Problem statement:

I inherited a MongoDB database server with 60 collections and 100 or so indexes.

The business users are complaining about slow report completion times.

What can I do to improve performance ?

Scope:

System tuning-
- Memory
- Process
- Disk
- Network

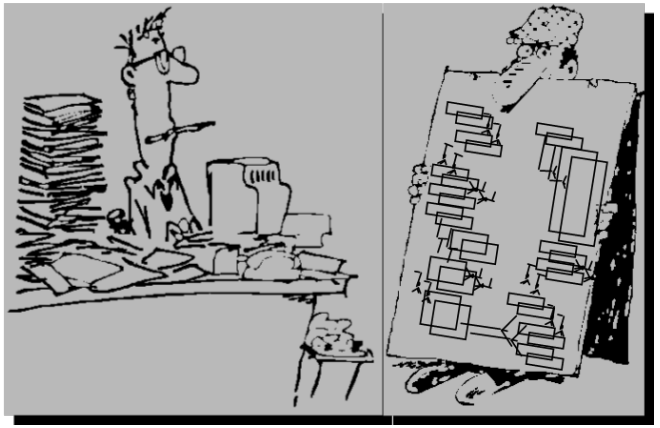
Application tuning-
- Application architecture
- Statement design
- Data model design
- Indexing, Query optimization

(Relational, so that we may compare/contrast)

- Detail command processing stages
- Can apply the 5 rules to a Rule Based query optimizer

- Apply 3 Index Negation guidelines
- Repair common query design problems-
  - Psuedo order by clause
  - OR topped queries
  - (And topped queries, non-compound)

- Drop and analyze query plans
- Articulate/control FJS, & ESR query processing patterns



What you will leave with in 60 minutes:

Queries 1 & 2:

```
SELECT * FROM phonebook
WHERE lastName LIKE “?son”;

SELECT * FROM phonebook
WHERE firstName = “Jennifer”;
```

Query 3:

```
CREATE TABLE t1 (col1, col_2, .. 80 more columns);
CREATE INDEX i1 ON t1 (col_gender);

SELECT * FROM t1
WHERE col_gender = “F” AND col_age > 50;
```

Negation of an index

1. Non-initial substring
2. Non-anchored compound/composite key
3. (Poor) selectivity of a filter

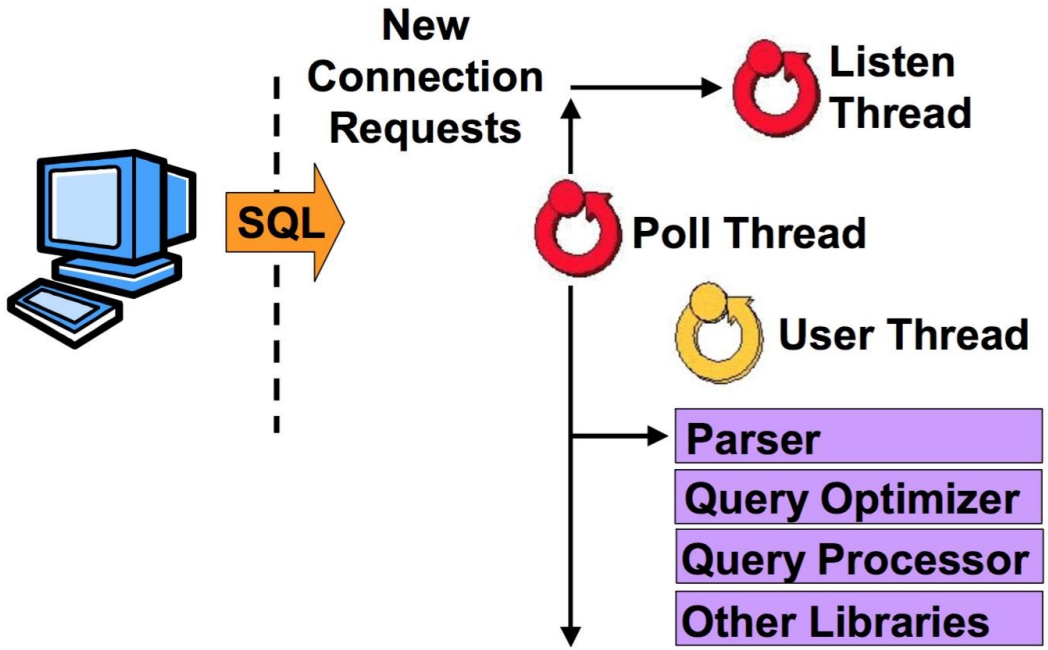
Partial negation of an index-

```
CREATE INDEX i1 ON t2 (col1, col2);
  //
SELECT * FROM t1
WHERE col1 > 100 AND col2 = “x”;
```

