AWD 1111

MongoDB Aggregation Pipeline Guide

$match, $group, $sort, $project, $lookup, and more

Aggregation Pipeline Overview

The aggregation pipeline is MongoDB's powerful framework for data transformation and analysis. It processes documents through a sequence of stages, where each stage transforms the documents as they pass through.

What is a Pipeline?

Think of it like an assembly line: documents enter the pipeline, pass through multiple processing stages, and emerge transformed at the end. Each stage performs one operation on the documents.

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

Common Pipeline Stages

Stage	Purpose	SQL Equivalent
$match	Filter documents	WHERE
$group	Group and aggregate	GROUP BY
$sort	Order results	ORDER BY
$project	Reshape documents	SELECT
$limit	Limit results	TOP / LIMIT
$skip	Skip documents	OFFSET
$lookup	Join collections	JOIN
$unwind	Flatten arrays	(no equivalent)
$count	Count documents	COUNT(*)
$addFields	Add new fields	AS (alias)

Database Used in Examples
All examples use the concert_demo database with collections: concerts, venues, reviews, and fans.

1. $match - Filter Documents

$match filters documents to pass only those that match the specified condition(s). It works exactly like find() queries and should be placed early in the pipeline for efficiency.

1.1 Basic $match

```
// Filter for Rock concerts only
db.concerts.aggregate([
{ $match: { genres: "Rock" } }
])

// Filter for concerts in 2024
db.concerts.aggregate([
{ $match: { releaseYear: 2024 } }
])

// Filter for Gold members
db.fans.aggregate([
{ $match: { membershipLevel: "Gold" } }
])
```

1.2 $match with Operators

```
// Tickets under $100
db.concerts.aggregate([
{ $match: { ticketPrice: { $lt: 100 } } }
])

// Attendance over 50,000
db.concerts.aggregate([
{ $match: { attendance: { $gt: 50000 } } }
])

// Venues in Missouri or Illinois
db.venues.aggregate([
{ $match: { state: { $in: ["MO", "IL"] } } }
])
```

1.3 $match with Multiple Conditions

```
// 2024 concerts under $100
db.concerts.aggregate([
{ $match: {
releaseYear: 2024,
ticketPrice: { $lt: 100 }
}}
])

// Rock concerts with high attendance
db.concerts.aggregate([
{ $match: {
genres: "Rock",
attendance: { $gte: 20000 }
}}
])
```

Performance Tip
Always place $match as early as possible in your pipeline. It reduces the number of documents that subsequent stages need to process.

2. $group - Group and Aggregate

$group groups documents by a specified expression and can perform calculations on each group (sum, average, count, etc.). This is similar to SQL's GROUP BY clause.

2.1 Basic $group Syntax

```
{ $group: {
_id: , // Group by this field
: { : },
: { : },
...
}}
```

2.2 Accumulator Operators

Accumulator	Description	Example
$sum	Sum of values	{ $sum: '$ticketPrice' }
$avg	Average of values	{ $avg: '$attendance' }
$min	Minimum value	{ $min: '$ticketPrice' }
$max	Maximum value	{ $max: '$attendance' }
$count	Count documents	{ $count: {} }
$push	Array of values	{ $push: '$artist' }
$addToSet	Array of unique values	{ $addToSet: '$state' }
$first	First value in group	{ $first: '$tourName' }
$last	Last value in group	{ $last: '$tourName' }

2.3 Count by Group

```
// Count concerts per year
db.concerts.aggregate([
{ $group: {
_id: "$releaseYear",
count: { $sum: 1 }
}}
])

// Count fans by membership level
db.fans.aggregate([
{ $group: {
_id: "$membershipLevel",
count: { $sum: 1 }
}}
])

// Count venues by state
db.venues.aggregate([
{ $group: {
_id: "$state",
count: { $sum: 1 }
}}
])
```

