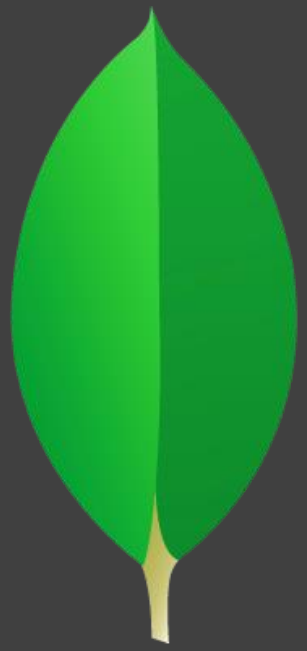


MangoDB Class

#MangoDB Notes

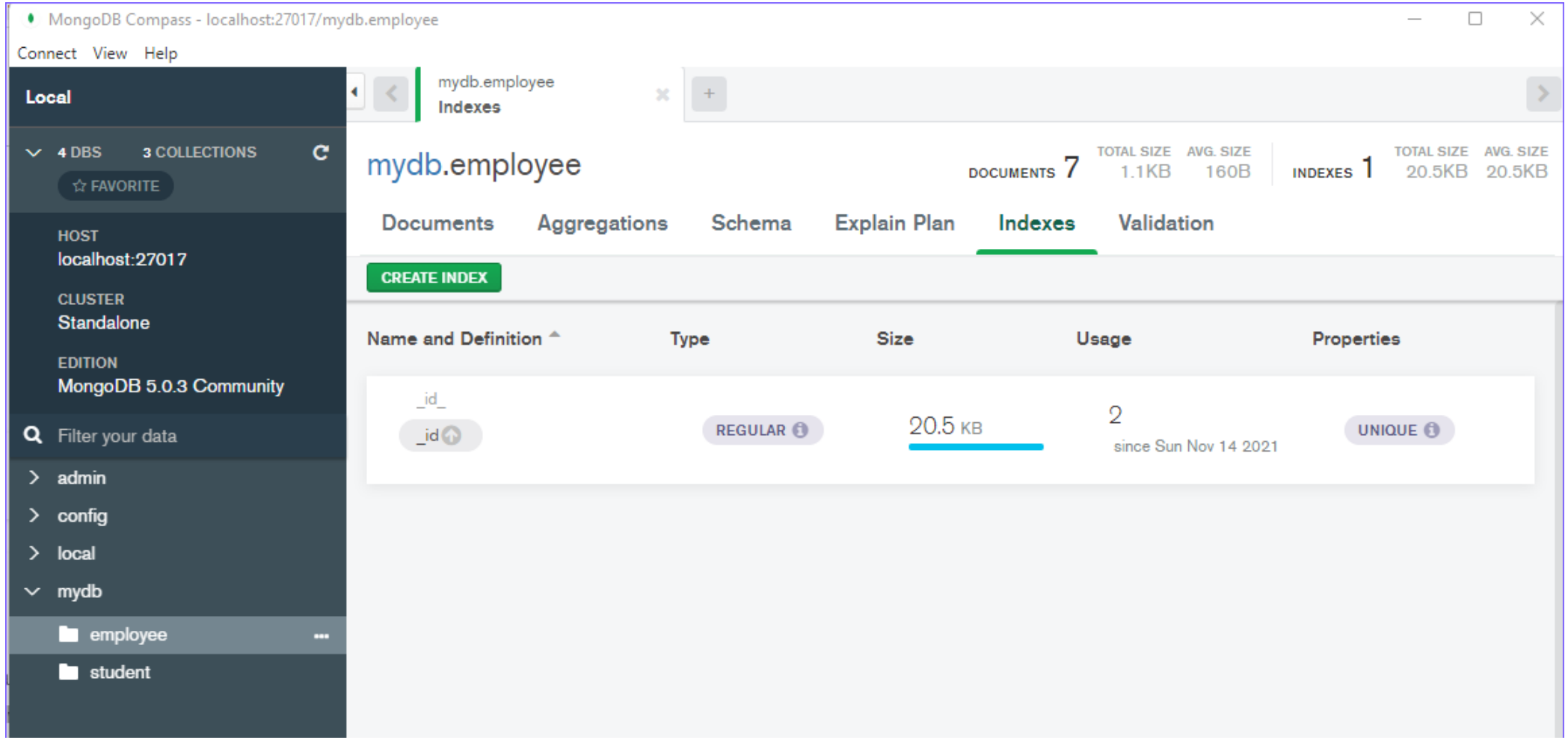
MongoDB Index

What is Indexing?

- Indexes helps to solve queries more efficiently. (Fastest way to Find information)
- Indexes are a special data structure used to locate the record in the given table very quickly without being required to traverse through every record in the table.
- MongoDB uses these indexes to limit the number of documents that had to be searched in a collection.
- By Default every collection will have an index on “_id” Key

https://docs.mongodb.com/manual/indexes/

Example


MongoDB Compass - localhost:27017/mydb.employee
Connect View Help
Local
4 DBS 3 COLLECTIONS
FAVORITE
HOST
localhost:27017
CLUSTER
Standalone
EDITION
MongoDB 5.0.3 Community
Filter your data
admin
config
local
mydb
employee
student
mydb.employee
Indexes
mydb.employee
DOCUMENTS 7
TOTAL SIZE 1.1KB
AVG. SIZE 160B
INDEXES 1
TOTAL SIZE 20.5KB
AVG. SIZE 20.5KB
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
CREATE INDEX
Name and Definition
Type
Size
Usage
Properties
_id_
_id
REGULAR
20.5 KB
2
since Sun Nov 14 2021
UNIQUE

Advantages of Indexing

- Indexes improve the speed of search operations in database .