Exception to all above-

Covered query/key-only

(n) Stage database server back end



Query Optimizers

Rule  **Cost**  **?**

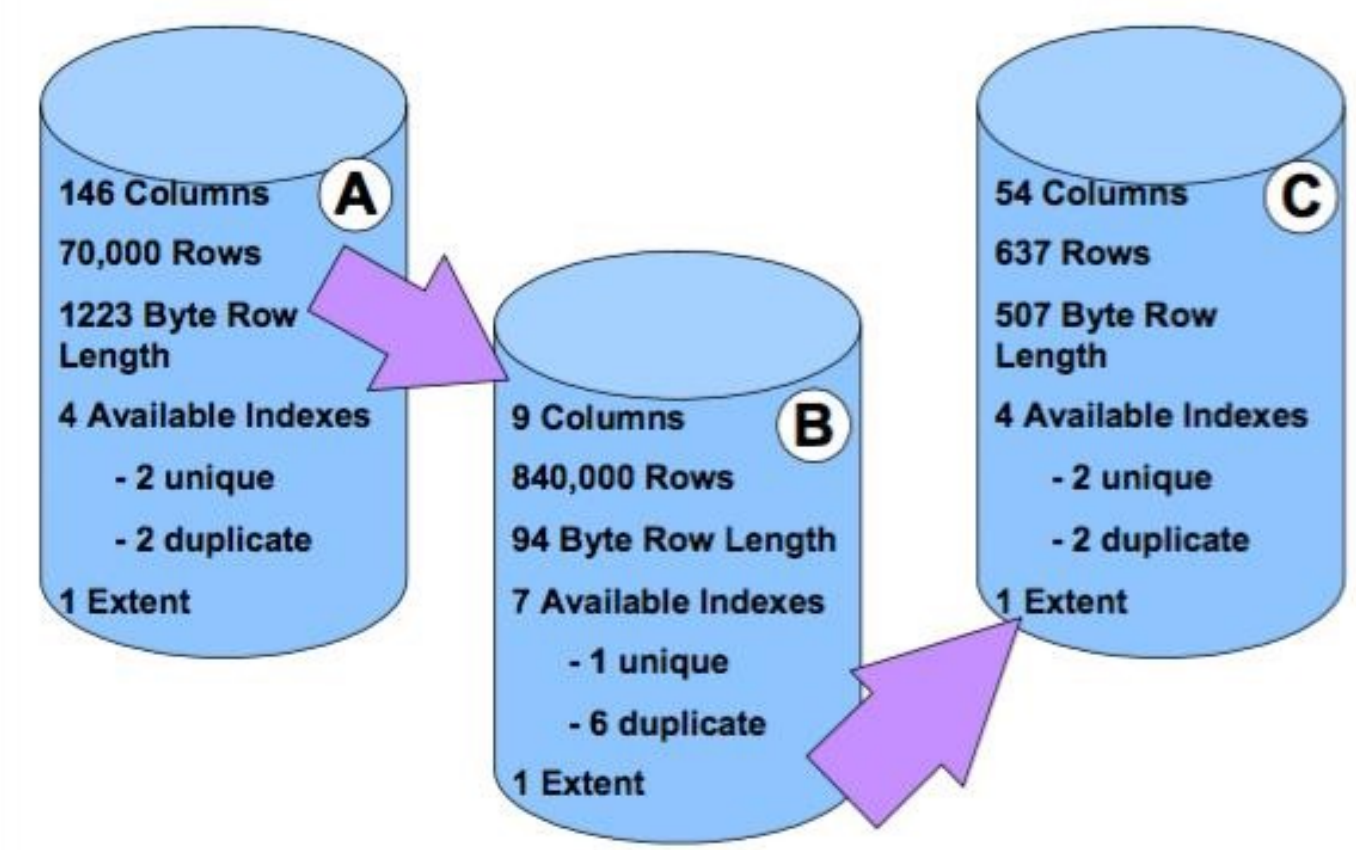
5 Rules to a rule based optimizer

1. Outer table joins
2. (Non-outer) table joins
3. Filter criteria (predicates)