2.4 Sum and Average

```
// Total attendance per year
db.concerts.aggregate([
{ $group: {
_id: "$releaseYear",
totalAttendance: { $sum: "$attendance" }
}}
])

// Average ticket price per year
db.concerts.aggregate([
{ $group: {
_id: "$releaseYear",
avgPrice: { $avg: "$ticketPrice" }
}}
])

// Total capacity by state
db.venues.aggregate([
{ $group: {
_id: "$state",
totalCapacity: { $sum: "$capacity" }
}}
])
```

2.5 Min, Max, and Multiple Accumulators

```
// Price statistics per year
db.concerts.aggregate([
{ $group: {
_id: "$releaseYear",
minPrice: { $min: "$ticketPrice" },
maxPrice: { $max: "$ticketPrice" },
avgPrice: { $avg: "$ticketPrice" },
count: { $sum: 1 }
}}
])

// Venue statistics by state
db.venues.aggregate([
{ $group: {
_id: "$state",
minCapacity: { $min: "$capacity" },
maxCapacity: { $max: "$capacity" },
venueCount: { $sum: 1 }
}}
])
```

2.6 Grouping All Documents (_id: null)

```
// Overall statistics (no grouping)
db.concerts.aggregate([
{ $group: {
_id: null,
totalConcerts: { $sum: 1 },
totalAttendance: { $sum: "$attendance" },
avgTicketPrice: { $avg: "$ticketPrice" }
}}
])

// Total fans across all levels
db.fans.aggregate([
{ $group: {
_id: null,
totalFans: { $sum: 1 }
}}
])
```

_id: null
Using _id: null groups ALL documents into a single group, useful for calculating totals across the entire collection.

2.7 Collecting Values with $push and $addToSet

```
// List all artists per year
db.concerts.aggregate([
{ $group: {
_id: "$releaseYear",
artists: { $push: "$artist" }
}}
])

// Unique states where fans live (per membership level)
db.fans.aggregate([
{ $group: {
_id: "$membershipLevel",
states: { $addToSet: "$state" }
}}
])

// Collect all venue names by state
db.venues.aggregate([
{ $group: {
_id: "$state",
venues: { $push: "$name" }
}}
])
```

$push vs $addToSet
$push adds ALL values (may have duplicates). $addToSet adds only UNIQUE values.

3. $sort - Order Results

$sort orders the documents. Use 1 for ascending order, -1 for descending order.

3.1 Basic Sorting

```
// Sort by ticket price (low to high)
db.concerts.aggregate([
{ $sort: { ticketPrice: 1 } }
])

// Sort by attendance (high to low)
db.concerts.aggregate([
{ $sort: { attendance: -1 } }
])

// Sort by release year (newest first)
db.concerts.aggregate([
{ $sort: { releaseYear: -1 } }
])
```

3.2 Multi-field Sorting

```
// Sort by year (desc), then price (asc)
db.concerts.aggregate([
{ $sort: { releaseYear: -1, ticketPrice: 1 } }
])

// Sort by state, then capacity
db.venues.aggregate([
{ $sort: { state: 1, capacity: -1 } }
])
```

3.3 Sorting After $group

```
// Count by year, sorted by count (most concerts first)
db.concerts.aggregate([
{ $group: {
_id: "$releaseYear",
count: { $sum: 1 }
}},
{ $sort: { count: -1 } }
])

// Average price by year, sorted by price
db.concerts.aggregate([
{ $group: {
_id: "$releaseYear",
avgPrice: { $avg: "$ticketPrice" }
}},
{ $sort: { avgPrice: -1 } }
])
```

4. $project - Reshape Documents

$project reshapes documents by including, excluding, or transforming fields. It's like SELECT in SQL but more powerful.

