1. **MongoDB Index**

An index in MongoDB is a special data structure that holds the data of few fields of documents on which the index is created. Indexes improve the speed of search operations in database because instead of searching the whole document, the search is performed on the indexes that holds only few fields. On the other hand, having too many indexes can hamper the performance of insert, update and delete operations because of the additional write and additional data space used by indexes.

**Default \_id index :**

The mongoDB creates a unique index on the field \_id during creation of collection.

The \_id index prevents clients from inserting two documents with the same value for the \_id field. You cannot drop this index on the \_id field.

**How to create Index**

Index type **Single field**

In addition to the MongoDB-defined \_id index, MongoDB supports the creation of user-defined ascending/descending indexes on a single field of a document

**Syntax of Index**

db.collection\_name({field\_name:1 or -1})

The value 1 is for ascending order, and the value -1 is for descending order

Example of index :

db.book.createIndex({writer:1});

After executing the above command, it shows like

db.book.createIndex({writer:1});

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

Where

numIndexesBefore – The number of indexes before command is executed {-id}

numIndexesAfter – The number of indexes after the command is executed {-id and the one we have created}

ok – Command is successful.

**Finding the indexes on collection**

Syntax of finding index on a collection :

db.collection\_name.getIndexes();

for example :

db.book.getIndexes();

**Droping the Indexes :**

db.book.dropIndex({writer:1})

After executing it shows as :

{"nIndexesWas" : 2, "ok" : 1 }

Where

nIndexesWas – It shows how many indexes were there before this command is executed.

ok – This means the command is executed successfully.

**Dropping all indexes on a collection**

db.collection\_name.dropIndexes()

db.book.dropIndexes();

Index type **Compound field**

MongoDB also supports user-defined indexes on multiple fields

Syntax of creating compound index

db.collection.createIndex( { <field1>: <type>, <field2>: <type2>, ... } )

for example :

db.book.createIndex({writer:1,category:1});

**The text indexes**

db.book.createIndex( { title: "text" } )

db.book.find({$text:{$search:"Spring"}});

1. **MongoDB aggregation**

Aggregations operations process data records and return computed 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.

In MongoDB, there are three ways to perform the aggregation.

1. Aggregation pipeline
2. Map reduce Function
3. Single purpose aggregation methods

**Aggregation Pipeline**

In Aggregation Pipeline, documents enter in multi stage pipeline that

Transforms the documents into a aggregated results.

The most basic pipeline provides filters that operate like queries and document transformations that modify the form of 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.

There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents.

**$project** – used to select some specific fields from a collection.

**$match** – It is a filtering operation, and this can reduce the amount of documents that are given as input to the next stage.

**$group** – This does actual aggregation.

**$sort** – This does sorting the documents

**$skip** - With this, it is possible to skip forward in the list of documents for a given amount of documents.

**$limit** - This limits the amount of documents to look at, by the given number starting from the current positions.

**$unwind** -This is used to unwind document that are using arrays.

There are some examples given for aggregation.

The customers collections contains some records having fields product, total and customer.

1. find out how many toothbrushes were sold

db.customers.count({product:"toothbrush"});

Output :

3

1. Find the list of all products sold

db.customers.distinct("product");

Output:

["toothbrush", "guitar", "milk", "pizza" ]

1. Find the total amount of money spent by each customer.

db.customers.aggregate(

[

{$match:{}},

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

]

);

Output :

{ "\_id" : "Dave", "total" : 4.75 }

{ "\_id" : "Karen", "total" : 13.25 }

{ "\_id" : "Tom", "total" : 199.99 }

{ "\_id" : "Mike", "total" : 20.78 }

4- Find the total amount of money spent by each customer

db.customers.aggregate(

[

{$match:{}},

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

]

);

Output :

