Doc 09 MongoDb Indexing

We are going learn indexing is done in mogngo db collection

1. indexing is done in below format

db.collection.createIndex( <key and index type specification>, <options> )

MongoDB indexes use a B-tree data structure.

2. let insert below data in the

db.indexcollection.insertMany( [

{ product: "register", status: "A", size: { height: 14, weight: 21, unit: "cm" }, instock: [ { warehouse: "A", quantity: 5 } ] , creationts: ISODate("2014-11-01T12:30:15Z")},

{ product: "book", status: "A", size: { height: 8.5, weight: 11, unit: "in" }, instock: [ { warehouse: "C", quantity: 5 } ], creationts: ISODate("2016-11-01T12:30:15Z") },

{ product: "sheet", status: "D", size: { height: 8.5, weight: 11, unit: "in" }, instock: [ { warehouse: "A", quantity: 60 } ] , creationts: ISODate("2017-11-01T12:30:15Z")},

{ product: "timetable", status: "D", size: { height: 22.85, weight: 30, unit: "cm" }, instock: [ { warehouse: "A", quantity: 40 } ] , creationts: ISODate("2018-11-01T12:30:15Z")},

{ product: "envelop", status: "A", size: { height: 10, weight: 15.25, unit: "cm" }, instock: [ { warehouse: "B", quantity: 15 }, { warehouse: "C", quantity: 35 } ], creationts: ISODate("2019-11-01T12:30:15Z") },

{ product: "register", status: "A", size: { height: 14, weight: 21, unit: "cm" }, instock: [ { warehouse: "A", quantity: 5 } ] , creationts: ISODate("2014-11-01T12:30:15Z")},

{ product: "register", status: "A", size: { height: 8.5, weight: 11, unit: "in" }, instock: [ { warehouse: "C", quantity: 5 } ], creationts: ISODate("2016-11-01T12:30:15Z") },

{ product: "register", status: "D", size: { height: 8.5, weight: 11, unit: "in" }, instock: [ { warehouse: "A", quantity: 60 } ] , creationts: ISODate("2017-11-01T12:30:15Z")},

{ product: "book", status: "D", size: { height: 22.85, weight: 30, unit: "cm" }, instock: [ { warehouse: "A", quantity: 40 } ] , creationts: ISODate("2018-11-01T12:30:15Z")},

{ product: "book", status: "A", size: { height: 10, weight: 15.25, unit: "cm" }, instock: [ { warehouse: "B", quantity: 15 }, { warehouse: "C", quantity: 35 } ], creationts: ISODate("2019-11-01T12:30:15Z") }

]);

we need indexes for data search effectively.

3. let check any present indexes.

db.indexcollection.getIndexes()

\_id index is created by default as we are aware of the primary key like RDBMS.

4. we can drop indexes as given below.

db.indexcollection.dropIndexes()

This command is going to drop all non id indexes from collection.

5. we can create index on any individual fields. There indexes are called single field indexes. below command is used to create index.

db.indexcollection.createIndex( { product: 1 } )

we can verify index creation by

db.indexcollection.getIndexes()

############ index performance ###################

var d = new Date();

var n1 = d.getTime();

db.largecollectionforindex.find({product : "register"}).count()

var d = new Date();

var n2 = d.getTime();

print("time taken : "+ (n2-n1))

db.largecollectionforindex.createIndex( { product: 1 } )

as you can see we can reduce the query execution time significantly. using indexes.

########################################

6. we can also create index on embedded field and embedded document

db.indexcollection.createIndex( { "size.height": 1 } )

This index will be useful when we are using embedded document field quite frequently like below.

db.indexcollection.find( { "size.height" : 8.5 } )

db.indexcollection.createIndex( { size: 1 } )

This index will be useful when we are using embedded document it self quite frequently like below

db.indexcollection.find( {size: { height: 10, weight: 15.25, unit: "cm" } } )

7. when we are using more then one fields then these are called compound indexes.

