

# Aggregation Pipeline Operators

## On this page

- [Stage Operators](#)
- [Expression Operators](#)
- [Accumulators](#)


**NOTE:**  
For details on specific operator, including syntax and examples, click on the specific operator to go to its reference page.


## Stage Operators

In the `db.collection.aggregate` method, pipeline stages appear in an array. Documents pass through the stages in sequence.

```
db.collection.aggregate( [ { <stage> }, ... ] )
```

Name	Description
<code>\$collStats</code>	Returns statistics regarding a collection or view.
<code>\$project</code>	Reshapes each document in the stream, such as by adding new fields or removing existing fields. For each input document, outputs one document.
<code>\$match</code>	Filters the document stream to allow only matching documents to pass unmodified into the next pipeline stage. <code>\$match</code> uses standard MongoDB queries. For each input document, outputs either one document (a match) or zero documents (no match).
<code>\$redact</code>	Reshapes each document in the stream by restricting the content for each document based on information stored in the documents themselves. Incorporates the functionality of <code>\$project</code> and <code>\$match</code> . Can be used to implement field level redaction. For each input document, outputs either one or zero documents.
<code>\$limit</code>	Passes the first <i>n</i> documents unmodified to the pipeline where <i>n</i> is the specified limit. For each input document, outputs either one document (for the first <i>n</i> documents) or zero documents (after the first <i>n</i> documents).

Name	Description
 \$skip	Skips the first <i>n</i> documents where <i>n</i> is the specified skip number and passes the remaining documents unmodified to the pipeline. For each input document, outputs either zero documents (for the first <i>n</i> documents) or one document (if after the first <i>n</i> documents).
\$unwind	Deconstructs an array field from the input documents to output a document for <i>each</i> element. Each output document replaces the array with an element value. For each input document, outputs <i>n</i> documents where <i>n</i> is the number of array elements and can be zero for an empty array.
\$group	Groups input documents by a specified identifier expression and applies the accumulator expression(s), if specified, to each group. Consumes all input documents and outputs one document per each distinct group. The output documents only contain the identifier field and, if specified, accumulated fields.
\$sample	Randomly selects the specified number of documents from its input.
\$sort	Reorders the document stream by a specified sort key. Only the order changes; the documents remain unmodified. For each input document, outputs one document.
\$geoNear	Returns an ordered stream of documents based on the proximity to a geospatial point. Incorporates the functionality of \$match, \$sort, and \$limit for geospatial data. The output documents include an additional distance field and can include a location identifier field.
\$lookup	Performs a left outer join to another collection in the <i>same</i> database to filter in documents from the “joined” collection for processing.
\$out	Writes the resulting documents of the aggregation pipeline to a collection. To use the \$out stage, it must be the last stage in the pipeline.
\$indexStats	Returns statistics regarding the use of each index for the collection.
\$facet	Processes multiple aggregation pipelines within a single stage on the same set of input documents. Enables the creation of multi-faceted aggregations capable of characterizing data across multiple dimensions, or facets, in a single stage.
\$bucket	Categorizes incoming documents into groups, called buckets, based on a specified expression and bucket boundaries.
\$bucketAuto	Categorizes incoming documents into a specific number of groups, called buckets, based on a specified expression. Bucket boundaries are automatically determined in an attempt to evenly distribute the documents into the specified number of buckets.
\$sortByCount	Groups incoming documents based on the value of a specified expression, then computes the count of documents in each distinct group.

Name	Description
 \$addFields	Adds new fields to documents. Outputs documents that contain all existing fields from the input documents and newly added fields.
\$replaceRoot	Replaces a document with the specified embedded document. The operation replaces all existing fields in the input document, including the <code>_id</code> field. Specify a document embedded in the input document to promote the embedded document to the top level.
\$count	Returns a count of the number of documents at this stage of the aggregation pipeline.
\$graphLookup	Performs a recursive search on a collection. To each output document, adds a new array field that contains the traversal results of the recursive search for that document.

## Expression Operators

These expression operators are available to construct expressions for use in the aggregation pipeline.

