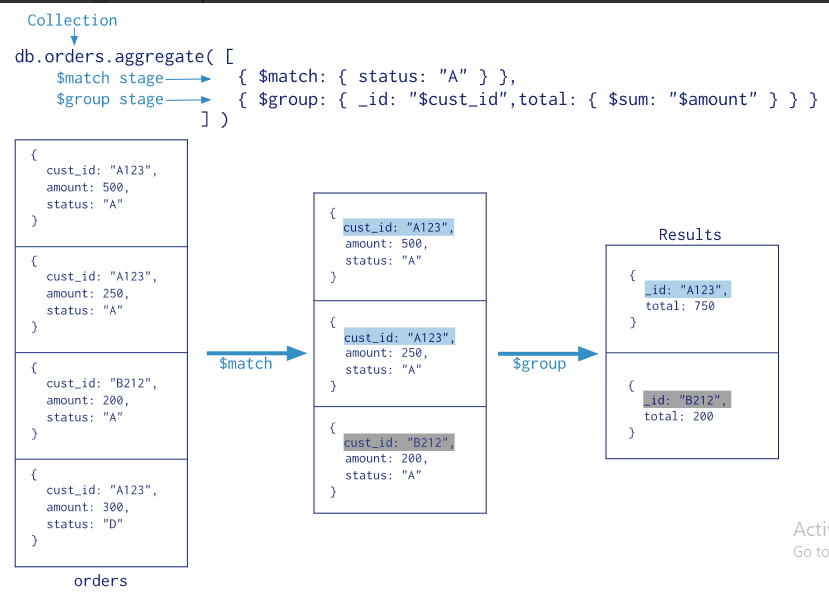
MongoDB Pipelining

**Aggregation Pipeline Stages**

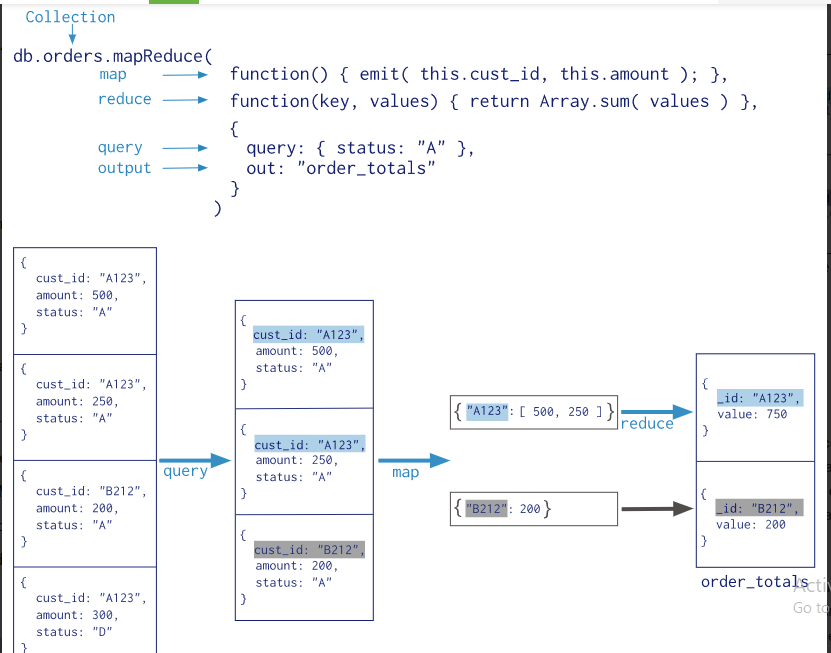
In the [db.collection.aggregate](https://docs.mongodb.com/manual/reference/method/db.collection.aggregate/#db.collection.aggregate) method and [db.aggregate](https://docs.mongodb.com/manual/reference/method/db.aggregate/#db.aggregate) method, [pipeline](https://docs.mongodb.com/manual/core/aggregation-pipeline/) stages appear in an array. Documents pass through the stages in sequence.

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.



# Map-Reduce

Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. For map-reduce operations, MongoDB provides the [mapReduce](https://docs.mongodb.com/manual/reference/command/mapReduce/#dbcmd.mapReduce) database command.

In this map-reduce operation, MongoDB applies the map phase to each input document (i.e. the documents in the collection that match the query condition). The map function emits key-value pairs. For those keys that have multiple values, MongoDB applies the reduce phase, which collects and condenses the aggregated data. MongoDB then stores the results in a collection. Optionally, the output of the reduce function may pass through a finalize function to further condense or process the results of the aggregation.

All map-reduce functions in MongoDB are JavaScript and run within the [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) process. Map-reduce operations take the documents of a single [collection](https://docs.mongodb.com/manual/reference/glossary/#term-collection) as the input and can perform any arbitrary sorting and limiting before beginning the map stage. [mapReduce](https://docs.mongodb.com/manual/reference/command/mapReduce/#dbcmd.mapReduce) can return the results of a map-reduce operation as a document, or may write the results to collections. The input and the output collections may be sharded.