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

db.indexcollection.createIndex( { status : 1 , "size.height": 1 } )

db.indexcollection.getIndexes()

8. compound index support individual index also and order of fields during index creation matters.

sort order is also important in compound indexes.

let consider previous query

db.indexcollection.getIndexes()

this can support sort order of below.

db.indexcollection.find( { product: 1 , "size.height": 1 } )

db.indexcollection.find( { product: -1 , "size.height": -1 } )

but below is not supported well

db.indexcollection.find( { product: 1 , "size.height": -1 } )

or

db.indexcollection.find( { product: -1 , "size.height": 1 } )

order of fields should be the same to be effective.

9. multikey index is created when index is created on on array field as given below.

db.indexcollection.createIndex( { instock: 1 } )

db.indexcollection.find( { instock : [ { warehouse: "B", quantity: 15 }, { warehouse: "C", quantity: 35 } ] } )

10 comound multikey index where we can have atmost one array in the index and rest will be other fields.

db.indexcollection.createIndex( { product : 1, instock: 1 } )

11. hashed indexes can be created on non array type of data or simply on basic field values. These are ver commonly used for shard keys and it support shard key vary well.

db.indexcollection.createIndex( { \_id: "hashed" } )

12.Some important observation regarding multikey indexes.

it should be array

squence of sorting start from the first element i.e. if command is db.indexcollection.find( { instock: [ { warehouse: "B", quantity: 15 } ] } )

13. earlier we have created text index.

db.indexcollection.createIndex( { product: "text" } )

14. we can create multiple text indexes.

db.indexcollection.createIndex( { product: "text" , status : "text" } )

15. we can create text index on every field in the document.

db.indexcollection.createIndex( { "$\*\*": "text" } )

There is also weight associated with each text index which we can manupulate. default is 1.

16. text index can be a compound index also

db.indexcollection.createIndex( {"size.weight" : 1, "$\*\*": "text" } )

17. some properties of text indexes are case insensitive ,

diacritic Insensitivity which means language specific charecters may be ignored for same character such as é, ê, and e.

A collection can have at most one text index.

Sort operations cannot obtain sort order from a text index, even from a compound text index; i.e. sort operations cannot use the ordering in the text index.

text indexes can ve memory consuming compared to other indexes.

Building a text index is very similar to building a large multi-key index and will take longer than building a simple ordered (scalar) index on the same data.

text index can slow down insertion as it needs to add an entry in the text index.

18. last important index is 2dsphere index

A 2dsphere index supports queries that calculate geometries on an earth-like sphere.

2dsphere index supports all MongoDB geospatial queries: queries for inclusion, intersection and proximity.

For more information on geospatial queries, see Geospatial Queries.

The 2dsphere index supports data stored as GeoJSON objects and legacy coordinate pairs

For legacy coordinate pairs, the index converts the data to GeoJSON Point.

db.collection.createIndex( { <location field> : "2dsphere" } )

GeoJson Point are specific type of data to express itself.

<field>: { type: <GeoJSON type> , coordinates: <coordinates> }

location:{

type:"Point",

coordinates:[

-73.97,

40.77

]

}

19. lets insert the data.

db.geoplaces.insertMany([

{

name: "Muncipal Park",

location: { type: "Point", coordinates: [ -73.97, 40.77 ] },

category: "Parks"

} ,

{

name: "City Park",

location: { type: "Point", coordinates: [ -73.9928, 40.7193 ] },

category: "Parks"

},

{

name: "kannedy Grounds",

location: { type: "Point", coordinates: [ -73.9375, 40.8303 ] },

category: "Stadiums"

}

]

);

20. we need to query below

db.geoplaces.find(

{

location:

{ $near:

{

$geometry: { type: "Point", coordinates: [ -73.9667, 40.78 ] },

$minDistance: 1000,

$maxDistance: 5000

}

}

}

)

now lets create index for location field

db.geoplaces.createIndex( { location: "2dsphere" } )