Operator expressions are similar to functions that take arguments. In general, these expressions take an array of arguments and have the following form:

```
{ <operator>: [ <argument1>, <argument2> ... ] }
```

If operator accepts a single argument, you can omit the outer array designating the argument list:

```
{ <operator>: <argument> }
```


To avoid parsing ambiguity if the argument is a literal array, you must wrap the literal array in a `$literal` expression or keep the outer array that designates the argument list.

## Boolean Operators

Boolean expressions evaluate their argument expressions as booleans and return a boolean as the result.

In addition to the `false` boolean value, Boolean expression evaluates as `false` the following: `null`, `0`, and `undefined` values. The Boolean expression evaluates all other values as `true`, including non-zero numeric values and arrays.

Name	Description
\$and	Returns <code>true</code> only when <i>all</i> its expressions evaluate to <code>true</code> . Accepts any number of argument expressions.

Name	Description
 <code>\$or</code>	Returns <code>true</code> when <i>any</i> of its expressions evaluates to <code>true</code> . Accepts any number of argument expressions.
<code>\$not</code>	Returns the boolean value that is the opposite of its argument expression. Accepts a single argument expression.

## Set Operators

Set expressions performs set operation on arrays, treating arrays as sets. Set expressions ignores the duplicate entries in each input array and the order of the elements.

If the set operation returns a set, the operation filters out duplicates in the result to output an array that contains only unique entries. The order of the elements in the output array is unspecified.

If a set contains a nested array element, the set expression does *not* descend into the nested array but evaluates the array at top-level.

Name	Description
<code>\$setEquals</code>	Returns <code>true</code> if the input sets have the same distinct elements. Accepts two or more argument expressions.
<code>\$setIntersection</code>	Returns a set with elements that appear in <i>all</i> of the input sets. Accepts any number of argument expressions.
<code>\$setUnion</code>	Returns a set with elements that appear in <i>any</i> of the input sets. Accepts any number of argument expressions.
<code>\$setDifference</code>	Returns a set with elements that appear in the first set but not in the second set; i.e. performs a relative complement ↗ of the second set relative to the first. Accepts exactly two argument expressions.
<code>\$setIsSubset</code>	Returns <code>true</code> if all elements of the first set appear in the second set, including when the first set equals the second set; i.e. not a strict subset ↗. Accepts exactly two argument expressions.
<code>\$anyElementTrue</code>	Returns <code>true</code> if <i>any</i> elements of a set evaluate to <code>true</code> ; otherwise, returns <code>false</code> . Accepts a single argument expression.
<code>\$allElementsTrue</code>	Returns <code>true</code> if <i>no</i> element of a set evaluates to <code>false</code> , otherwise, returns <code>false</code> . Accepts a single argument expression.

## Comparison Operators

Comparison expressions return a boolean except for `$cmp` which returns a number.

The comparison expressions take two argument expressions and compare both value and type, using the specified BSON comparison order for values of different types.

Name	Description
\$cmp	Returns: 0 if the two values are equivalent, 1 if the first value is greater than the second, and −1 if the first value is less than the second.
\$eq	Returns true if the values are equivalent.
\$gt	Returns true if the first value is greater than the second.
\$gte	Returns true if the first value is greater than or equal to the second.
\$lt	Returns true if the first value is less than the second.
\$lte	Returns true if the first value is less than or equal to the second.
\$ne	Returns true if the values are <i>not</i> equivalent.

## Arithmetic Operators

Arithmetic expressions perform mathematic operations on numbers. Some arithmetic expressions can also support date arithmetic.


Name	Description
\$abs	Returns the absolute value of a number.
\$add	Adds numbers to return the sum, or adds numbers and a date to return a new date. If adding numbers and a date, treats the numbers as milliseconds. Accepts any number of argument expressions, but at most, one expression can resolve to a date.
\$ceil	Returns the smallest integer greater than or equal to the specified number.
\$divide	Returns the result of dividing the first number by the second. Accepts two argument expressions.
\$exp	Raises e to the specified exponent.
\$floor	Returns the largest integer less than or equal to the specified number.
\$ln	Calculates the natural log of a number.
\$log	Calculates the log of a number in the specified base.
\$log10	Calculates the log base 10 of a number.



