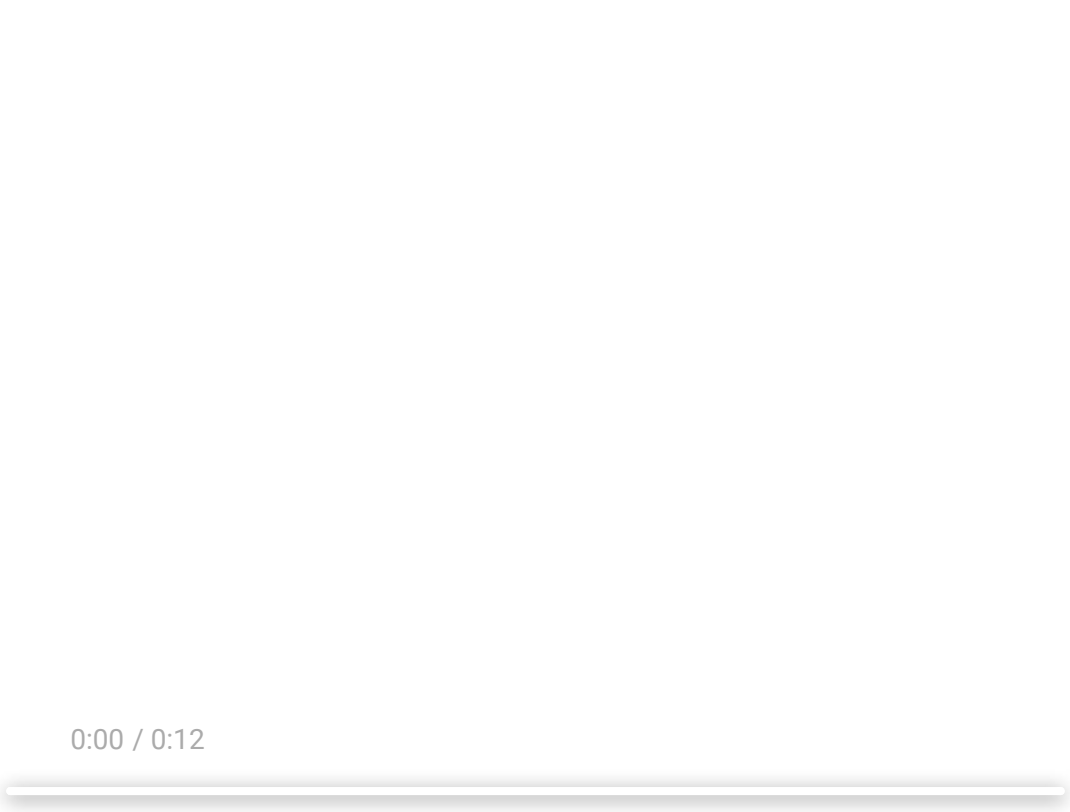
Aggregation Pipeline

On this page

The aggregation pipeline is a framework for data aggregation modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into aggregated results. For example:



In the example,

```
db.orders.aggreg
   { $match: { s
   { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
])
```


First Stage: The `$match` stage filters the documents by the `status` field and passes to the next stage those documents that have `status` equal to `"A"`.

Second Stage: The `$group` stage groups the documents by the `cust_id` field to calculate the sum of the amount for each unique `cust_id`.

Pipeline

The MongoDB aggregation pipeline consists of stages. Each stage transforms the documents as they pass through the pipeline. Pipeline stages do not need to produce one output document for every input document; e.g., some stages may generate new documents or filter out documents.

Pipeline stages can appear multiple times in the pipeline with the exception of `$out`, `$merge`, and `$geoNear` stages. For a list of all available stages, see Aggregation Pipeline Stages.

MongoDB provides the `db.collection.aggregate()` method in the `mongo` shell and the `aggregate` command to run the aggregation pipeline.

For example usage of the aggregation pipeline, consider Aggregation with User Preference Data and Aggregation with the Zip Code Data Set.

Starting in MongoDB 4.2, you can use the aggregation pipeline for updates in:

Command	`mongo` Shell Methods
`findAndModify`	`db.collection.findAndModify()`
	`db.collection.findOneAndUpdate()`

`update`	`db.collection.updateOne()`
	`db.collection.updateMany()`
	`db.collection.update()`

Pipeline Expressions

Some pipeline stages take a pipeline expression as the operand. Pipeline expressions specify the transformation to apply to the input documents. Expressions have a document structure and can contain other expression.

Pipeline expressions can only operate on the current document in the pipeline and cannot refer to data from other documents: expression operations provide in-memory transformation of documents.

Generally, expressions are stateless and are only evaluated when seen by the aggregation process with one exception: accumulator expressions.

The accumulators, used in the `$group` stage, maintain their state (e.g. totals, maximums, minimums, and related data) as documents progress through the pipeline.

Changed in version 3.2: Some accumulators are available in the `$project` stage; however, when used in the `$project` stage, the accumulators do not maintain their state across documents.

For more information on expressions, see Expressions.

Aggregation Pipeline Behavior

In MongoDB, the `aggregate` command operates on a single collection, logically passing the *entire* collection into the aggregation pipeline. To optimize the operation, wherever possible, use the following strategies to avoid scanning the entire collection.

Pipeline Operators and Indexes

MongoDB's query plar
improve pipeline perfor

> **NOTE:**
> The following pipeline stages do not represent a complete list of all stages which can use an index.

`$match`
The `$match` stage can use an index to filter documents if it occurs at the beginning of a pipeline.

`$sort`
The `$sort` stage can use an index as long as it is not preceded by a `$project`, `$unwind`, or `$group` stage.

`$group`
The `$group` stage can sometimes use an index to find the first document in each group if all of the following criteria are met:

- The `$group` stage is preceded by a `$sort` stage that sorts the field to group by,
- There is an index on the grouped field which matches the sort order and
- The only accumulator used in the `$group` stage is `$first`.

See Optimization to Return the First Document of Each Group for an example.

`$geoNear`
The `$geoNear` pipeline operator takes advantage of a geospatial index. When using `$geoNear`, the `$geoNear` pipeline operation must appear as the first stage in an aggregation pipeline.

Changed in version 3.2: Starting in MongoDB 3.2, indexes can cover an aggregation pipeline. In MongoDB 2.6 and 3.0, indexes could not cover an aggregation pipeline since even when the pipeline uses an index, aggregation still requires access to the actual documents.

Early Filtering

If your aggregation operation requires only a subset of the data in a collection, use the `$match`, `$limit`, and `$skip` stages to restrict the documents that enter at the beginning of the pipeline. When placed at the beginning of a pipeline, `$match` operations use suitable indexes to scan only the matching documents in a collection.

Placing a `$match` pipe
equivalent to a single q
the beginning of the pipeline.

Considerations

Sharded Collections

The aggregation pipeline supports operations on sharded collections. See Aggregation Pipeline and Sharded Collections.

Aggregation vs Map-Reduce

The aggregation pipeline provides an alternative to map-reduce and may be the preferred solution for aggregation tasks where the complexity of map-reduce may be unwarranted.

Limitations

Aggregation pipeline have some limitations on value types and result size. See Aggregation Pipeline Limits for details on limits and restrictions on the aggregation pipeline.

Pipeline Optimization

The aggregation pipeline has an internal optimization phase that provides improved performance for certain sequences of operators. For details, see Aggregation Pipeline Optimization.