**Birla Institute of Technology and Science,Pilani**

**CS F212 Database Systems**

**Lab No # 9 C**

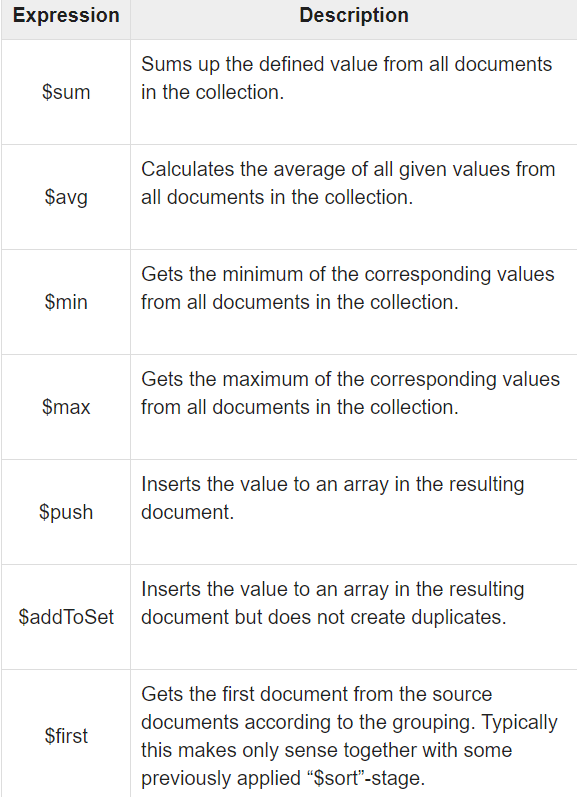
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Aggregation in MongoDB

Aggregation operations process data records and return results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.

Basic syntax for aggregate() method:

>db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION)



Aggregation Pipeline

Pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on. MongoDB also supports the same concept in the aggregation framework.

Following are possible in pipeline aggregation framework:

* $project − Used to select some specific fields from a collection.
* $match − This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.
* $group − used for grouping in aggregation
* $sort − Sorts the documents.
* $count- passes a document to the next stage that contains a count of the number of documents input to the stage.

Transactions in MongoDB

In MongoDB, an operation on a single document is atomic. Transactions can be used across multiple operations, collections, databases, and documents. Can specify read/write (CRUD) operations on existing collections.

**Count Operation**

To count the number of items. It can also be used within a transaction with [$group](https://docs.mongodb.com/manual/reference/operator/aggregation/group/#mongodb-pipeline-pipe.-group) (with a [$sum](https://docs.mongodb.com/manual/reference/operator/aggregation/sum/#mongodb-group-grp.-sum) expression) aggregation.

**Distinct Operation**

To find the distinct values for a sharded collection, use the aggregation pipeline with the [$group](https://docs.mongodb.com/manual/reference/operator/aggregation/group/#mongodb-pipeline-pipe.-group) stage instead.

Example, consider the following database.

**Orders** is a collection in a database with a list of products, their rate and their purchaser with ‘product’ and ‘total’ as field.

*db.orders.insertMany(*

*[*

*{product: "toothbrush", total: 50, customer: "Jim"},*

*{product: "guitar" , total: 400, customer: "Andy"},*

*{product: "milk" , total: 100, customer: "Mike"},*

*{product: "pizza" , total: 80, customer: "Karen"},*

*{product: "toothbrush", total: 40, customer: "pam"},*

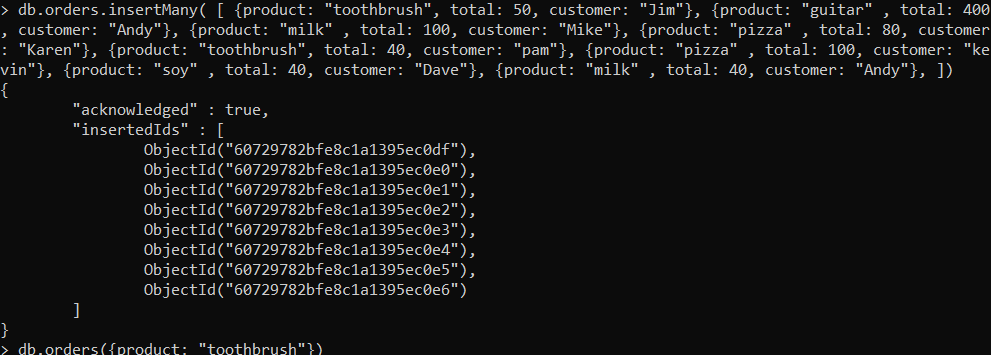
*{product: "pizza" , total: 100, customer: "kevin"},*

*{product: "soy" , total: 40, customer: "Dave"},*

*{product: "milk" , total: 40, customer: "Andy"},*

*]*

*)*

**

**Try out these aggregation methods for calculating the following queries:**

*Find out how many toothbrushes were sold*

Query: db.orders.count({product: "toothbrush"})

OUTPUT : 2.