4.1 Include/Exclude Fields

```
// Include only specific fields
db.concerts.aggregate([
{ $project: {
artist: 1,
tourName: 1,
ticketPrice: 1
}}
])

// Exclude specific fields
db.concerts.aggregate([
{ $project: {
setList: 0,
venue_ids: 0
}}
])

// Exclude _id, include others
db.concerts.aggregate([
{ $project: {
_id: 0,
artist: 1,
tourName: 1
}}
])
```

4.2 Rename Fields

```
// Rename fields using $project
db.concerts.aggregate([
{ $project: {
_id: 0,
name: "$artist",
tour: "$tourName",
price: "$ticketPrice"
}}
])

// Rename in venues
db.venues.aggregate([
{ $project: {
_id: 0,
venueName: "$name",
location: "$city"
}}
])
```

4.3 Computed Fields

```
// Calculate revenue (price * attendance)
db.concerts.aggregate([
{ $project: {
artist: 1,
tourName: 1,
estimatedRevenue: {
$multiply: ["$ticketPrice", "$attendance"]
}
}}
])

// Calculate price per 1000 attendees
db.concerts.aggregate([
{ $project: {
artist: 1,
pricePerThousand: {
$divide: ["$ticketPrice", { $divide: ["$attendance", 1000] }]
}
}}
])
```

4.4 Array Operations in $project

```
// Get array size
db.concerts.aggregate([
{ $project: {
artist: 1,
numberOfGenres: { $size: "$genres" },
numberOfVenues: { $size: "$venue_ids" },
numberOfSongs: { $size: "$setList" }
}}
])

// Get first element of array
db.concerts.aggregate([
{ $project: {
artist: 1,
firstGenre: { $arrayElemAt: ["$genres", 0] },
firstSong: { $arrayElemAt: ["$setList", 0] }
}}
])
```

4.5 String Operations

```
// Uppercase artist names
db.concerts.aggregate([
{ $project: {
artist: { $toUpper: "$artist" },
tourName: 1
}}
])

// Concatenate fields
db.concerts.aggregate([
{ $project: {
fullTitle: {
$concat: ["$artist", " - ", "$tourName"]
}
}}
])

// Get string length
db.concerts.aggregate([
{ $project: {
artist: 1,
nameLength: { $strLenCP: "$artist" }
}}
])
```

5. $limit and $skip - Pagination

$limit restricts the number of documents. $skip skips a number of documents. Together they enable pagination.

5.1 $limit

```
// Get top 3 most expensive concerts
db.concerts.aggregate([
{ $sort: { ticketPrice: -1 } },
{ $limit: 3 }
])

// Get top 5 largest venues
db.venues.aggregate([
{ $sort: { capacity: -1 } },
{ $limit: 5 }
])
```

5.2 $skip

```
// Skip first 2 concerts
db.concerts.aggregate([
{ $sort: { ticketPrice: -1 } },
{ $skip: 2 }
])
```

5.3 Pagination Pattern

```
// Page 1 (first 2 results)
db.concerts.aggregate([
{ $sort: { ticketPrice: -1 } },
{ $skip: 0 },
{ $limit: 2 }
])

// Page 2 (next 2 results)
db.concerts.aggregate([
{ $sort: { ticketPrice: -1 } },
{ $skip: 2 },
{ $limit: 2 }
])

// Page 3 (next 2 results)
db.concerts.aggregate([
{ $sort: { ticketPrice: -1 } },
{ $skip: 4 },
{ $limit: 2 }
])
```

Pagination Formula
For page N with pageSize items: $skip: (N - 1) * pageSize, then $limit: pageSize

6. $lookup - Join Collections

$lookup performs a left outer join between two collections. This is how you combine data from related collections, similar to SQL JOIN.

6.1 Basic $lookup Syntax

```
{ $lookup: {
from: "other_collection", // Collection to join
localField: "field_in_this", // Field in current collection
foreignField: "field_in_other", // Field in other collection
as: "output_array_name" // Name for joined data
}}
```

6.2 Join Concerts with Venues

```
// Get concert details with venue information
db.concerts.aggregate([
{ $lookup: {
from: "venues",
localField: "venue_ids",
foreignField: "_id",
as: "venueDetails"
}}
])
```

This adds a venueDetails array to each concert containing the matched venue documents.