4. Table size
5. Table cardinality

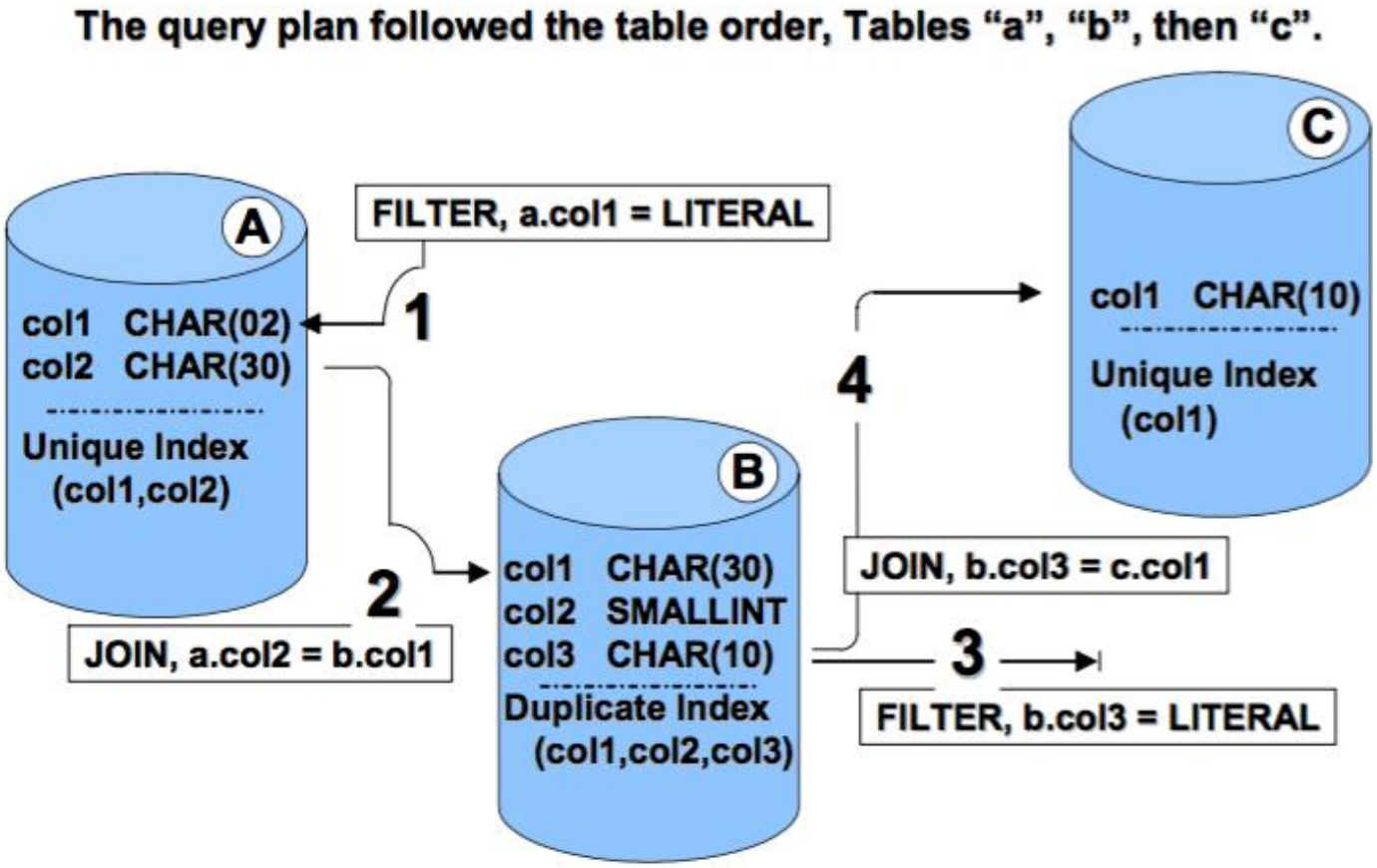
```
SELECT *
FROM orderHeader, orderLineItems
WHERE
   oh.orderNumber =
   oi.orderNumber;
```

```
SELECT *
FROM persons, OUTER automobiles
WHERE
   p.personId = a.personId;
```