**Count method** is used to calculate the number of products marked as “toothbrush”.

*Find list of all products sold*

Query: db.orders.distinct("product")

OUTPUT : [ "guitar", "milk", "pizza", "soy", "toothbrush" ]

**Distinct method** is used to find the **distinct** values for a specified field across a single collection and returns the results in an array.

*Find the total amount of money spent by each customer*

Query: db.orders.aggregate(

[

{$match: {} },

{$group: {\_id: "$customer", total: { $sum: "$total"} } }

]

)

OUTPUT :

{ "\_id" : "kevin", "total" : 100 }

{ "\_id" : "Karen", "total" : 80 }

{ "\_id" : "Andy", "total" : 440 }

{ "\_id" : "Dave", "total" : 40 }

{ "\_id" : "pam", "total" : 40 }

{ "\_id" : "Jim", "total" : 50 }

{ "\_id" : "Mike", "total" : 100 }

The [$group](https://docs.mongodb.com/manual/reference/operator/aggregation/group/#mongodb-pipeline-pipe.-group) stage groups the documents by \_id value to retrieve the distinct item values.

*Find how much has been spent on each product and sort it by price*

Query: db.purchase\_orders.aggregate(

[

{$match: {} },

{$group: {\_id: "$product", total: { $sum: "$total"} } },

{$sort: {total: -1}}

]

)

OUTPUT: {"\_id" : "guitar", "total" : 400 }

{ "\_id" : "pizza", "total" : 180 }

{ "\_id" : "milk", "total" : 140 }

{ "\_id" : "toothbrush", "total" : 90 }

{ "\_id" : "soy", "total" : 40 }

$**match method** takes a document that specifies the query conditions.

*Find how much money each customer has spent on toothbrushes and pizza*

Query: db.purchase\_orders.aggregate(

[

{$match: {product: {$in: ["toothbrush", "pizza"]} } },

{$group: {\_id: "$product", total: { $sum: "$total"} } },

]

)

OUTPUT:

{ "\_id" : "toothbrush", "total" : 90 }

{ "\_id" : "pizza", "total" : 180 }

Indexing

Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a *collection scan*, i.e. scan every document in a collection, to select those documents that match the query statement.

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement.

Indexes are special data structures that store a small portion of the data set in an easy-to-traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

**createIndex()** used for creating an Index in MongoDB:

>db.COLLECTION\_NAME.createIndex({KEY:1})

Here key is the name of the field on which you want to create the index and 1 is for ascending order.

## **dropIndex()** : To drop a particular index using the dropIndex() method of MongoDB.

Syntax: >db.COLLECTION\_NAME.dropIndex({KEY:1})

## **The getIndexes() method:**

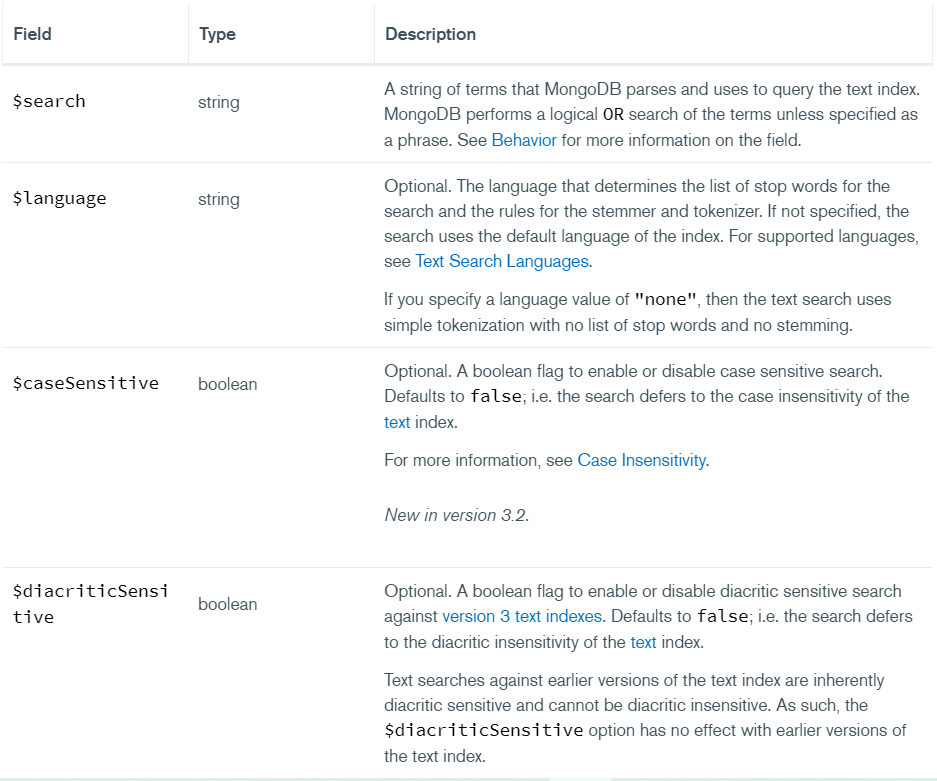
## This method returns the description of all the indexes int the collection.