- Instead of searching the whole document, the search is performed on the indexes that holds only few fields.

Best case and Worst case of Indexing

- The biggest advantage of indexing is that it speeds up your find, update and delete queries.

- Quite naturally because it is easier to search for the elements based on the indexed field.

1. It takes up memory (obviously).
2. It slows down write queries.

The disadvantages of indexing is that

- The write queries will obviously be slowed down because every time you make a write query you need to update the indexed field list in the collection as well and sort it again based on that field

Types of Index

- Single field index
- Compound index
- Multi key index
- Geospatial index
- Text index
- Hashed index

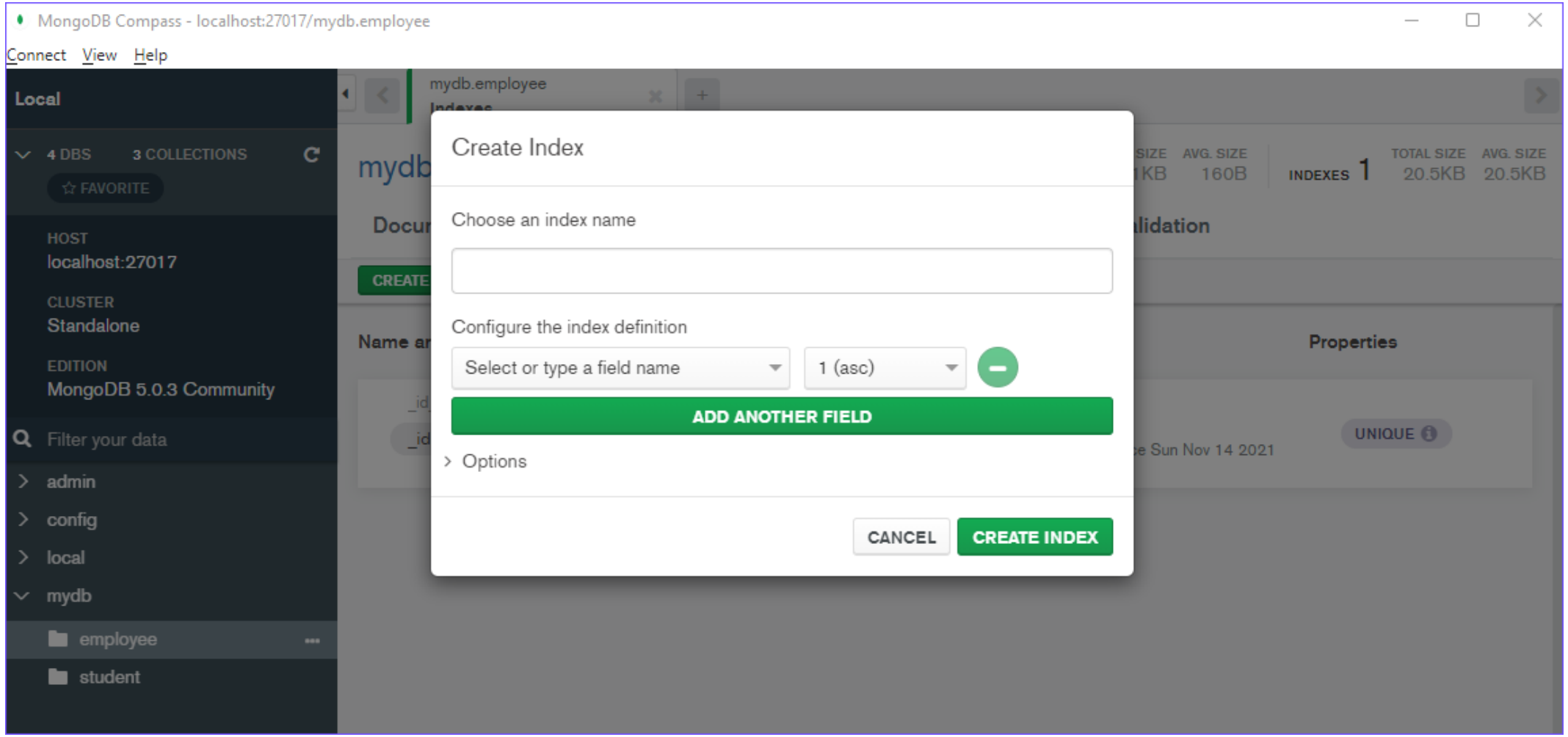
Index Type

Index Type	Description
Single field index	Used to create an index on a single field and it can be a user defined as well apart from the default _id one.
Compound index	MongoDB supports the user-defined indexes on multiple fields.
Multi key index	MongoDB uses multi key indexes basically to store the arrays. MongoDB creates a separate index for each element in an array. MongoDB intelligently identifies to create a multi key index if the index contains elements from an array.
Geospatial index	Used to support the queries required for the geospatial coordinate data.
Text index	This index is used to search for a string content in a collection
Hashed index	Used for hash based Sharding

Creating indexes

- When creating documents in a collection, MongoDB creates a unique index using the _id field.
- MongoDB refers to this as the **Default _id Index**. This default index cannot be dropped from the collection.

