Advanced Mongodb

Miti Bhat

BSON Types

- A serialization format used to store documents and make remote procedure calls in MongoDB. “BSON” is a portmanteau of the words “binary” and “JSON”. Think of BSON as a binary representation of JSON (JavaScript Object Notation) documents.

- The BSON specification is located at bsonspec.org.

Type	Number	Alias	Notes
Double	1	“double”	
String	2	“string”	
Object	3	“object”	
Array	4	“array”	
Binary data	5	“binData”	
Undefined	6	“undefined”	Deprecated.
ObjectId	7	“objectId”	
Boolean	8	“bool”	

Date	9	“date”	
Null	10	“null”	
Regular Expression	11	“regex”	
DBPointer	12	“dbPointer”	Deprecated.
JavaScript	13	“javascript”	
Symbol	14	“symbol”	Deprecated.
JavaScript (with scope)	15	“javascriptWithScope”	
32-bit integer	16	“int”	
Timestamp	17	“timestamp”	
64-bit integer	18	“long”	
Decimal128	19	“decimal”	New in version 3.4.
Min key	-1	“minKey”	
Max key	127	“maxKey”	

Aggregation Pipeline

- MongoDB's aggregation framework is modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into an aggregated result. For example:

```
db.orders.aggregate([
  { $match: { status: "A" } },
  { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
])
```

Pipeline

- The most basic pipeline stages provide *filters* that operate like queries and *document transformations* that modify the form of the output document.
- Other pipeline operations provide tools for grouping and sorting documents by specific field or fields as well as tools for aggregating the contents of arrays, including arrays of documents. In addition, pipeline stages can use operators for tasks such as calculating the average or concatenating a string.
- The pipeline provides efficient data aggregation using native operations within MongoDB, and is the preferred method for data aggregation in MongoDB.
- The aggregation pipeline can operate on a sharded collection.

Pipeline Operators and Indexes

- The $match and $sort pipeline operators can take advantage of an index when they occur at the beginning of the pipeline.
- 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.
- 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.

Mongodb Aggregation

Aggregation Commands

Name **Description**

aggregate Performs aggregation tasks such as group using the aggregation framework.

count Counts the number of documents in a collection or a view.

distinct Displays the distinct values found for a specified key in a collection or a view.

group Deprecated. Groups documents in a collection by the specified key and performs simple aggregation.

mapReduce Performs map-reduce aggregation for large data sets.

Aggregation

- db.collection.aggregate(pipeline, options)

pipeline	array	A sequence of data aggregation operations or stages. See the aggregation pipeline operators for details. *Changed in version 2.6:* The method can still accept the pipeline stages as separate arguments instead of as elements in an array; however, if you do not specify the **pipeline** as an array, you cannot specify the **options** parameter.
options	document	Optional. Additional options that **aggregate()** passes to the **aggregate** command.

Aggregation Pipeline Stages

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

Aggregation Stages

Stage	Description
$addFields	Adds new fields to documents. Similar to $project, $addFields reshapes each document in the stream; specifically, by adding new fields to output documents that contain both the existing fields from the input documents and the newly added fields.
$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.
$collStats	Returns statistics regarding a collection or view.

Aggregation

$count	Returns a count of the number of documents at this stage of the aggregation pipeline.
$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.
$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.
$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.

Aggregation

$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.
$indexStats	Returns statistics regarding the use of each index for the collection.
$limit	Passes the first *n* documents unmodified to the pipeline where *n* is the specified limit. For each input document, outputs either one document (for the first *n* documents) or zero documents (after the first *n*documents).
$listSessions	Lists all sessions that have been active long enough to propagate to the system.sessionscollection.

Aggregation

$lookup	Performs a left outer join to another collection in the *same* database to filter in documents from the "joined" collection for processing.
$match	Filters the document stream to allow only matching documents to pass unmodified into the next pipeline stage. $match uses standard MongoDB queries. For each input document, outputs either one document (a match) or zero documents (no match).
$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.
$project	Reshapes each document in the stream, such as by adding new fields or removing existing fields. For each input document, outputs one document.

Aggregation

$redact	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 $project and $match. Can be used to implement field level redaction. For each input document, outputs either one or zero documents.
$replaceRoot	Replaces a document with the specified embedded document. The operation replaces all existing fields in the input document, including the _id field. Specify a document embedded in the input document to promote the embedded document to the top level.
$sample	Randomly selects the specified number of documents from its input.
$skip	Skips the first *n* documents where *n* 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 *n* documents) or one document (if after the first *n* documents).