Name	Description
\$mod	Returns the remainder of the first number divided by the second. Accepts two argument expressions.
\$multiply	Multiplies numbers to return the product. Accepts any number of argument expressions.
\$pow	Raises a number to the specified exponent.
\$sqrt	Calculates the square root.
\$subtract	Returns the result of subtracting the second value from the first. If the two values are numbers, return the difference. If the two values are dates, return the difference in milliseconds. If the two values are a date and a number in milliseconds, return the resulting date. Accepts two argument expressions. If the two values are a date and a number, specify the date argument first as it is not meaningful to subtract a date from a number.
\$trunc	Truncates a number to its integer.

## String Operators

String expressions, with the exception of `$concat`, only have a well-defined behavior for strings of ASCII characters.

`$concat` behavior is well-defined regardless of the characters used.

Name	Description
\$concat	Concatenates any number of strings.
\$indexOfBytes	Searches a string for an occurrence of a substring and returns the UTF-8 byte index of the first occurrence. If the substring is not found, returns <code>-1</code> .
\$indexOfCP	Searches a string for an occurrence of a substring and returns the UTF-8 code point index of the first occurrence. If the substring is not found, returns <code>-1</code> .
\$split	Splits a string into substrings based on a delimiter. Returns an array of substrings. If the delimiter is not found within the string, returns an array containing the original string.
\$strLenBytes	Returns the number of UTF-8 encoded bytes in a string.
\$strLenCP	Returns the number of UTF-8 code points  in a string.
\$strcasecmp	Performs case-insensitive string comparison and returns: <code>0</code> if two strings are equivalent, <code>1</code> if the first string is greater than the second, and <code>-1</code> if the first string is less than the second.
\$substr	Deprecated. Use <code>\$substrBytes</code> or <code>\$substrCP</code> .


Name	Description
 \$substrBytes	Returns the substring of a string. Starts with the character at the specified UTF-8 byte index (zero-based) in the string and continues for the specified number of bytes.
\$substrCP	Returns the substring of a string. Starts with the character at the specified UTF-8 code point (CP)  index (zero-based) in the string and continues for the number of code points specified.
\$toLower	Converts a string to lowercase. Accepts a single argument expression.
\$toUpper	Converts a string to uppercase. Accepts a single argument expression.

Text Search Operators

Name	Description
\$meta	Access text search metadata.

Array Operators

Name	Description
\$arrayElemAt	Returns the element at the specified array index.
\$arrayToObject	Converts an array of key value pairs to a document.
\$concatArrays	Concatenates arrays to return the concatenated array.
\$filter	Selects a subset of the array to return an array with only the elements that match the filter condition.
\$in	Returns a boolean indicating whether a specified value is in an array.
\$indexOfArray	Searches an array for an occurrence of a specified value and returns the array index of the first occurrence. If the substring is not found, returns <code>-1</code> .
\$isArray	Determines if the operand is an array. Returns a boolean.
\$map	Applies a subexpression to each element of an array and returns the array of resulting values in order. Accepts named parameters.
\$objectToArray	Converts a document to an array of documents representing key-value pairs.
\$range	Outputs an array containing a sequence of integers according to user-defined inputs.

Name	Description
 \$reduce	Applies an expression to each element in an array and combines them into a single value.
\$reverseArray	Returns an array with the elements in reverse order.
\$size	Returns the number of elements in the array. Accepts a single expression as argument.
\$slice	Returns a subset of an array.
\$zip	Merge two arrays together.

Variable Operators

Name	Description
\$let	Defines variables for use within the scope of a subexpression and returns the result of the subexpression. Accepts named parameters.


Literal Operators

Name	Description
\$literal	Return a value without parsing. Use for values that the aggregation pipeline may interpret as an expression. For example, use a \$literal expression to a string that starts with a \$ to avoid parsing as a field path.