GUI


MongoDB Compass - localhost:27017/mydb.employee
Connect View Help
Local
4 DBS 3 COLLECTIONS
FAVORITE
HOST
localhost:27017
CLUSTER
Standalone
EDITION
MongoDB 5.0.3 Community
Filter your data
admin
config
local
mydb
employee
student
mydb.employee
Create Index
Choose an index name
Configure the index definition
Select or type a field name
1 (asc)
ADD ANOTHER FIELD
Options
CANCEL
CREATE INDEX
INDEXES 1
Properties
UNIQUE

Student Collection Data

```
mydb> db.student.find()
[
  {
    _id: ObjectId("618e338a32d48ca345e0355e"),
    name: 'abc',
    subject: 'php',
    mark: 85
  },
  {
    _id: ObjectId("618e33b932d48ca345e0355f"),
    name: 'xyz',
    subject: 'php',
    mark: 75
  },
  {
    _id: ObjectId("618e33d032d48ca345e03560"),
    name: 'pqr',
    subject: 'mongoDb',
    mark: 82
  },
  {
    _id: ObjectId("618e33e832d48ca345e03561"),
    name: 'jkl',
    subject: 'angular',
    mark: 72
  }
]
mydb>
```

Single Field Index

- These user-defined indexes use a single field in a document to create an index in an ascending or descending sort order (1 or -1).
- In a single field index, the sort order of the index key does not have an impact because MongoDB can traverse the index in either direction.
- When creating an index, you need to define the field to be indexed and the direction of the key (1 or -1) to indicate ascending or descending order.

Single Field Index

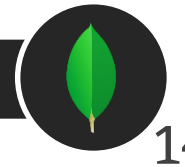
- **Syntax :-**

  db.<collection>.createIndex(<Key and Index Type>, <Options>)

- **Example :-**

  db.student.createIndex({name: 1})

```
> db.student.createIndex({name:1})
{
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "createdCollectionAutomatically" : false,
        "ok" : 1
}
>
```



Finding indexes

- You can find all the available indexes in a MongoDB collection by using the getIndexes method.
- This will return all the indexes in a specific collection.

- **Syntax :-**

  db.<collection>.getIndexes()

GetIndex

- **Syntax :-**

  db.<collection>.getIndexes()

- **Example :-**

  db.student.getIndexes()

```
> db.student.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_"
        },
        {
                "v" : 2,
                "key" : {
                        "name" : 1
                },
                "name" : "name_1"
        }
]
>
```

About Index Name

- By default, MongoDB will generate index names by concatenating the indexed keys with the direction of each key in the index using an underscore as the separator.
- For example: {name: 1} will be created as name_1.
- The best option is to use the name option to define a custom index name when creating an index.
- Indexes cannot be renamed after creation.
- The only way to rename an index is to first drop that index and recreate it using the desired name.

Student Name Index Example

```
mydb> db.student.createIndex({name:1},{name:"student name index"})
student name index
mydb>
```

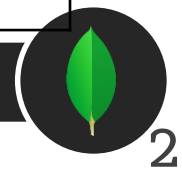
Example

- The output contains the default _id index and the user-created index student name index.

```
mydb> db.student.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { name: 1 }, name: 'student name index' }
]
mydb>
```

Optional Methods

Parameter	Type	Description
background	Boolean	Builds the index in the background so that building an index does not block other database activities. Specify true to build in the background. The default value is false.
unique	Boolean	Creates a unique index so that the collection will not accept insertion of documents where the index key or keys match an existing value in the index. Specify true to create a unique index. The default value is false.
name	string	The name of the index. If unspecified, MongoDB generates an index name by concatenating the names of the indexed fields and the sort order.
sparse	Boolean	If true, the index only references documents with the specified field. These indexes use less space but behave differently in some situations (particularly sorts). The default value is false.
expireAfterSeconds	integer	Specifies a value, in seconds, as a TTL to control how long MongoDB retains documents in this collection.
weights	document	The weight is a number ranging from 1 to 99,999 and denotes the significance of the field relative to the other indexed fields in terms of the score.
default_language	string	For a text index, the language that determines the list of stop words and the rules for the stemmer and tokenizer. The default value is English.
language_override	string	For a text index, specify the name of the field in the document that contains, the language to override the default language. The default value is language.



Unique index

- The unique property enables users to create a MongoDB index that only includes unique values. This will:
  - Reject any duplicate values in the indexed field
  - Limit the index to documents containing unique values

- **db.student.createIndex({name:1},{unique: true})**

Sort Data

- The name index will sort the data in ascending order using the name field.
- You can use the sort() method to see how the data will be represented in the index.

- **Syntax :-**

  db.student.find().sort({name:1})

After adding index student collection data

```
mydb> db.student.find().sort({name:1})
[
  {
    _id: ObjectId("618e338a32d48ca345e0355e"),
    name: 'abc',
    subject: 'php',
    mark: 85
  },
  {
    _id: ObjectId("618e33e832d48ca345e03561"),
    name: 'jkl',
    subject: 'angular',
    mark: 72
  },
  {
    _id: ObjectId("618e33d032d48ca345e03560"),
    name: 'pqr',
    subject: 'mongoDb',
    mark: 82
  },
  {
    _id: ObjectId("618e33b932d48ca345e0355f"),
    name: 'xyz',
    subject: 'php',
    mark: 75
  }
]
mydb>
```

Dropping indexes

- To delete an index from a collection, use the dropIndex method while specifying the index name to be dropped.
- You can also use the index field value for removing an index without a defined name:

- **Syntax :-**

  db.<collection>.dropIndex(<Index Name / Field Name>)

  Example:-

  db.student.dropIndex("student name index")

  db.student.dropIndex({name:1})

Example

```
mydb> db.student.dropIndex("student name index")
{ nIndexesWas: 2, ok: 1 }
mydb> db.student.getIndexes()
[ { v: 2, key: { _id: 1 }, name: '_id_' } ]
mydb>
```

dropIndexes

- The command can also drop all the indexes excluding the default _id index.
- **Example :-**

  db.studentgrades.dropIndexes()

Compound Index

- MongoDB supports a user-defined index on multiple fields as well. For this MongoDB has a compound index.
- You can use multiple fields in a MongoDB document to create a compound index.
- This type of index will use the first field for the initial sort and then sort by the preceding fields.