6.3 Join Reviews with Concerts

```
// Get reviews with concert information
db.reviews.aggregate([
{ $lookup: {
from: "concerts",
localField: "concert_id",
foreignField: "_id",
as: "concertInfo"
}}
])

// With projection to clean up output
db.reviews.aggregate([
{ $lookup: {
from: "concerts",
localField: "concert_id",
foreignField: "_id",
as: "concertInfo"
}},
{ $project: {
fan_name: 1,
text: 1,
date: 1,
concertArtist: { $arrayElemAt: ["$concertInfo.artist", 0] },
concertTour: { $arrayElemAt: ["$concertInfo.tourName", 0] }
}}
])
```

6.4 $lookup with $match

```
// Get Rock concerts with venue details
db.concerts.aggregate([
{ $match: { genres: "Rock" } },
{ $lookup: {
from: "venues",
localField: "venue_ids",
foreignField: "_id",
as: "venueDetails"
}},
{ $project: {
artist: 1,
tourName: 1,
venueDetails: { name: 1, city: 1, state: 1 }
}}
])
```

6.5 Reverse Lookup: Venues with Concerts

```
// Get venues with concerts that played there
db.venues.aggregate([
{ $lookup: {
from: "concerts",
localField: "_id",
foreignField: "venue_ids",
as: "concertsAtVenue"
}},
{ $project: {
name: 1,
city: 1,
numberOfConcerts: { $size: "$concertsAtVenue" },
artists: "$concertsAtVenue.artist"
}}
])
```

$lookup Result
$lookup always returns an array in the 'as' field, even if there's only one match or no matches (empty array).

7. $unwind - Flatten Arrays

$unwind deconstructs an array field, creating a separate document for each array element. This is essential when you need to group or aggregate by array elements.

7.1 Basic $unwind

```
// Create one document per genre
db.concerts.aggregate([
{ $unwind: "$genres" }
])

// Before: { artist: "Taylor Swift", genres: ["Pop", "Country"] }
// After: { artist: "Taylor Swift", genres: "Pop" }
// { artist: "Taylor Swift", genres: "Country" }
```

7.2 $unwind + $group (Count by Array Element)

```
// Count concerts per genre
db.concerts.aggregate([
{ $unwind: "$genres" },
{ $group: {
_id: "$genres",
count: { $sum: 1 }
}},
{ $sort: { count: -1 } }
])

// Count concerts per venue
db.concerts.aggregate([
{ $unwind: "$venue_ids" },
{ $group: {
_id: "$venue_ids",
concertCount: { $sum: 1 }
}}
])
```

7.3 $unwind with $lookup Results

```
// Flatten venue details after lookup
db.concerts.aggregate([
{ $lookup: {
from: "venues",
localField: "venue_ids",
foreignField: "_id",
as: "venueDetails"
}},
{ $unwind: "$venueDetails" },
{ $project: {
artist: 1,
tourName: 1,
venueName: "$venueDetails.name",
venueCity: "$venueDetails.city"
}}
])
```

7.4 Preserving Empty Arrays

```
// By default, $unwind removes documents with empty/missing arrays
// Use preserveNullAndEmptyArrays to keep them

db.concerts.aggregate([
{ $unwind: {
path: "$genres",
preserveNullAndEmptyArrays: true
}}
])
```

$unwind Warning
If a document has an array with 5 elements, $unwind creates 5 documents. This can significantly increase the number of documents in your pipeline.

8. $addFields and $count

8.1 $addFields

$addFields adds new fields to documents without removing existing ones. Unlike $project, it keeps all original fields.

```
// Add calculated revenue field
db.concerts.aggregate([
{ $addFields: {
estimatedRevenue: { $multiply: ["$ticketPrice", "$attendance"] }
}}
])

// Add multiple fields
db.concerts.aggregate([
{ $addFields: {
genreCount: { $size: "$genres" },
venueCount: { $size: "$venue_ids" },
isExpensive: { $gte: ["$ticketPrice", 100] }
}}
])
```

8.2 $count

$count returns a document with a count of the documents at that stage.

```
// Count all concerts
db.concerts.aggregate([
{ $count: "totalConcerts" }
])
// Returns: { "totalConcerts": 5 }

// Count Rock concerts
db.concerts.aggregate([
{ $match: { genres: "Rock" } },
{ $count: "rockConcertCount" }
])

// Count venues in Missouri
db.venues.aggregate([
{ $match: { state: "MO" } },
{ $count: "moVenueCount" }
])
```