## **Syntax: db.COLLECTION\_NAME.getIndexes()**

**Query and Projection Operator**

[$text](https://docs.mongodb.com/manual/reference/operator/query/text/#mongodb-query-op.-text) performs a text search on the content of the fields indexed with a [text index](https://docs.mongodb.com/manual/core/index-text/).

$text operator accepts a text query document with the following fields:

****

### **$search Field**

### In the $search field, specify a string of words that the [$text](https://docs.mongodb.com/manual/reference/operator/query/text/#mongodb-query-op.-text) operator parses and uses to query the [text index](https://docs.mongodb.com/manual/core/index-text/).

Sort by Text Search Score

A text search is performed for a particular term and uses the [$meta](https://docs.mongodb.com/manual/reference/operator/aggregation/meta/#mongodb-expression-exp.-meta) operator in the projection document to append the relevance score to each matching document.

**$meta:** Returns the metadata associated with a document, e.g. "textScore" when performing text search.

|  |
| --- |
|  |

Consider the previous database with collection as orders:

db.orders.insertMany(

[

{product: "toothbrush", total: 50, customer: "Jim"},

{product: "guitar" , total: 400, customer: "Andy"},

{product: "milk" , total: 100, customer: "Mike"},

{product: "pizza" , total: 80, customer: "Karen"},

{product: "toothbrush", total: 40, customer: "pam"},

{product: "pizza" , total: 100, customer: "kevin"},

{product: "soy" , total: 40, customer: "Dave"},

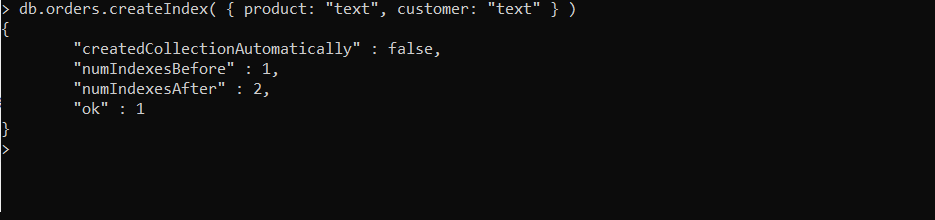
{product: "milk" , total: 40, customer: "Andy"},

]

)

Creating an Index using the product and customer details:

QUERY: db.orders.createIndex( { product: "text", customer: "text" } )



Searching for documents using index as “milk”

QUERY: db.orders.find({ $text: {$search: "milk" } })

OUTPUT:

{ "\_id" : ObjectId("60729782bfe8c1a1395ec0e6"), "product" : "milk", "total" : 40, "customer" : "Andy" }

{ "\_id" : ObjectId("60729782bfe8c1a1395ec0e1"), "product" : "milk", "total" : 100, "customer" : "Mike" }

Searching for documents using index as “Andy”

QUERY: db.orders.find({ $text: {$search: "Andy" } })

OUTPUT:

{ "\_id" : ObjectId("60729782bfe8c1a1395ec0e6"), "product" : "milk", "total" : 40, "customer" : "Andy" }

{ "\_id" : ObjectId("60729782bfe8c1a1395ec0e0"), "product" : "guitar", "total" : 400, "customer" : "Andy" }

**Searching and sorting using textScore**

QUERY:

db.orders.find(

{ $text: { $search: "toothbrush" } },

{ score: { $meta: "textScore" } }

).sort( { score: { $meta: "textScore" } } )

Using $search for a specific string, and using textScore to calculate the metadata for the searched string in the document.

OUTPUT:

{ "\_id" : ObjectId("60729782bfe8c1a1395ec0df"), "product" : "toothbrush", "total" : 50, "customer" : "Jim", "score" : 1.1 }

{ "\_id" : ObjectId("60729782bfe8c1a1395ec0e3"), "product" : "toothbrush", "total" : 40, "customer" : "pam", "score" : 1.1 }

For more Reference and Operators goto or view from the following link:

<https://docs.mongodb.com/manual/reference/operator/query/text/>

<https://docs.mongodb.com/manual/reference/command/>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*