{ "\_id" : "pizza", "total" : 13.25 }

{ "\_id" : "milk", "total" : 11.33 }

{ "\_id" : "guitar", "total" : 199.99 }

{ "\_id" : "toothbrush", "total" : 14.2 }

5- Find the total amount of money spent by each product in desending order

db.customers.aggregate(

[

{$match:{}},

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

{$sort:{total:-1}}

]

);

output :

{ "\_id" : "guitar", "total" : 199.99 }

{ "\_id" : "toothbrush", "total" : 14.2 }

{ "\_id" : "pizza", "total" : 13.25 }

{ "\_id" : "milk", "total" : 11.33 }

6 :Find the total amount of money spent by each in given customers

db.customers.aggregate(

[

{$match:{customer:{$in:["Mike","Karen"]}}},

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

{$sort:{total:-1}}

]

);

Output :

{ "\_id" : "Mike", "total" : 20.78 }

{ "\_id" : "Karen", "total" : 13.25 }

we have grouped documents by field customer and on each occurrence of by customer previous value of sum is incremented.

**$sum** – sums the defined values from all documents in the collection.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", total : {$sum : "$total"}}}]);

Output :

{ "\_id" : "Dave", "total" : 4.75 }

{ "\_id" : "Karen", "total" : 13.25 }

{ "\_id" : "Tom", "total" : 199.99 }

{ "\_id" : "Mike", "total" : 20.78 }

**$avg** - Calculates the average of all given values from all documents in the collection.

Query : db.customers.aggregate([{$group : {\_id : "$customer", total : {$avg : "$total"}}}]);

Output :

{ "\_id" : "Dave", "total" : 4.75 }

{ "\_id" : "Karen", "total" : 6.625 }

{ "\_id" : "Tom", "total" : 199.99 }

{ "\_id" : "Mike", "total" : 6.926666666666667 }

**$min** - Gets the minimum of the corresponding values from all documents in the collection.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", total : {$min : "$total"}}}]);

Output :

{ "\_id" : "Dave", "total" : 4.75 }

{ "\_id" : "Karen", "total" : 4.75 }

{ "\_id" : "Tom", "total" : 199.99 }

{ "\_id" : "Mike", "total" : 4.7 }

**$max** - Gets the maximum of the corresponding values from all documents in the collection.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", total : {$max : "$total"}}}]);

**$push** - Inserts the value to an array in the resulting document.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", product : {$push: "$product"}}}]);

Output :

{ "\_id" : "Dave", "product" : [ "pizza" ] }

{ "\_id" : "Karen", "product" : [ "pizza", "toothbrush" ] }

{ "\_id" : "Tom", "product" : [ "guitar" ] }

{ "\_id" : "Mike", "product" : [ "toothbrush", "milk", "toothbrush" ] }

**$addToSet** - Inserts the value to an array in the resulting document but does not create duplicates.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", product : {$addToSet : "$product"}}}]);

Output :

{ "\_id" : "Dave", "product" : [ "pizza" ] }

{ "\_id" : "Karen", "product" : [ "toothbrush", "pizza" ] }

{ "\_id" : "Tom", "product" : [ "guitar" ] }

{ "\_id" : "Mike", "product" : [ "milk", "toothbrush" ] }

**$first** - Gets the first document from the source documents according to the grouping.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", first\_product : {$first : "$product"}}}]);

Output :

{ "\_id" : "Dave", "first\_product" : "pizza" }

{ "\_id" : "Karen", "first\_ product " : "pizza" }

{ "\_id" : "Tom", "first\_ product " : "guitar" }

{ "\_id" : "Mike", "first\_ product " : "toothbrush" }

$last - Gets the last document from the source documents according to the grouping.

Query :

db.customers.aggregate([{$group : {\_id : "$customer", last\_product : {$first : "$product"}}}]);

Output :

{ "\_id" : "Dave", "last\_product" : "pizza" }

{ "\_id" : "Karen", "last\_product" : "toothbrush" }

{ "\_id" : "Tom", "last\_product" : "guitar" }

{ "\_id" : "Mike", "last\_product" : "toothbrush" }