Example

- In this compound index, MongoDB will:
  - First sort by the name field
  - Then, within each subject value, sort by mark

```
mydb> db.student.createIndex({name:1, mark:-1})
name_1_mark_-1
mydb>
```

After adding indexes Student collection data

```
mydb> db.student.find().sort({name:1,mark:-1})
[
  {
    _id: ObjectId("618e338a32d48ca345e0355e"),
    name: 'abc',
    subject: 'php',
    mark: 85
  },
  {
    _id: ObjectId("618e33e832d48ca345e03561"),
    name: 'jkl',
    subject: 'angular',
    mark: 72
  },
  {
    _id: ObjectId("618e33d032d48ca345e03560"),
    name: 'pqr',
    subject: 'mongoDb',
    mark: 82
  },
  {
    _id: ObjectId("618e33b932d48ca345e0355f"),
    name: 'xyz',
    subject: 'php',
    mark: 75
  }
]
mydb>
```

Multikey Index

- MongoDB uses the multikey indexes to index the values stored in arrays.
- When we index a field that holds an array value then MongoDB automatically creates a separate index of each and every value present in that array.
- Using these multikey indexes we can easily find a document that contains an array by matching the items.
- In MongoDB, you don’t need to explicitly specify the multikey index because MongoDB automatically determines whether to create a multikey index if the indexed field contains an array value.

StudentDetail Collection Data

```
mydb> db.studentDetail.find()
[
  {
    _id: ObjectId("618e503332d48ca345e03562"),
    name: 'abc',
    marks: [ 85, 75, 90, 92 ]
  },
  {
    _id: ObjectId("618e503332d48ca345e03563"),
    name: 'xyz',
    marks: [ 74, 66, 55, 68 ]
  },
  {
    _id: ObjectId("618e503332d48ca345e03564"),
    name: 'jkl',
    marks: [ 80, 78, 71, 89 ]
  }
]
mydb>
```

Multikey Index Example

- create an index using the marks field.

```
mydb> db.studentdetail.createIndex({marks:1})
marks_1
mydb>
```

- Now we view the document that holds marks:[80,78,71,89]

```
mydb> db.studentDetail.find({marks:[80,78,71,89]})
[
  {
    _id: ObjectId("618e503332d48ca345e03564"),
    name: 'jkl',
    marks: [ 80, 78, 71, 89 ]
  }
]
mydb>
```

Geospatial Index

- MongoDB provides two geospatial indexes known as 2d indexes and 2d sphere indexes using these indexes we can query geospatial data.
- Here, the 2d indexes support queries that are used to find data that is stored in a two-dimensional plane.
- It only supports data that is stored in legacy coordinate pairs.
- Whereas 2d sphere indexes support queries that are used to find the data that is stored in spherical geometry.
- It supports data that is stored in legacy coordinate pairs as well as GeoJSON objects.

- **Syntax :-**

  db.<collection>.createIndex({ <Locationfield>: “2dsphere”})

MongoDB supports GEO JSON (http://geojson.org) for performing GEO location queries.

{ loc: { type: **"Point"**, coordinates: [60, 79] }, type: **"house"** }

Text Index

- MongoDB supports query operations that perform a text search of string content.
- Text index allows us to find the string content in the specified collection.
- It can include any field that contains string content or an array of string items.
- A collection can contain at most one text index. You are allowed to use text index in the compound index.

- **Syntax:**

  db.<collection>.createIndex({ <field>: “text”})

Product Collection Data

```
mydb> db.product.find()
[
  {
    _id: ObjectId("618e574632d48ca345e03565"),
    name: 'keyboard',
    desc: 'input device'
  },
  {
    _id: ObjectId("618e574632d48ca345e03566"),
    name: 'mouse',
    desc: 'input device'
  },
  {
    _id: ObjectId("618e574632d48ca345e03567"),
    name: 'printer',
    desc: 'output device'
  }
]
mydb>
```

Create Index Example

- create an index using the name and desc field.

```
mydb> db.product.createIndex({name:"text",desc:"text"})
name_text_desc_text
mydb>
```

- Now we view the document that holds **input** keyword.

```
mydb> db.product.find({$text:{$search:"input"}})
[
  {
    _id: ObjectId("618e574632d48ca345e03566"),
    name: 'mouse',
    desc: 'input device'
  },
  {
    _id: ObjectId("618e574632d48ca345e03565"),
    name: 'keyboard',
    desc: 'input device'
  }
]
mydb>
```

Hashed Index

- To maintain the entries with hashes of the values of the indexed field(mostly _id field in all collections), we use Hash Index.
- This kind of index is mainly required in the even distribution of data via sharding.
- Hashed keys are helpful to partition the data across the sharded cluster.

- **Syntax:**

db.<collection>.createIndex({ _id: “hashed” })

$match operator

- The MongoDB $match operator filters the documents to pass only those documents that match the specified condition to the next pipeline stage.