9. Complete Pipeline Examples

Example 1: Top Reviewers

Find the fans who have written the most reviews.

```
db.reviews.aggregate([
// Group by fan name and count
{ $group: {
_id: "$fan_name",
reviewCount: { $sum: 1 }
}},
// Sort by count descending
{ $sort: { reviewCount: -1 } },
// Get top 10
{ $limit: 10 }
])
```

Example 2: Genre Popularity

Calculate total attendance by genre.

```
db.concerts.aggregate([
// Flatten genres array
{ $unwind: "$genres" },
// Group by genre
{ $group: {
_id: "$genres",
totalAttendance: { $sum: "$attendance" },
avgTicketPrice: { $avg: "$ticketPrice" },
concertCount: { $sum: 1 }
}},
// Sort by attendance
{ $sort: { totalAttendance: -1 } }
])
```

Example 3: Venue Performance Report

Get detailed statistics for each venue.

```
db.venues.aggregate([
// Join with concerts
{ $lookup: {
from: "concerts",
localField: "_id",
foreignField: "venue_ids",
as: "concerts"
}},
// Calculate statistics
{ $project: {
name: 1,
city: 1,
state: 1,
capacity: 1,
totalConcerts: { $size: "$concerts" },
artists: "$concerts.artist"
}},
// Sort by concert count
{ $sort: { totalConcerts: -1 } }
])
```

Example 4: Recent Reviews with Concert Info

Get the 10 most recent reviews with full concert details.

```
db.reviews.aggregate([
// Sort by date (newest first)
{ $sort: { date: -1 } },
// Limit to 10
{ $limit: 10 },
// Join with concerts
{ $lookup: {
from: "concerts",
localField: "concert_id",
foreignField: "_id",
as: "concert"
}},
// Flatten the concert array
{ $unwind: "$concert" },
// Clean up output
{ $project: {
fan_name: 1,
text: 1,
date: 1,
artist: "$concert.artist",
tourName: "$concert.tourName"
}}
])
```

10. Pipeline Stages Quick Reference

Stage	Syntax	Purpose
$match	{ $match: { field: value } }	Filter documents
$group	{ $group: { _id: '$field', ... } }	Group and aggregate
$sort	{ $sort: { field: 1 or -1 } }	Order results
$project	{ $project: { field: 1 or 0 } }	Include/exclude fields
$limit	{ $limit: number }	Limit results
$skip	{ $skip: number }	Skip documents
$lookup	{ $lookup: { from, localField, ... } }	Join collections
$unwind	{ $unwind: '$arrayField' }	Flatten arrays
$count	{ $count: 'fieldName' }	Count documents
$addFields	{ $addFields: { newField: expr } }	Add new fields

Common Accumulators (for $group)

Accumulator	Example	Description
$sum	{ $sum: '$field' } or { $sum: 1 }	Sum values or count
$avg	{ $avg: '$field' }	Average
$min	{ $min: '$field' }	Minimum
$max	{ $max: '$field' }	Maximum
$push	{ $push: '$field' }	Collect into array
$addToSet	{ $addToSet: '$field' }	Collect unique values
$first	{ $first: '$field' }	First value
$last	{ $last: '$field' }	Last value

11. Reference Links

For more information, see the official MongoDB documentation:

- Aggregation Pipeline:
  https://www.mongodb.com/docs/manual/core/aggregation-pipeline/
- Pipeline Stages:
  https://www.mongodb.com/docs/manual/reference/operator/aggregation-pipeline/
- $group Stage:
  https://www.mongodb.com/docs/manual/reference/operator/aggregation/group/
- $lookup Stage:
  https://www.mongodb.com/docs/manual/reference/operator/aggregation/lookup/
- Aggregation Operators:
  https://www.mongodb.com/docs/manual/reference/operator/aggregation/