Aggregation

$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.
$sortByCount	Groups incoming documents based on the value of a specified expression, then computes the count of documents in each distinct group.
$unwind	Deconstructs an array field from the input documents to output a document for *each* element. Each output document replaces the array with an element value. For each input document, outputs *n*documents where *n* is the number of array elements and can be zero for an empty array.

Db Aggregation

- db.aggregate([{ <stage> }, ...])
- The following stages use the db.aggregate() method and not the db.collection.aggregate() method

Stage	Description
$currentOp	Returns information on active and/or dormant operations for the MongoDB deployment.
$listLocalSessions	Lists all active sessions recently in use on the currently connected mongos or mongod instance. These sessions may have not yet propagated to the system.sessions collection.

Grouping

{ $group: { _id: <expression>, <field1>: { <accumulator1> : <expression1> }, ... } }

The <accumulator> operator must be one of the following accumulator operators:

Name	Description
$addToSet	Returns an array of *unique* expression values for each group. Order of the array elements is undefined.
$avg	Returns an average of numerical values. Ignores non-numeric values.
$first	Returns a value from the first document for each group. Order is only defined if the documents are in a defined order.
$last	Returns a value from the last document for each group. Order is only defined if the documents are in a defined order.

Aggregation

$max	Returns the highest expression value for each group.
$mergeObjects	Returns a document created by combining the input documents for each group.
$min	Returns the lowest expression value for each group.
$push	Returns an array of expression values for each group.
$stdDevPop	Returns the population standard deviation of the input values.
$stdDevSamp	Returns the sample standard deviation of the input values.
$sum	Returns a sum of numerical values. Ignores non-numeric values.

Example

- Example: sample.js(D:\Mongodb)

$lookup (aggregation)

- Performs a left outer join to an unsharded collection in the same database to filter in documents from the “joined” collection for processing. To each input document, the $lookup stage adds a new array field whose elements are the matching documents from the “joined” collection. The $lookup stage passes these reshaped documents to the next stage.

Equality Match

```
{
   $lookup:
    {
      from: <collection to join>,
      localField: <field from the input documents>,
      foreignField: <field from the documents of the "from" collection>,
      as: <output array field>
    }
}
```

```
SELECT *, <output array field>
FROM collection
WHERE <output array field> IN
 (SELECT *
      FROM <collection to join>
      WHERE <foreignField>= <collection.localField>);
```

Specify Multiple Join Conditions with $lookup

```
db.orders.insert([
  { "_id" : 1, "item" : "almonds",
"price" : 12, "ordered" : 2 },
  { "_id" : 2, "item" : "pecans",
"price" : 20, "ordered" : 1 },
  { "_id" : 3, "item" : "cookies",
"price" : 10, "ordered" : 60 }
])
```

```
db.inventory.insert([
  { "_id" : 1, "stock_item" : "almonds", warehouse: "A",
"instock" : 120 },
  { "_id" : 2, "stock_item" : "pecans", warehouse: "A",
"instock" : 80 },
  { "_id" : 3, "stock_item" : "almonds", warehouse: "B",
"instock" : 60 },
  { "_id" : 4, "stock_item" : "cookies", warehouse: "B",
"instock" : 40 },
  { "_id" : 5, "stock_item" : "cookies", warehouse: "A",
"instock" : 80 }
])
```

```
db.orders.aggregate([
  {
    $lookup:
      {
        from: "warehouses",
        let: { order_item: "$item", order_qty: "$ordered" },
        pipeline: [
          { $match:
            { $expr:
              { $and:
                 [
                   { $eq: [ "$stock_item",  "$$order_item" ] },
                   { $gte: [ "$instock", "$$order_qty" ] }
                 ]
              }
            },
            { $project: { stock_item: 0, _id: 0 } }
        ],
        as: "stockdata"
      }
  }
])
```

Output

{ "_id" : 1, "item" : "almonds", "price" : 12, "ordered" : 2,

"stockdata" : [{ "warehouse" : "A", "instock" : 120 }, { "warehouse" : "B", "instock" : 60 }] }

{ "_id" : 2, "item" : "pecans", "price" : 20, "ordered" : 1,

"stockdata" : [{ "warehouse" : "A", "instock" : 80 }] }

{ "_id" : 3, "item" : "cookies", "price" : 10, "ordered" : 60,

"stockdata" : [{ "warehouse" : "A", "instock" : 80 }] }

```
SELECT *, stockdata
FROM orders
WHERE stockdata IN (SELECT warehouse, instock
                    FROM warehouses
                    WHERE stock_item= orders.item
                    AND instock >= orders.ordered );
```

Thank You