**Index Intro**

John/

blonde

Zoe/red

Andrew/ blonde

Jane/

brown

After indexing the array is sorted and items are found using B-TREE

 Andrew/red

Andrew/brown

Andrew/blonde

Zoe/

Blonde

John/

blonde

Jane/

brown

Before indexing

Zoe/red

{name: Zoe, hair\_color:brown, dob: ‘2001-03-23’}

 jane/

red

Zoe/

Brown

To trigger IXSCAN [name, hair], [name] are fine, however just [hair] is not fine

Compound index

Create index with more than 1 field

example

db.collection.createIndex( {field : sort\_type, field2 : sort\_type} )

where type is sort type 1, -1

multikey index

if any indexed field is array, mongo db supports only one array field while creating indexes (as the cartesian product increase exponentially )

unique indexes

db.students.createIndex({a:1}, {unique: true})

the field should be unique

partial

db.students.createIndex({a:1},{partialFilterExpression: {c:{$gt:1}}})

sparse

db.students.createIndex({a:1}, {sparse:true})

where a could be non unique

full text search

"words" : "dog shrub granite"

"words" : "dog moss ruby"

"words" : "dog moss granite"

"words" : "dog moss obsidian"

"words" : "rat moss obsidian"

"words" : "rat moss granite"

"words" : "rat moss ruby"

"words" : "rat shrub granite"

"words" : "cow shrub granite"

"words" : "cow moss ruby"

"words" : "cow moss granite"

"words" : "cow moss obsidian"

db.students.createIndex({words: “text”})

db.student.find({$text: {$search: ‘rat’}})

geospatial indexes

{ "\_id" : ObjectId("5adb3a4647ad6386409f9593"), "name" : "Tickel candy", "type" : "food", "location" : [ 41.232, -75.343 ] }

{ "\_id" : ObjectId("5adb3a1a47ad6386409f9592"), "name" : "ACE hardware store", "type" : "hardware", "location" : [ 40.323, -74.343 ] }

{ "\_id" : ObjectId("5adb39d847ad6386409f9591"), "name" : "rubys", "type" : "barber", "location" : [ 40, 74 ] }

db.world.createIndex({“location”: “2d”, type:1})

type is sorting (asc/dsc)

db.world.find(“location”: {$near: [x,y]})

geosherical indexes

geojson.org/goejson-spec.html

**Explain**

Note:

To create a million document,

for(i=0; i<100; i++){for(j=0;j<100;j++){ x=[]; for(k=0; k<100; k++){x.push({a:i, b:j,c:k, \_id:(100\*100\*i + 100 \*j +k)})}; db.students.insert(x)}}

3 modes

1. Query planner - db.students.explain()

Increase level of verbosity

1. Execution stats - db.students.explain("executionStats")
2. All plans exection - db.students.explain("allPlansExecution")

Note:

Explain doesn’t return documents, it returns pointer to docs

1. execution stats - look at following keys

"totalKeysExamined" : 100, //docs with index keys examined

"totalDocsExamined" : 100,

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 100, // documents with required data returned

Note: fetch stage is final stage, before which another matching stage is executed

Whose results are given to fetch stage

Usually read the explain object bottom to top

"inputStage" : {

"stage" : "IXSCAN",

* For each stage how many documents where returned -

"nReturned" : 100,

"executionTimeMillisEstimate" : 10,

"keyPattern" : {

"a" : 1,

"b" : 1

},

* Selectivity -

"indexBounds" : {

"a" : [

"[17.0, 17.0]"

],

"b" : [

"[55.0, 55.0]"

]

},

1. All plans execution

Show rejected plan data as well

* Covered query

Where the documents are examined only by index key.

Example – db.students.find({a:1}, {\_id:0, a:1});

Where index is on field a and we are asking to show field a only