- Syntax:

  { $match: { <query> } }

Vehicle Collection

```
mydb> db.createCollection("vehicle")
{ ok: 1 }
mydb> db.vehicle.insertOne({name: "Audi",model: "A1",modelYear: [2017,2019,2020]})
{
  acknowledged: true,
  insertedId: ObjectId("619252183e535cd459096101")
}
mydb> db.vehicle.insertOne({name: "BMW",model: "X3",modelYear: []})
{
  acknowledged: true,
  insertedId: ObjectId("6192538f3e535cd459096102")
}
mydb>
```

Vehicle Collection Data

```
mydb> db.vehicle.find()
[
  {
    _id: ObjectId("619252183e535cd459096101"),
    name: 'Audi',
    model: 'A1',
    modelYear: [ 2017, 2019, 2020 ]
  },
  {
    _id: ObjectId("6192538f3e535cd459096102"),
    name: 'BMW',
    model: 'X3',
    modelYear: []
  }
]
mydb>
```

$match Example

```
mydb> db.vehicle.aggregate([{$match: {name:"Audi"}}])
[
  {
    _id: ObjectId("619252183e535cd459096101"),
    name: 'Audi',
    model: 'A1',
    modelYear: [ 2017, 2019, 2020 ]
  }
]
mydb>
```

Unwind Operator

- The MongoDB $unwind operator is used to deconstruct an array field in a document and create separate output documents for each item in the array.
- MongoDB $unwind transforms complex documents into simpler documents, which increase readability and understanding.
- This also allows us to perform additional operations, like grouping and sorting on the resulting output.

- Syntax:

  { $unwind: <field path> }

- Points to remember:
  - If the value of a field is not an array, db.collection.aggregate() generates an error.
  - If the specified path for a field does not exist in an input document, the pipeline ignores the input document and displaying no output.
  - If the array is empty in an input document, the pipeline ignores the input document and displaying no output.

Vehicle Collection

```
mydb> db.createCollection("vehicle")
{ ok: 1 }
mydb> db.vehicle.insertOne({name: "Audi",model: "A1",modelYear: [2017,2019,2020]})
{
  acknowledged: true,
  insertedId: ObjectId("619252183e535cd459096101")
}
mydb> db.vehicle.insertOne({name: "BMW",model: "X3",modelYear: []})
{
  acknowledged: true,
  insertedId: ObjectId("6192538f3e535cd459096102")
}
mydb>
```

Vehicle Collection Data

```
mydb> db.vehicle.find()
[
  {
    _id: ObjectId("619252183e535cd459096101"),
    name: 'Audi',
    model: 'A1',
    modelYear: [ 2017, 2019, 2020 ]
  },
  {
    _id: ObjectId("6192538f3e535cd459096102"),
    name: 'BMW',
    model: 'X3',
    modelYear: []
  }
]
mydb>
```

$unwind Example

```
mydb> db.vehicle.aggregate([{$unwind: "$modelYear"}])
[
  {
    _id: ObjectId("619252183e535cd459096101"),
    name: 'Audi',
    model: 'A1',
    modelYear: 2017
  },
  {
    _id: ObjectId("619252183e535cd459096101"),
    name: 'Audi',
    model: 'A1',
    modelYear: 2019
  },
  {
    _id: ObjectId("619252183e535cd459096101"),
    name: 'Audi',
    model: 'A1',
    modelYear: 2020
  }
]
mydb>
```

$project

- $project lets you specify what fields you want to have returned in the documents returned by your aggregation.
- This is not limited to existing fields but can include new fields that are computed.

- **Syntax :-**
  - db.collectionName.aggregate([{ $project : { <field>: 1 } }])

- You need to set a list of fields with value **1 or 0**.
- **1** is used to show the field while **0** is used to hide the fields.
- You can also use **true** to show the field and **false** to hide the fields.

Product Collection Data

```
mydb> db.product.find()
[
  {
    _id: ObjectId("618e574632d48ca345e03565"),
    name: 'keyboard',
    desc: 'input device'
  },
  {
    _id: ObjectId("618e574632d48ca345e03566"),
    name: 'mouse',
    desc: 'input device'
  },
  {
    _id: ObjectId("618e574632d48ca345e03567"),
    name: 'printer',
    desc: 'output device'
  }
]
mydb>
```

$project Example

```
mydb> db.product.aggregate([{$project: {name:true,desc:true,_id:0}}])
[
  { name: 'keyboard', desc: 'input device' },
  { name: 'mouse', desc: 'input device' },
  { name: 'printer', desc: 'output device' }
]
mydb>
```

$sort

- The $sort is used to sort all the documents in the aggregation pipeline and pass a sorted order to the next stage of the pipeline.

- **Syntax :-**

  { $sort: { <field1>: <sort order>, <field2>: <sort order> ... } }

  - 1 to specify ascending order.
  - -1 to specify descending order.

Student Collection Data

```
mydb> db.student.find()
[
  {
    _id: ObjectId("618e338a32d48ca345e0355e"),
    name: 'abc',
    subject: 'php',
    mark: 85
  },
  {
    _id: ObjectId("618e33b932d48ca345e0355f"),
    name: 'xyz',
    subject: 'php',
    mark: 75
  },
  {
    _id: ObjectId("618e33d032d48ca345e03560"),
    name: 'pqr',
    subject: 'mongoDb',
    mark: 82
  },
  {
    _id: ObjectId("618e33e832d48ca345e03561"),
    name: 'jkl',
    subject: 'angular',
    mark: 72
  }
]
mydb>
```

$sort Example

```
mydb> db.student.aggregate([{$sort: {mark:1}}])
[
  {
    _id: ObjectId("618e33e832d48ca345e03561"),
    name: 'jkl',
    subject: 'angular',
    mark: 72
  },
  {
    _id: ObjectId("618e33b932d48ca345e0355f"),
    name: 'xyz',
    subject: 'php',
    mark: 75
  },
  {
    _id: ObjectId("618e33d032d48ca345e03560"),
    name: 'pqr',
    subject: 'mongoDb',
    mark: 82
  },
  {
    _id: ObjectId("618e338a32d48ca345e0355e"),
    name: 'abc',
    subject: 'php',
    mark: 85
  }
]
mydb>
```

$skip

- Skips over the specified number of documents that pass into the stage and passes the remaining documents to the next stage in the pipeline.

