Outlin
e

Indexing ←

Aggregatio

Indexing

Indexes support the efficient execution of queries in

Indexing Types

Single Field Indexes	• A single field index only includes data from a single field of the documents in a collection.
Compound Indexes	• A compound index includes more than one field of the documents in a collection.
Multikey Indexes	• A multikey index is an index on an array field, adding an index key for each value in the array.
Geospatial Indexes and	• Geospatial indexes support location-based searches.
Text Indexes	• Text indexes support search of string content in documents.
Hashed Index	• Hashed indexes maintain entries with hashes of the values of the indexed field and are used with sharded clusters to support hashed shard keys.

Index Properties

Index Properties The properties you can specify when building indexes.

TTL Indexes The TTL index is used for TTL collections, which expire data after a period of time.

Unique Indexes A unique index causes MongoDB to reject all documents that contain a duplicate value for the indexed field.

Sparse Indexes A sparse index does not index documents that do not have the indexed field.

Index Creation

[illegible]

Using

[illegible]

- db.CollectionName.createIndex({ KeyName: 1 or -1})

Using ensureIndex

- db.CollectionName.ensureIndex({KeyName: 1 or -1})

1 for Ascending Sorting
-1 for Descending Sorting

Index Creation

Creation

Using CreateIndex

- Single: db.stud.createIndex({ zipcode: 1})
- Compound: db.stud.createIndex({ dob: 1, zipcode: -1 })
- Unique: db.stud.createIndex({ rollno: 1 }, { unique: true })
- Sparse: db.stud.createIndex({ age: 1 }, { sparse: true })

Using ensureIndex

- Single: db.stud.ensureIndex({“name":1})
- Compound: db.stud.ensureIndex ({“address":1,“name":-1})

Index Display

Display

db.collection.getIndexes()

- Returns an array that holds a list of documents that identify and describe the existing indexes on the collection.

db.collection.getIndexStats()

- Displays a human-readable summary of aggregated statistics about an index’s B-tree data structure.
- db.<collection>.getIndexStats({ index : "<index name>" })

Index Drop

Drop

Syntax

- db.collection.dropIndex()
- db.collection.dropIndex(*index*)

Example

- db.stud.dropIndex()
- db.stud.dropIndex({ “name" : 1 })

Indexing and Querying

- create an ascending index on the field ***name*** for a collection records:

```
db.records.createIndex( { name: 1 } )
```

- This index can support an ascending sort on ***name***

: `db.records.find().sort( { name: 1 } )`

- The index can also support descending sort

```
db.records.find().sort( { a: -1 } )
```

Indexing and Querying

db.stud.findOne({rno:2}), using index {rno:1}

db.stud.find ({rno:5}), using index {rno:1}

db.stud.find({rno:{$in:[2,3]}}), using index {rno:1}

db.stud.find({age:{$gt:15}}), using index {age:1}

db.stud.find({age :{$gt:2,$lt:5}}), using index {age

:1} db.stud.count({age:19}) using index {age:1}

db.stud.distinct({branch: “Computer”}) using index

Indexing and Querying

db.stud.find({}, {name:1,age:1}), using index {name:1,age:1}

db.c.find().sort({name:1,age:1}), using index {name:1,age:1}

db.stud.update({age:20}, {age:19}) using index {age:1}
db.stud.remove({name: “Jiya”}) using index {name:1}

Indexing with Unique

db.collectionname.ensureIndex

({x:1}, {unique:true})

- Don’t allow {_id:10,x:2} and {_id:11,x:2}
- Don’t allow {_id:12} and {_id:13} (both match

What if duplicates exist before index is created?

- Normally index creation fails and the index is removed
- db.ensureIndex({x:1}, {unique:true,dropDups:true})