## Map-Reduce JavaScript Functions

In MongoDB, map-reduce operations use custom JavaScript functions to map, or associate, values to a key. If a key has multiple values mapped to it, the operation reduces the values for the key to a single object.

The use of custom JavaScript functions provide flexibility to map-reduce operations. For instance, when processing a document, the map function can create more than one key and value mapping or no mapping. Map-reduce operations can also use a custom JavaScript function to make final modifications to the results at the end of the map and reduce operation, such as perform additional calculations.

## Map-Reduce Behavior

In MongoDB, the map-reduce operation can write results to a collection or return the results inline. If you write map-reduce output to a collection, you can perform subsequent map-reduce operations on the same input collection that merge replace, merge, or reduce new results with previous results. See [mapReduce](https://docs.mongodb.com/manual/reference/command/mapReduce/#dbcmd.mapReduce) and [Perform Incremental Map-Reduce](https://docs.mongodb.com/manual/tutorial/perform-incremental-map-reduce/) for details and examples.

When returning the results of a map-reduce operation inline, the result documents must be within the [BSON Document Size](https://docs.mongodb.com/manual/reference/limits/#BSON-Document-Size) limit, which is currently 16 megabytes. For additional information on limits and restrictions on map-reduce operations, see the [mapReduce](https://docs.mongodb.com/manual/reference/command/mapReduce/) reference page.

# Map-Reduce Examples

{

\_id: ObjectId("50a8240b927d5d8b5891743c"),

cust\_id: "abc123",

ord\_date: new Date("Oct 04, 2012"),

status: 'A',

price: 25,

items: [ { sku: "mmm", qty: 5, price: 2.5 },

{ sku: "nnn", qty: 5, price: 2.5 } ]

}

## Return the Total Price Per Customer

Perform the map-reduce operation on the orders collection to group by the cust\_id, and calculate the sum of the price for each cust\_id:

1. Define the map function to process each input document:
   * In the function, this refers to the document that the map-reduce operation is processing.
   * The function maps the price to the cust\_id for each document and emits the cust\_id and price pair.

var mapFunction1 = function() {

emit(this.cust\_id, this.price); };

Define the corresponding reduce function with two arguments keyCustId and valuesPrices:

* The valuesPrices is an array whose elements are the price values emitted by the map function and grouped by keyCustId.
* The function reduces the valuesPrice array to the sum of its elements.

var reduceFunction1 = function(keyCustId, valuesPrices) {

return Array.sum(valuesPrices);

};

Perform the map-reduce on all documents in the orders collection using the mapFunction1 map function and the reduceFunction1 reduce function.

db.orders.mapReduce(

mapFunction1,

reduceFunction1,

{ out: "map\_reduce\_example" }

)

## Calculate Order and Total Quantity with Average Quantity Per Item

In this example, you will perform a map-reduce operation on the orders collection for all documents that have an ord\_date value greater than 01/01/2012. The operation groups by the item.sku field, and calculates the number of orders and the total quantity ordered for each sku. The operation concludes by calculating the average quantity per order for each sku value:

1. Define the map function to process each input document:
   * In the function, this refers to the document that the map-reduce operation is processing.
   * For each item, the function associates the sku with a new object value that contains the count of 1 and the item qty for the order and emits the sku and value pair.

 var mapFunction2 = function() {

for (var idx = 0; idx < this.items.length; idx++) {

var key = this.items[idx].sku;

var value = {

count: 1,

qty: this.items[idx].qty

};

emit(key, value);

}

};

 Define the corresponding reduce function with two arguments keySKU and countObjVals:

* countObjVals is an array whose elements are the objects mapped to the grouped keySKU values passed by map function to the reducer function.
* The function reduces the countObjVals array to a single object reducedValue that contains the count and the qty fields.
* In reducedVal, the count field contains the sum of the count fields from the individual array elements, and the qty field contains the sum of the qty fields from the individual array elements.

 var reduceFunction2 = function(keySKU, countObjVals) {

reducedVal = { count: 0, qty: 0 };

for (var idx = 0; idx < countObjVals.length; idx++) {

reducedVal.count += countObjVals[idx].count;

reducedVal.qty += countObjVals[idx].qty;

}

return reducedVal;

};

 Define a finalize function with two arguments key and reducedVal. The function modifies the reducedVal object to add a computed field named avg and returns the modified object:

 var finalizeFunction2 = function (key, reducedVal) {

reducedVal.avg = reducedVal.qty/reducedVal.count;

return reducedVal;

};

 Perform the map-reduce operation on the orders collection using the mapFunction2, reduceFunction2, and finalizeFunction2 functions.

db.orders.mapReduce( mapFunction2,

reduceFunction2,

{

out: { merge: "map\_reduce\_example" },

query: { ord\_date:

{ $gt: new Date('01/01/2012') }

},

finalize: finalizeFunction2

}

)

This operation uses the query field to select only those documents with ord\_date greater than new Date(01/01/2012). Then it output the results to a collection map\_reduce\_example. If the map\_reduce\_example collection already exists, the operation will merge the existing contents with the results of this map-reduce operation.