- **Syntax :-**

  db.collectionName.aggregate([{ $skip : <positive integer> }])

Student Collection Data

```
mydb> db.student.find()
[
  {
    _id: ObjectId("618e338a32d48ca345e0355e"),
    name: 'abc',
    subject: 'php',
    mark: 85
  },
  {
    _id: ObjectId("618e33b932d48ca345e0355f"),
    name: 'xyz',
    subject: 'php',
    mark: 75
  },
  {
    _id: ObjectId("618e33d032d48ca345e03560"),
    name: 'pqr',
    subject: 'mongoDb',
    mark: 82
  },
  {
    _id: ObjectId("618e33e832d48ca345e03561"),
    name: 'jkl',
    subject: 'angular',
    mark: 72
  }
]
mydb>
```

$skip Example

```
mydb> db.student.aggregate([{$skip: 2}])
[
  {
    _id: ObjectId("618e33d032d48ca345e03560"),
    name: 'pqr',
    subject: 'mongoDb',
    mark: 82
  },
  {
    _id: ObjectId("618e33e832d48ca345e03561"),
    name: 'jkl',
    subject: 'angular',
    mark: 72
  }
]
mydb>

mydb>
```

$limit

- It simply limits the number of documents being passed to the next stage of the pipeline.

- **Syntax :-**
  - **db.collectionName.aggregate([{ $limit : <positive integer> }])**

Student Collection Data

```
mydb> db.student.find()
[
  {
    _id: ObjectId("618e338a32d48ca345e0355e"),
    name: 'abc',
    subject: 'php',
    mark: 85
  },
  {
    _id: ObjectId("618e33b932d48ca345e0355f"),
    name: 'xyz',
    subject: 'php',
    mark: 75
  },
  {
    _id: ObjectId("618e33d032d48ca345e03560"),
    name: 'pqr',
    subject: 'mongoDb',
    mark: 82
  },
  {
    _id: ObjectId("618e33e832d48ca345e03561"),
    name: 'jkl',
    subject: 'angular',
    mark: 72
  }
]
mydb>
```

$limit Example

```
mydb> db.student.aggregate([{$limit: 2}])
[
  {
    _id: ObjectId("618e338a32d48ca345e0355e"),
    name: 'abc',
    subject: 'php',
    mark: 85
  },
  {
    _id: ObjectId("618e33b932d48ca345e0355f"),
    name: 'xyz',
    subject: 'php',
    mark: 75
  }
]
mydb>
```

$count

- $count operator allows us to pass a document to the next phase of the aggregation pipeline that contains a count of the documents.

- **Syntax :-**

  $count: {<the_string>}}

- There a couple of important things to note about this syntax:
  - First, we invoke the $count operator and then specify the string.
  - In this syntax, ‘the_string’ represents the label or the name of the output field. It must be non-empty and cannot start with a dollar sign ‘$’ or dot ‘.’ character.

Student Collection Data

```
mydb> db.student.find()
[
  {
    _id: ObjectId("618e338a32d48ca345e0355e"),
    name: 'abc',
    subject: 'php',
    mark: 85
  },
  {
    _id: ObjectId("618e33b932d48ca345e0355f"),
    name: 'xyz',
    subject: 'php',
    mark: 75
  },
  {
    _id: ObjectId("618e33d032d48ca345e03560"),
    name: 'pqr',
    subject: 'mongoDb',
    mark: 82
  },
  {
    _id: ObjectId("618e33e832d48ca345e03561"),
    name: 'jkl',
    subject: 'angular',
    mark: 72
  }
]
mydb>
```

$count Example

```
mydb> db.student.aggregate([{$match:{mark:{$lt:80}}},{$count:"Count Student"}])
[ { 'Count Student': 2 } ]
mydb>
```

$addFields

- The $addFields stage allows to add new fields in the document.
- The generated output document contains the existing fields and new fields added using $addFields stage.

- **Syntax :-**

  { $addFields: { <newField1>: <expression1>, <newField2>: <expression2>,... } }

- Point to Consider for $addFields Stage:
  - $addFields appends new fields to existing documents
  - An aggregation operation can include one or more $addFields stages
  - $addFields can be added to embedded documents having arrays using dot notation
  - $addFields can be added to an existing array field using $concatArrays

Student Collection Data

```
mydb> db.Student.find()
[
  {
    _id: 1,
    name: 'abc',
    assignment: [ 14, 17 ],
    test: [ 18, 12 ],
    extraCredit: 15
  },
  {
    _id: 2,
    name: 'xyz',
    assignment: [ 18, 16 ],
    test: [ 14, 16 ],
    extraCredit: 14
  }
]
mydb>
```

$addFields Example

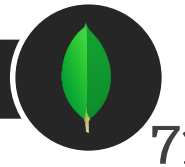
```
mydb> db.Student.aggregate([{$addFields:{assignmentTotal:{$sum:"$assignment"},testTotal:{$sum:"$test"}}}])
[
  {
    _id: 1,
    name: 'abc',
    assignment: [ 14, 17 ],
    test: [ 18, 12 ],
    extraCredit: 15,
    assignmentTotal: 31,
    testTotal: 30
  },
  {
    _id: 2,
    name: 'xyz',
    assignment: [ 18, 16 ],
    test: [ 14, 16 ],
    extraCredit: 14,
    assignmentTotal: 34,
    testTotal: 30
  }
]
mydb>
```

Adding Fields to an Embedded Document

- Embedded documents can be added with new fields using dot notation.

