operators** and **date expression operators** in one place. These operators are heavily used in `$group`, `$project`, and reporting-style pipelines.

9.1 Accumulator Operators

Accumulator operators process **multiple documents** and return a **single calculated value** per group.

Reference Table

Operator	Purpose	Where Used
\$sum	Calculates total or count	\$group , \$project
\$avg	Calculates average value	\$group
\$min	Finds minimum value	\$group
\$max	Finds maximum value	\$group
\$push	Collects values into array	\$group
\$addToSet	Collects unique values	\$group
\$multiply	Multiplies numeric values	\$project , \$group

\$sum

Problem Statement: Calculate totals or count documents.

Example 1: Count users

```
db.users.aggregate([
  { $group: { _id: null, totalUsers: { $sum: 1 } } }
])
```

Example 2: Total order amount per user

```
db.orders.aggregate([
  { $group: { _id: "$userId", totalAmount: { $sum: "$amount" } } }
])
```

Explanation:

- \$sum: 1 counts documents
- \$sum: "\$amount" adds numeric field values

\$avg

Problem Statement: Calculate average values.

```
db.users.aggregate([
  { $group: { _id: "$city", averageAge: { $avg: "$age" } } }
])
```

Explanation: Computes average age for each city group.

\$min and \$max

Problem Statement: Find boundary values.

```
db.users.aggregate([
  {
    $group: {
      _id: null,
      youngestUser: { $min: "$age" },
      oldestUser: { $max: "$age" }
    }
  }
])
```

\$multiply

Problem Statement: Calculate derived numeric values.

Example: Calculate tax-inclusive order amount (10%)

```
db.orders.aggregate([
  {
    $project: {
      product: 1,
      totalWithTax: { $multiply: ["$amount", 1.1] }
    }
  }
])
```

Explanation: Multiplies amount by tax factor.

9.2 Date Expression Operators

Date operators extract or transform date values from Date fields.

 These operators require fields stored as ISODate.

Reference Table

Operator	Purpose	Output
\$year	Extract year	Number
\$month	Extract month (1-12)	Number
\$dayOfMonth	Extract day	Number
\$hour	Extract hour	Number
\$minute	Extract minute	Number
\$dateToString	Format date	String

\$year and \$month

Problem Statement: Group records by year and month.

```
db.orders.aggregate([
  {
    $group: {
      _id: {
        year: { $year: "$createdAt" },
        month: { $month: "$createdAt" }
      },
      totalSales: { $sum: "$amount" }
    }
  }
])
```

Explanation:

- Extracts year and month from `createdAt`
- Groups documents accordingly

\$dayOfMonth

Problem Statement: Analyze daily activity.

```
db.orders.aggregate([
  {
    $group: {
      _id: { day: { $dayOfMonth: "$createdAt" } },
      orderCount: { $sum: 1 }
    }
  }
])
```

```
    }  
])
```

```
$dateToString
```

Problem Statement: Create formatted date labels for reports.

```
db.orders.aggregate([  
  {  
    $project: {  
      orderDate: {  
        $dateToString: { format: "%Y-%m-%d", date: "$createdAt" }  
      },  
      amount: 1  
    }  
  }  
])
```

9.3 Best Practices for Accumulators & Dates

- Always ensure numeric fields before using `$sum`, `$avg`, `$multiply`
- Use `$match` before `$group` to reduce dataset size
- Store dates as `ISODate`, not strings
- Use `$dateToString` only at the **final reporting stage**
- Prefer `$addToSet` over `$push` when duplicates are not allowed

10. `$facet` - Multi-Result Pipelines

Purpose

Run **multiple aggregation pipelines in parallel** on the same input data.

Problem Statement

Generate multiple reports in a single query.

```
db.users.aggregate([  
  {  
    $facet: {  
      activeUsers: [  
        { $match: { isActive: true } },  
        { $count: "count" }  
      ],  
    }  
  }  
])
```

```

        usersByCity: [
            { $group: { _id: "$city", total: { $sum: 1 } } }
        ]
    }
])

```

Explanation: - `activeUsers` and `usersByCity` run independently - Output is a single document with multiple result arrays

11. Array Operators - `$map`, `$filter`, `$reduce`

`$map`

Transform array elements.

```

db.users.aggregate([
{
    $project: {
        name: 1,
        upperCities: {
            $map: {
                input: ["$city"],
                as: "c",
                in: { $toUpper: "$$c" }
            }
        }
    }
})

```

`$filter`

Select array elements conditionally.

```

db.users.aggregate([
{
    $lookup: { from: "orders", localField: "_id", foreignField: "userId",
    as: "orders" }
},
{
    $project: {
        name: 1,
        completedOrders: {
            $filter: {

```

```

        input: "$orders",
        as: "o",
        cond: { $eq: ["$$o.status", "completed"] }
    }
}
]
)
])

```

\$reduce

Aggregate array values into a single result.

```

db.users.aggregate([
{
    $lookup: { from: "orders", localField: "_id", foreignField: "userId",
    as: "orders" }
},
{
    $project: {
        name: 1,
        totalSpent: {
            $reduce: {
                input: "$orders",
                initialValue: 0,
                in: { $add: ["$$value", "$$this.amount"] }
            }
        }
    }
}
])

```

12. Performance & Optimization Guidelines

- Always place `$match` as early as possible
- Use indexes on fields used in `$match` and `$lookup`
- Avoid unnecessary `$unwind`
- Use `$project` to limit fields early
- Prefer pipeline `$lookup` for complex joins

13. Interview Preparation Checklist

You should be comfortable with:

- Designing pipelines stage-by-stage
- Explaining why each stage exists
- Writing `$lookup` + `$group` pipelines
- Using accumulator and date operators
- Optimizing pipelines for performance

14. Final Notes

This document is designed to be a **complete MongoDB Aggregation reference** from beginner to advanced level.

Next recommended steps:

- Practice with real datasets
- Rebuild analytics queries using aggregation
- Prepare interview-style problem statements



You now have a **production-ready MongoDB Aggregation guide**.