15. Aggregation

Aggregation in MongoDB is a process of transforming and analyzing data stored in collections. It is similar to SQL's GROUP BY, SUM, JOIN, or WHERE clauses but is more flexible and powerful.

Uses of Aggregation:

1. Summarizing data (e.g., total sales, average age).
2. Restructuring data (e.g., filtering, grouping, or reshaping).
3. Analyzing data (e.g., performing statistical or conditional computations).

Aggregation Pipeline

The **aggregation pipeline** is a framework in MongoDB that allows the transformation of documents through a series of stages. Each stage performs a specific operation on the input documents and passes the transformed documents to the next stage.

Basic Syntax:

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

users Collection:

```
[
   {
     "_id": 1,
     "name": "Ajay",
     "age": 28,
     "gender": "Male",
     "hobbies": ["reading", "traveling", "yoga"],
     "scores": [40, 60, 80]
   },
   {
     "_id": 2,
     "name": "Sunita",
     "age": 25,
     "gender": "Female",
     "hobbies": ["painting", "gardening", "music"],
     "scores": [55, 45, 90]
   },
   {
     "_id": 3,
     "name": "Ravi",
     "age": 35,
     "gender": "Male",
     "hobbies": ["gaming", "science fiction", "technology"],
     "scores": [70, 85, 95]
   },
   {
     "_id": 4,
     "name": "Madhavi",
     "age": 30,
     "gender": "Female",
     "hobbies": ["cooking", "dance", "fitness"],
     "scores": [30, 50, 70]
   },
   {
     "_id": 5,
     "name": "Sai",
     "age": 40,
     "gender": "Male",
     "hobbies": ["history", "painting", "walking"],
     "scores": [90, 95, 100]
   }
  ]
```

orders Collection:

```
[
  {
    "_id": 101,
    "orderId": "ORD001",
    "customerId": 1,
    "amount": 500,
    "status": "completed",
    "orderDate": "2024-11-20T10:00:00Z",
    "items": [
      { "product": "Book", "quantity": 2, "price": 250 }
    ]
  },
  {
    "_id": 102,
    "orderId": "ORD002",
    "customerId": 2,
    "amount": 150,
    "status": "pending",
    "orderDate": "2024-11-21T12:00:00Z",
    "items": [
      { "product": "Pen", "quantity": 10, "price": 15 }
    ]
  },
  {
    "_id": 103,
    "orderId": "ORD003",
    "customerId": 3,
    "amount": 300,
    "status": "completed",
    "orderDate": "2024-11-22T15:00:00Z",
    "items": [
      { "product": "Laptop", "quantity": 1, "price": 300 }
    ]
  },
  {
    "_id": 104,
    "orderId": "ORD004",
    "customerId": 1,
    "amount": 200,
    "status": "completed",
    "orderDate": "2024-11-23T14:00:00Z",
    "items": [
      { "product": "Headphones", "quantity": 1, "price": 200 }
    ]
  },
  {
    "_id": 105,
    "orderId": "ORD005",
    "customerId": 4,
    "amount": 100,
    "status": "canceled",
    "orderDate": "2024-11-24T16:00:00Z",
    "items": [
      { "product": "Notebook", "quantity": 5, "price": 20 }
    ]
  }
 ]
```

Key Aggregation Stages

1. $match

Filters documents based on specified conditions. It's equivalent to the WHERE clause in SQL.

Example: Filter users with age greater than 25:

```
db.users.aggregate([
  { $match: { age: { $gt: 25 } } }
  ]);
```

2. $group

Groups documents by a specified key and applies accumulator expressions to compute aggregated values for each group.

Example: Group users by gender and count the number of users in each group:

```

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

Accumulators in $group:

- $sum: Calculates the sum of numeric values.
- $avg: Computes the average of numeric values.
- $min: Returns the minimum value.
- $max: Returns the maximum value.
- $push: Adds values to an array.
- $addToSet: Adds unique values to an array.

3. $project

Reshapes documents by including, excluding, or computing new fields.

Example: Include only name and age, and compute a new field isAdult:

```
db.users.aggregate([
   {
     $project: {
       _id: 0,
       name: 1,
       age: 1,
       isAdult: { $gte: ["$age", 18] }
     }
   }
  ]);
```

4. $sort

Sorts documents by a specified field.

Example: Sort users by age in descending order:

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

5. $limit

Limits the number of documents in the output.

Example: Retrieve the first 5 documents:

```
db.users.aggregate([
   { $limit: 5 }
 ]);
```

6. $skip

Skips a specified number of documents.

Example: Skip the first 10 documents:

```
db.users.aggregate([
   { $skip: 10 }
 ]);
```

7. $unwind

Deconstructs an array field into multiple documents, one for each element in the array.

Example: Unwind the hobbies array:

```
db.users.aggregate([
   { $unwind: "$hobbies" }
 ]);
```

Input:

```
{ "_id": 1, "name": "Alice", "hobbies": ["reading", "cycling"] }
```

Output:

```

{ "_id": 1, "name": "Alice", "hobbies": "reading" }
{ "_id": 1, "name": "Alice", "hobbies": "cycling" }
```

8. $lookup

Performs a left outer join with another collection.

Example: Join orders with customers collection:

```
db.orders.aggregate([
  {
    $lookup: {
      from: "customers",
      localField: "customerId",
      foreignField: "_id",
      as: "customerDetails"
    }
  }
]);
```

9. $facet

Executes multiple pipelines in parallel and outputs the results in a single document.

Example: Calculate the total number of users and the top 5 users by age:

```
db.users.aggregate([
  {
    $facet: {
      totalUsers: [{ $count: "count" }],
      topUsers: [{ $sort: { age: -1 } }, { $limit: 5 }]
    }
  }
]);
```

10. $bucket

Groups documents into ranges (buckets) defined by boundaries.

Example: Group users into age ranges (buckets):

```
db.users.aggregate([
  {
    $bucket: {
      groupBy: "$age",
      boundaries: [0, 18, 30, 50],
      default: "Other",
      output: {
        count: { $sum: 1 },
        names: { $push: "$name" }
      }
    }
  }
]);
```

11. $filter

Filters elements of an array based on a condition.

Example: Filter scores array to retain only values greater than 50:

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      passingScores: {
        $filter: {
          input: "$scores",
          as: "score",
          cond: { $gt: ["$$score", 50] }
        }
      }
    }
  }
 ]);
```

Pipeline Expressions

MongoDB's aggregation framework supports expressions to compute values. Some common expressions include:

- $add: Adds numbers.
- $subtract: Subtracts numbers.
- $multiply: Multiplies numbers.
- $divide: Divides numbers.
- $concat: Concatenates strings.
- $arrayElemAt: Returns an element from an array.

Performance Tips

1. **Use $match Early**:
   - Filter as many documents as possible early in the pipeline to reduce processing.
2. **Optimize with Indexes**:
   - Fields used in $match and $sort stages should be indexed for faster performance.
3. **Avoid $unwind on Large Arrays**:
   - $unwind can be resource-intensive if applied to arrays with many elements.
4. **Use $facet Sparingly**:
   - It runs parallel pipelines but can consume more memory.
5. **Analyze with explain()**:
   - Use the explain() method to analyze query performance.

interview questions and answers

1. What is the Aggregation Framework in MongoDB?

Answer: The aggregation framework is a set of tools in MongoDB used to process and transform data stored in collections. It allows developers to:

- Filter, group, and sort data.
- Compute aggregated values like sums, averages, and counts.
- Perform operations like joining collections and filtering array elements.

It is similar to SQL's GROUP BY, SUM, and WHERE clauses.

2. What is an aggregation pipeline?

Answer: An aggregation pipeline is a sequence of stages, where each stage processes documents and passes the output to the next stage. Each stage performs a specific operation such as filtering, grouping, or reshaping documents.

Example Syntax:

```
db.collection.aggregate([
   { $match: { age: { $gt: 25 } } },
   { $group: { _id: "$gender", avgAge: { $avg: "$age" } } }
]);
```

3. What are the main stages in an aggregation pipeline?

Answer: Key aggregation stages include:

- **$match**: Filters documents based on conditions.
- **$group**: Groups documents by a field and performs aggregations.
- **$project**: Reshapes documents by including, excluding, or adding fields.
- **$sort**: Sorts documents by one or more fields.
- **$limit**: Limits the number of output documents.
- **$skip**: Skips a specified number of documents.
- **$unwind**: Deconstructs arrays into multiple documents.
- **$lookup**: Joins two collections.

---

4. What is the $group stage used for?

Answer: The $group stage groups documents by a specified field and applies accumulator expressions to compute values for each group.

Example: Calculate the total number of users in each gender group:

```
db.users.aggregate([
   { $group: { _id: "$gender", userCount: { $sum: 1 } } }
]);
```

5. What is the difference between $match and $filter?

Answer:

- **$match:**
  - Filters documents at the pipeline level.
  - Used to filter documents based on conditions (like a WHERE clause in SQL).
  - Operates on entire documents.

```
{ $match: { age: { $gt: 30 } } }
```

$filter:

- Filters elements within an array.
- Used within stages like $project or $addFields

```

{
    $project: {
      passingScores: {
        $filter: {
          input: "$scores",
          as: "score",
          cond: { $gt: ["$$score", 50] }
        }
      }
    }
  }

```

6. What is the $lookup stage, and how is it used?

Answer: The $lookup stage is used to perform a left outer join between two collections. It allows you to merge fields from another collection into the output.

Example: Join the orders collection with the users collection:

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

7. Explain the $unwind stage with an example.

Answer: The $unwind stage deconstructs an array field from a document into multiple documents, one for each element in the array.

Example: Input:

```
{ "_id": 1, "name": "Ajay", "hobbies": ["reading", "yoga"] }
```

Query:

```


db.users.aggregate([
   { $unwind: "$hobbies" }
 ]);
```

Output:

```
{ "_id": 1, "name": "Ajay", "hobbies": "reading" }
{ "_id": 1, "name": "Ajay", "hobbies": "yoga" }
```

8. What is $facet, and how is it used?

Answer: The $facet stage allows you to run multiple aggregation pipelines in parallel and output their results as a single document.

Example: Find the total number of users and the top 5 oldest users:

```
db.users.aggregate([
   {
    $facet: {
      totalUsers: [{ $count: "count" }],
      topUsers: [{ $sort: { age: -1 } }, { $limit: 5 }]
    }
   }
 ]);
```

9. What is $bucket, and how is it different from $group?

Answer:

- **$bucket:**
  - Groups documents into defined ranges (buckets) based on a field or expression.
  - Requires specifying boundaries.

Example: Group users into age ranges:

```
db.users.aggregate([
   {
    $bucket: {
      groupBy: "$age",
      boundaries: [20, 30, 40, 50],
      output: { count: { $sum: 1 } }
    }
  }
]);
```

- **$group:**
    - Groups documents by a specific field or computed expression.
    - Does not require fixed ranges.

---

10. How does $sort work in aggregation?

Answer: The $sort stage orders documents based on a field in ascending (1) or descending (-1) order.

Example: Sort users by age in descending order:

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