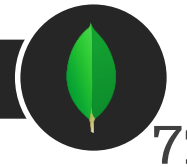
Car Collection Data

```
mydb> db.Car.find()
[
  { _id: 1, model: 'Ford', specs: { capacity: 5, wheels: 4 } },
  { _id: 2, model: 'Toyota', specs: { capacity: 5, wheels: 4 } }
]
mydb>
```



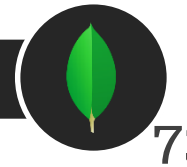
Example

```
mydb> db.Car.aggregate([{$addFields:{"specs.gear":"automatic"}}])
[
  {
    _id: 1,
    model: 'Ford',
    specs: { capacity: 5, wheels: 4, gear: 'automatic' }
  },
  {
    _id: 2,
    model: 'Toyota',
    specs: { capacity: 5, wheels: 4, gear: 'automatic' }
  }
]
mydb>
```



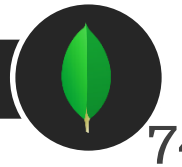
Overwriting an existing field

- If $addFields includes the existing field then the value provided in the $addField will replace the existing field value.



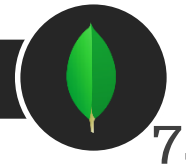
Student Collection Data

```
mydb> db.student.find()
[
  {
    _id: ObjectId("618e338a32d48ca345e0355e"),
    name: 'abc',
    subject: 'php',
    mark: 85
  },
  {
    _id: ObjectId("618e33b932d48ca345e0355f"),
    name: 'xyz',
    subject: 'php',
    mark: 75
  },
  {
    _id: ObjectId("618e33d032d48ca345e03560"),
    name: 'pqr',
    subject: 'mongoDb',
    mark: 82
  },
  {
    _id: ObjectId("618e33e832d48ca345e03561"),
    name: 'jkl',
    subject: 'angular',
    mark: 72
  }
]
mydb>
```



Example

```
mydb> db.student.aggregate([{$addFields:{subject:"HTML"}}])
[
  {
    _id: ObjectId("618e338a32d48ca345e0355e"),
    name: 'abc',
    subject: 'HTML',
    mark: 85
  },
  {
    _id: ObjectId("618e33b932d48ca345e0355f"),
    name: 'xyz',
    subject: 'HTML',
    mark: 75
  },
  {
    _id: ObjectId("618e33d032d48ca345e03560"),
    name: 'pqr',
    subject: 'HTML',
    mark: 82
  },
  {
    _id: ObjectId("618e33e832d48ca345e03561"),
    name: 'jkl',
    subject: 'HTML',
    mark: 72
  }
]
mydb>
```



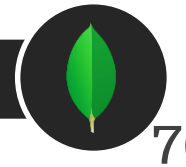
Add $addField to an Array

- $addFields allow to add new element into an Array using the $concatArrays

- $concatArrays returns the concatenated array as the result.

- **Syntax :-**

  { $concatArrays: [<array1>, <array2>, ...] }



Student Collection Data

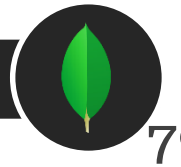
```
mydb> db.Student.find()
[
  {
    _id: 1,
    name: 'abc',
    assignment: [ 14, 17 ],
    test: [ 18, 12 ],
    extraCredit: 15
  },
  {
    _id: 2,
    name: 'xyz',
    assignment: [ 18, 16 ],
    test: [ 14, 16 ],
    extraCredit: 14
  }
]
mydb>
```

Example

```
mydb> db.Student.aggregate([{$match:{_id:1}},{$addFields:{test:{$concatArrays:["$test",[20]]}}}])
[
  {
    _id: 1,
    name: 'abc',
    assignment: [ 14, 17 ],
    test: [ 18, 12, 20 ],
    extraCredit: 15
  }
]
mydb>
```

$each

- The $each operator has brought a lot of convenience with its introduction into MongoDB.
- This operator works on arrays and can be used with the $addToSet operator and the $push operator.
- The $each operator when used with the $addToSet operator, allows us to add multiple elements to an array if they don’t exist.
- When it is used with the $push operator, we can simply append or insert elements in an array.



