

Indexing in MongoDB

- **Default `_id` Index:**

- MongoDB creates a ***unique index*** on the ***_id*** field during the creation of a collection and we cannot drop this index.
- The ***_id index*** prevents clients from inserting two documents with the same value for the `_id` field.

- **Single Field Index:**

- It is used to create user-defined ascending/descending indexes on a single field of a document.

`db.collection.createIndex({ field_name: -1/1 })`

- Use 1 asc order and -1 desc order

- We can view the index names using: **`db.collection.getIndexes()`**

- **Compound Index:** `db.students.createIndex({roll:1, name:1})`

- **Unique Index:** `db.students.createIndex({roll:1}, {unique:true})`

Aggregation in MongoDB

- Aggregation allows to perform operations on data records in the collection and return computed results.
- It **groups values from multiple documents** together and can perform variety of operation on the grouped data **to return a single value**.
- *Aggregation* operations get the information which is already available in the database and organizes it.
- MongoDB provides three ways to perform aggregation:
 - the **aggregation pipeline**,
 - **single purpose aggregation methods** (distinct, count) and
 - the **map-reduce function**
- To perform aggregation operation, aggregate() method is used.

Aggregation Pipeline

- Pipeline means the possibility to execute an operation on some input and use the output as the input for the next operation and so on.
- In MongoDB, there is a set of possible stages/operations.
- At each stage set of documents are taken as input and produce a resulting set of documents as output.
- The resulted set of documents are then used for next stage and so on.

Aggregation Stages

- **\$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** – This does the actual aggregation as discussed above.
- **\$sort** – Sorts the documents.
- **\$skip** – With this, it is possible to skip a given number of documents in the list 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 array as value for any field. When using an array, this operation defragments/deconstructs the array elements into individual documents. Thus, with this stage we will increase the amount of documents for the next stage.

\$unwind stage

Consider the `clothing` collection:

```
db.clothing.insertMany([
  { "_id" : 1, "item" : "Shirt", "sizes": [ "S", "M", "L" ] },
  { "_id" : 2, "item" : "Shorts", "sizes" : [ ] },
  { "_id" : 3, "item" : "Hat", "sizes": "M" },
  { "_id" : 4, "item" : "Gloves" },
  { "_id" : 5, "item" : "Scarf", "sizes" : null }
])
```

- We expand the *sizes* array with `$unwind` operation:

```
db.clothing.aggregate( [ { $unwind: "$sizes" } ] )
```

The `$unwind` operation returns:

```
{ _id: 1, item: 'Shirt', sizes: 'S' },
{ _id: 1, item: 'Shirt', sizes: 'M' },
{ _id: 1, item: 'Shirt', sizes: 'L' },
{ _id: 3, item: 'Hat', sizes: 'M' }
```

Aggregation Method

- Group by dept. name and sum of sal (i.e. add salary of all faculties)

```
db.faculty.aggregate([ { $group:{ _id: "$d_name", sum_sal: {$sum: "$sal" } } } ] );
```

- `_id` indicates on which field we have to perform group by.
- It will sum up all `sal` based upon `d_name` (group by).
- `sum_sal` is the name given for the field in resultant output document.
- Some aggregate operators offered by MongoDB: `$avg`, `$min`, `$max`, `$first`, `$last`

Find the cust_id and their total amount for customers with status 'A'

Collection
↓
db.orders.aggregate([
 \$match stage → { \$match: { status: "A" } },
 \$group stage → { \$group: { _id: "\$cust_id", total: { \$sum: "\$amount" } } }
])

{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }

\$match →

{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }

\$group →

Results	
{	<code>_id: "A123",</code>
	<code>total: 750</code>
}	
{	<code>_id: "B212",</code>
	<code>total: 200</code>
}	

distinct() & count() functions

`db.orders.distinct("cust_id")`

<pre>{ cust_id: "A123", amount: 500, status: "A" }</pre>
<pre>{ cust_id: "A123", amount: 250, status: "A" }</pre>
<pre>{ cust_id: "B212", amount: 200, status: "A" }</pre>
<pre>{ cust_id: "A123", amount: 300, status: "D" }</pre>

orders

`distinct` → ["A123", "B212"]

Counting documents in a collection:

`db.col_name.count();` → all docs

`db.col_name.count({status: 'A'});` → 3 docs

Aggregation Example

```
db.purchase_orders.insertMany( [  
  {product: "toothbrush", total: 4.75, customer: "Mike"},  
  {product: "guitar", total: 199.99, customer: "Tom"},  
  {product: "milk", total: 11.33, customer: "Mike"},  
  {product: "pizza", total: 8.50, customer: "Karen"},  
  {product: "toothbrush", total: 4.75, customer: "Karen"},  
  {product: "pizza", total: 4.75, customer: "Dave"},  
  {product: "toothbrush", total: 4.75, customer: "Mike"}  
]);
```

Find how much money has been earned by selling the products toothbrushes and pizza.

```
db.purchase_orders.aggregate( [  
  {$match: {product: {$in: ['toothbrush','pizza'] } } },  
  {$group: { _id: '$product', earning: {$sum: '$total' } } }  
])
```

Queries to solve:

1. Find out how many toothbrushes were sold
2. Find the list of all sold products
3. Find the total amount of money spent by each customer
4. Find how much has been spent on each product and sort it by amount spent
5. Find the product with least earnings.
6. Find how much money each customer has spent on toothbrushes and pizza
7. Find the customer who has given highest business for the product **toothbrush**

Solution:

1. `db.purchase_orders.count({product: 'toothbrush'})`
2. `db.purchase_orders.distinct("product")`
3. `db.purchase_orders.aggregate([
 { $match: {} },
 { $group: { _id: "$customer", money: { $sum: "$total" } } }
])`

```
4. db.purchase_orders.aggregate([
  {$match: {} },
  {$group: {_id:'$product', expenses:{$sum:"$total" } } },
  {$sort: {expenses: 1} },
])
```

```
5. db.purchase_orders.aggregate([
  {$match: {} },
  {$group: { _id: "$product", money: { $sum: "$total" } } },
  {$sort: {money: 1}},
  {$limit:1}
])
```

```
6. db.purchase_orders.aggregate([
  {$match: { product:{ $in: ['toothbrush', 'pizza'] } } },
  {$group: { _id:"$customer", money:{ $sum:"$total" } } }
])
```

```
7. db.purchase_orders.aggregate([
  {$match: { product: 'toothbrush' } },
  {$group: { _id: "$customer", earning: {$sum: "$total" } } },
  {$sort: {earning:-1} },
  {$limit:1}
])
```