Date Operators

Name	Description
\$dayOfYear	Returns the day of the year for a date as a number between 1 and 366 (leap year).
\$dayOfMonth	Returns the day of the month for a date as a number between 1 and 31.
\$dayOfWeek	Returns the day of the week for a date as a number between 1 (Sunday) and 7 (Saturday).
\$year	Returns the year for a date as a number (e.g. 2014).
\$month	Returns the month for a date as a number between 1 (January) and 12 (December).
\$week	Returns the week number for a date as a number between 0 (the partial week that precedes the first Sunday of the year) and 53 (leap year).
\$hour	Returns the hour for a date as a number between 0 and 23.



Name	Description
 <code>\$minute</code>	Returns the minute for a date as a number between 0 and 59.
<code>\$second</code>	Returns the seconds for a date as a number between 0 and 60 (leap seconds).
<code>\$millisecond</code>	Returns the milliseconds of a date as a number between 0 and 999.
<code>\$dateToString</code>	Returns the date as a formatted string.
<code>\$isoDayOfWeek</code>	Returns the weekday number in ISO 8601 format, ranging from 1 (for Monday) to 7 (for Sunday).
<code>\$isoWeek</code>	Returns the week number in ISO 8601 format, ranging from 1 to 53. Week numbers start at 1 with the week (Monday through Sunday) that contains the year’s first Thursday.
<code>\$isoWeekYear</code>	Returns the year number in ISO 8601 format. The year starts with the Monday of week 1 (ISO 8601) and ends with the Sunday of the last week (ISO 8601).

## Conditional Expressions

Name	Description
<code>\$cond</code>	A ternary operator that evaluates one expression, and depending on the result, returns the value of one of the other two expressions. Accepts either three expressions in an ordered list or three named parameters.
<code>\$ifNull</code>	Returns either the non-null result of the first expression or the result of the second expression if the first expression results in a null result. Null result encompasses instances of undefined values or missing fields. Accepts two expressions as arguments. The result of the second expression can be null.
<code>\$switch</code>	Evaluates a series of case expressions. When it finds an expression which evaluates to <code>true</code> , <code>\$switch</code> executes a specified expression and breaks out of the control flow.

## Data Type Expressions

Name	Description
<code>\$type</code>	Return the BSON data type of the field.

## Accumulators

*Changed in version 3.2:* Some accumulators are now available in the `$project` stage. In previous versions of MongoDB , accumulators are available only for the `$group` stage.

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

When used in the `$group` stage, accumulators take as input a single expression, evaluating the expression once for each input document, and maintain their stage for the group of documents that share the same group key.

When used in the `$project` stage, the accumulators do not maintain their state. When used in the `$project` stage, accumulators take as input either a single argument or multiple arguments.

Name	Description
<code>\$sum</code>	Returns a sum of numerical values. Ignores non-numeric values.  <i>Changed in version 3.2:</i> Available in both <code>\$group</code> and <code>\$project</code> stages.
<code>\$avg</code>	Returns an average of numerical values. Ignores non-numeric values.  <i>Changed in version 3.2:</i> Available in both <code>\$group</code> and <code>\$project</code> stages.
<code>\$first</code>	Returns a value from the first document for each group. Order is only defined if the documents are in a defined order.  Available in <code>\$group</code> stage only.
<code>\$last</code>	Returns a value from the last document for each group. Order is only defined if the documents are in a defined order.  Available in <code>\$group</code> stage only.
<code>\$max</code>	Returns the highest expression value for each group.  <i>Changed in version 3.2:</i> Available in both <code>\$group</code> and <code>\$project</code> stages.
<code>\$min</code>	Returns the lowest expression value for each group.  <i>Changed in version 3.2:</i> Available in both <code>\$group</code> and <code>\$project</code> stages.
<code>\$push</code>	Returns an array of expression values for each group.  Available in <code>\$group</code> stage only.

mongoDB

Name	Description
\$addToSet	<p>Returns an array of <i>unique</i> expression values for each group. Order of the array elements is undefined.</p> <p>Available in \$group stage only.</p>
\$stdDevPop	<p>Returns the population standard deviation of the input values.</p> <p><i>Changed in version 3.2:</i> Available in both \$group and \$project stages.</p>
\$stdDevSamp	<p>Returns the sample standard deviation of the input values.</p> <p><i>Changed in version 3.2:</i> Available in both \$group and \$project stages.</p>

