**Indexes:** A special kind of data structures that store only small subset of the data held in the collections documents separately. Indexes are a quicker and efficient way to traverse the data in searching values.

**Single field index:** It is used to speedup the document queries that filter the data using the field as part of the filtering condition.

To check if the collection has any indexes: db.system.indexes.find();

To find all the records where the peak height is greater than 8700. For this we can create a single field index for the height field.

General syntax for creating the index:

db.colection.createIndex({ attribute : 1 });

Check the status of the field you want to create an index on, if there is any existing index on it.

db.peaks.find( { “height” : { $gt : 8700 } }).explain(“executionStats”)

Create the index for the peaks collection on height field:

db.peaks.createIndex( { height : 1 } );

**Unique Index:** Here a second attribute apart from \_id which can be used for the uniqueness of the column that is known as unique index.

General syntax for Unique index:

db.collection.createIndex( { attribute : 1}, { “unique” : true } )

In this peaks collection, we can use mountain names as the attribute for unique indexes as the mountain names are unique and do not repeat. To check the status of the name field if it has indexes on it or not.

db.peaks.find( { “name” : “Everest” }.explain(“executionStats”);

Now to create the unique index: db.peaks.createIndex( { “name” : 1 }, { “unique” : True } )

The attribute used to create a unique index cannot have any duplicate values.

**Embedded Field Index:** When we want to create an index inside the nested document attribute then we use this. It will traverse each nested document.

To create an index on the nested document “total”

db.peaks.createIndex( { “ascents.total” : 1 } );

Using the index to search and sort:

db.peaks.find( { “ascents.total” : { $gt : 350 } } ).sort( { “ascents.total” : -1} );

Can try running the execution stats query before and after creating the index to check the execution time difference.

db.peaks.find( { “ascents.total” : { $gt : 350 } } ).sort( { “ascents.total” : -1} ).explain(“executionStats”);

**Compound Field Index**: It's a combination of two indexes where one is a primary attribute and the other is an embedded nested attribute.

db.peaks.createIndex( { “ascents.first\_winter.year” : 1, “height” : -1 } );

Here ascents.first\_winter.year” : 1 is the nested attribute and “height” : -1 will be the primary attribute as height is used to create an index earlier.

Example of compound field index to search and sort:

db.peaks.find( { “ascents.first\_winter.year” : { $gt : 1990 }, “height” : { $lt : 8600 } } ).sort( { “height” : -1} );

**Query Optimization:** Check the execution stats before and after creating the index, if there is a significant difference in time then the index is good if not then the index can be dropped.

**To drop the indexes:**

**Drop all Indexes:** db.collection\_name.dropIndexes()

**Drop single index:** db.collection\_name.dropIndex(“index\_name”)

[Name of index can be found in execution stats”] or

**Get all the indexes names:** db.collection\_name.getIndexes()

**Assignment:**

Create a unique index and a single index on the cities collection. And then drop any one index and check

1. **Unique index:**

learnbay> db.cities.createIndex( { "name" : 1 }, { "unique" : true });

name\_1

1. **Single Index**:

learnbay> db.cities.createIndex ( { "population" :1 } );

population\_1

**Checking the indexes created:**

learnbay> db.cities.getIndexes();

[

{ v: 2, key: { \_id: 1 }, name: '\_id\_' },

{ v: 2, key: { name: 1 }, name: 'name\_1', unique: true },

{ v: 2, key: { population: 1 }, name: 'population\_1' }

]

**Verifying the unique index by duplication:**

learnbay> db.cities.insertOne({name:"Istanbul", country:"Turkey", continent: "Europe", population: 14.751});

MongoServerError: E11000 duplicate key error collection: learnbay.cities index: name\_1 dup key: { name: "Istanbul" }

**Dropping the index:**

learnbay> db.cities.dropIndex("population\_1");

{ nIndexesWas: 3, ok: 1 }

learnbay> db.cities.getIndexes();

[

{ v: 2, key: { \_id: 1 }, name: '\_id\_' },

{ v: 2, key: { name: 1 }, name: 'name\_1', unique: true }

]