- **The $each Operator Syntax**

- **For $addToSet:**

  { $addToSet: { field: { $each: [value1, value2 ...] } } }

- **For $push**

  { $push: { field: { $each: [value1, value2 ...] } } }

Student Collection Data

```
mydb> db.Student.find()
[
  {
    _id: 1,
    name: 'abc',
    assignment: [ 14, 17 ],
    test: [ 18, 12 ],
    extraCredit: 15
  },
  {
    _id: 2,
    name: 'xyz',
    assignment: [ 18, 16 ],
    test: [ 14, 16 ],
    extraCredit: 14
  }
]
mydb>
```

$addToSet Example

```
mydb> db.Student.updateMany({},{$addToSet:{test:{$each:[30,40,50]}}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
mydb> db.Student.find()
[
  {
    _id: 1,
    name: 'abc',
    assignment: [ 14, 17 ],
    test: [ 18, 12, 30, 40, 50 ],
    extraCredit: 15
  },
  {
    _id: 2,
    name: 'xyz',
    assignment: [ 18, 16 ],
    test: [ 14, 16, 30, 40, 50 ],
    extraCredit: 14
  }
]
mydb>
```

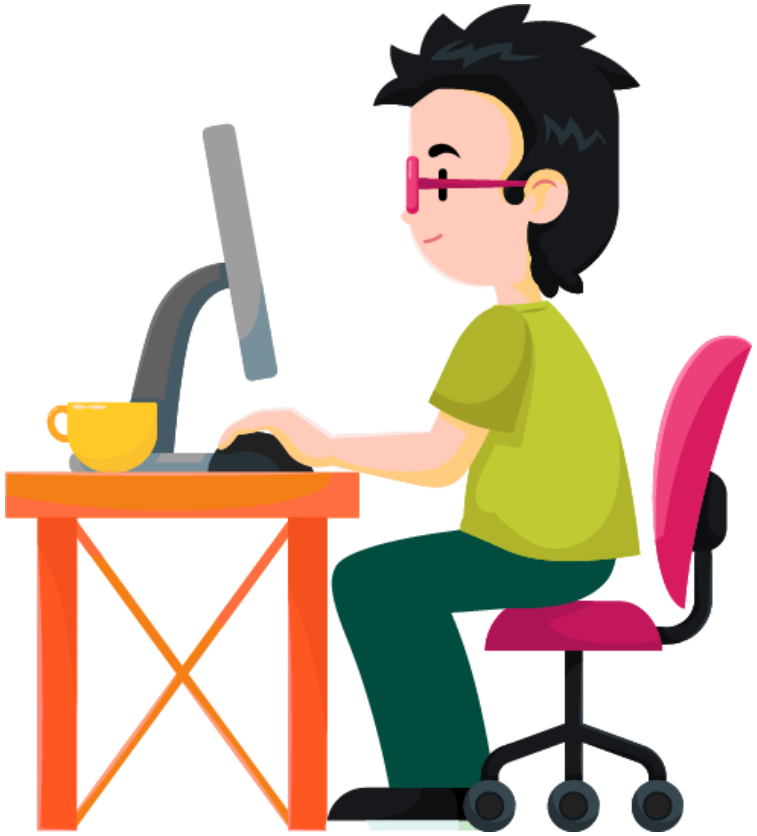
Student Collection Data

```
mydb> db.Student.find()
[
  {
    _id: 1,
    name: 'abc',
    assignment: [ 14, 17 ],
    test: [ 18, 12, 30, 40, 50 ],
    extraCredit: 15
  },
  {
    _id: 2,
    name: 'xyz',
    assignment: [ 18, 16 ],
    test: [ 14, 16, 30, 40, 50 ],
    extraCredit: 14
  }
]
mydb>
```

$push Example

```
mydb> db.Student.updateOne({name:"abc"},{$push:{assignment:{$each:[25,30,35]}}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
mydb> db.Student.find()
[
  {
    _id: 1,
    name: 'abc',
    assignment: [ 14, 17, 25, 30, 35 ],
    test: [ 18, 12, 30, 40, 50 ],
    extraCredit: 15
  },
  {
    _id: 2,
    name: 'xyz',
    assignment: [ 18, 16 ],
    test: [ 14, 16, 30, 40, 50 ],
    extraCredit: 14
  }
]
mydb>
```

Get Exclusive Video Tutorials



www.aptutorials.com

https://www.youtube.com/user/Akashtips

Get More Details

www.akashsir.com

If You Liked It !

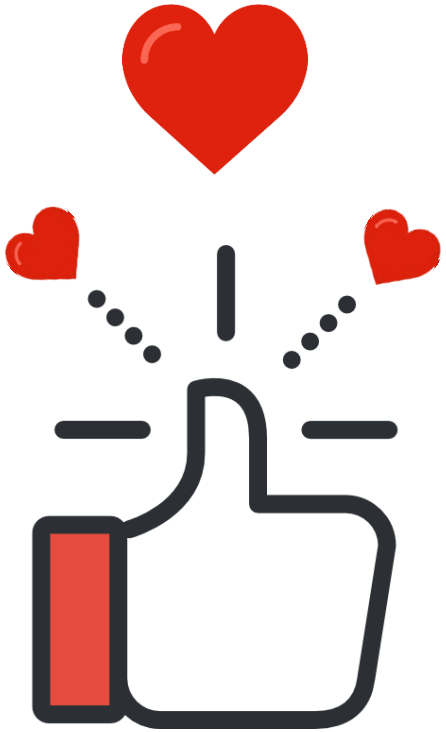
Rating Us Now

Just Dial

https://www.justdial.com/Ahmedabad/Akash-Technolabs-Navrangpura-Bus-Stop-Navrangpura/079PXX79-XX79-170615221520-S5C4_BZDET

Sulekha

https://www.sulekha.com/akash-technolabs-navrangpura-ahmedabad-contact-address/ahmedabad



Connect With Me

Akash Padhiyar

#AkashSir

www.akashsir.com

www.akashtechnolabs.com

www.akashpadhiyar.com

www.aptutorials.com

Social Info

Akash.padhiyar

Akashpadhiyar

Akash_padhiyar

+91 99786-21654

#Akashpadhiyar

#aptutorials