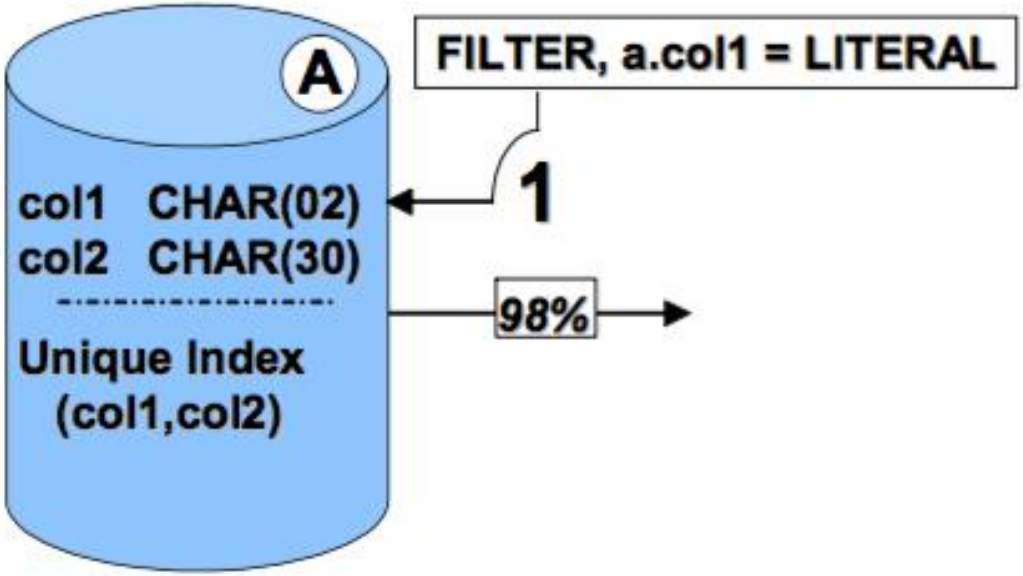
Query 4: final/larger example using SQL



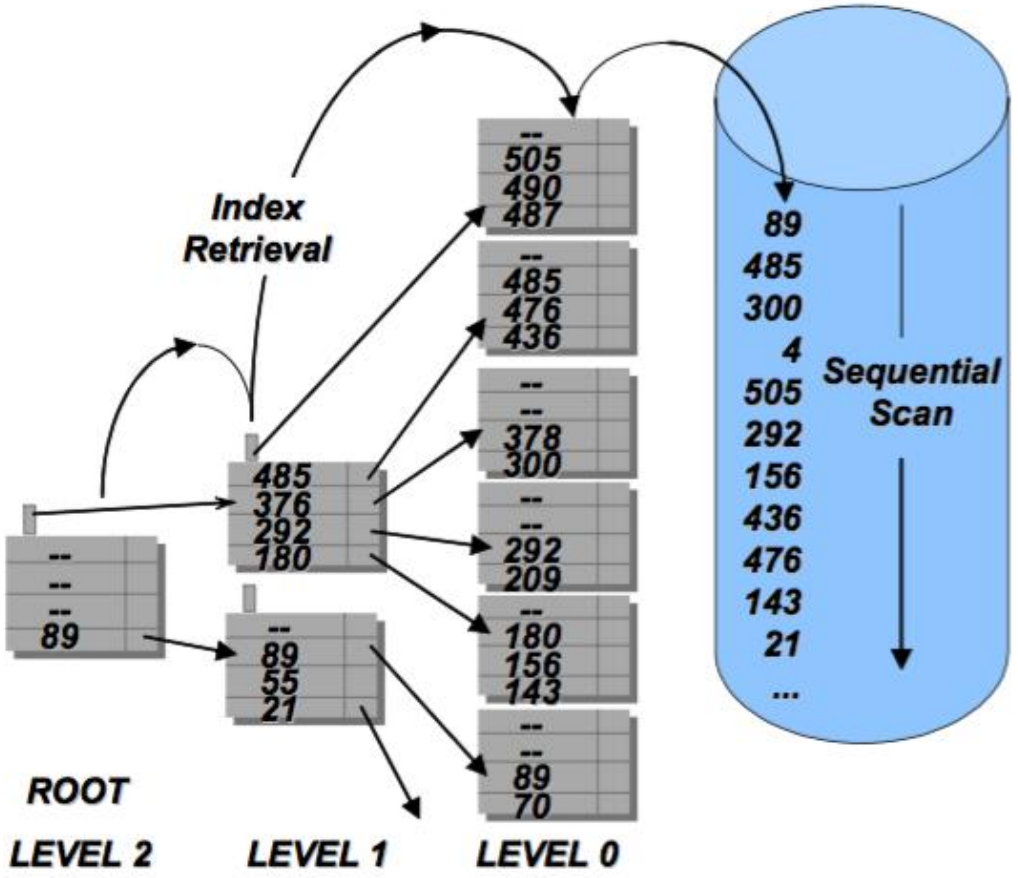
Example 4: query



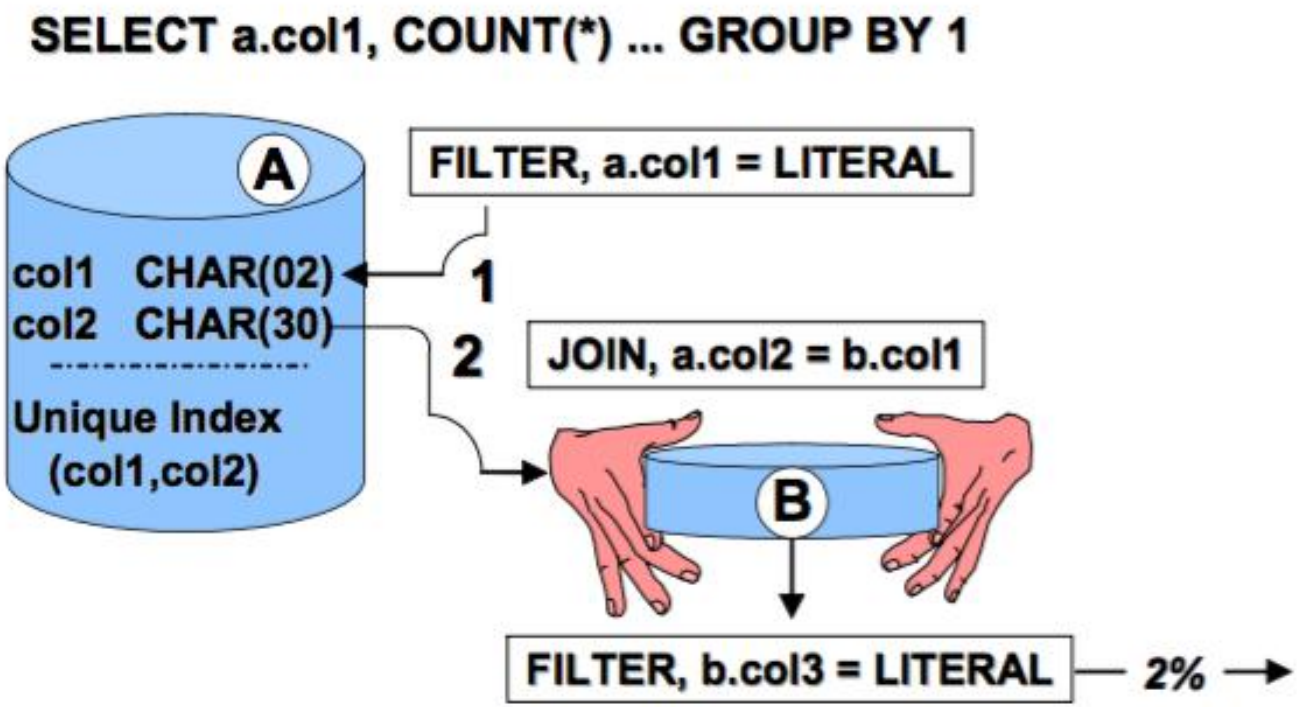
Query 4: First predicate



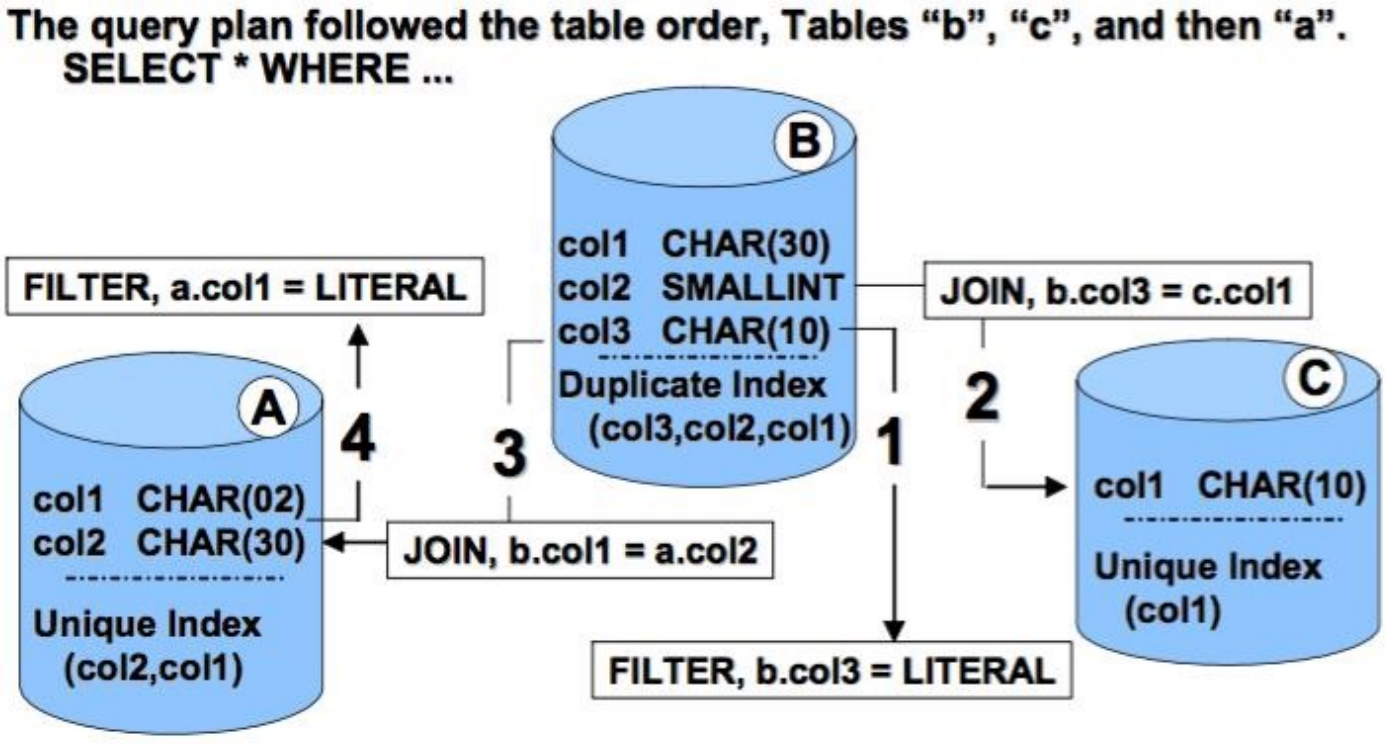
Collection access method: collection scan versus index scan



Query 4: Join and predicate



Query 4: Optimal plan



FJS versus ESR: MongoDB

```
SELECT *
FROM collection
WHERE
   col1 = ‘x’ and col2 > ‘y’
ORDER BY col3;
```

Filter -> Join -> Sort (FJS)

Equality -> Sort -> Range (ESR)

Problem statement:

I inherited a MongoDB database server with 60 collections and 100 or so indexes.

The business users are complaining about slow report completion times.

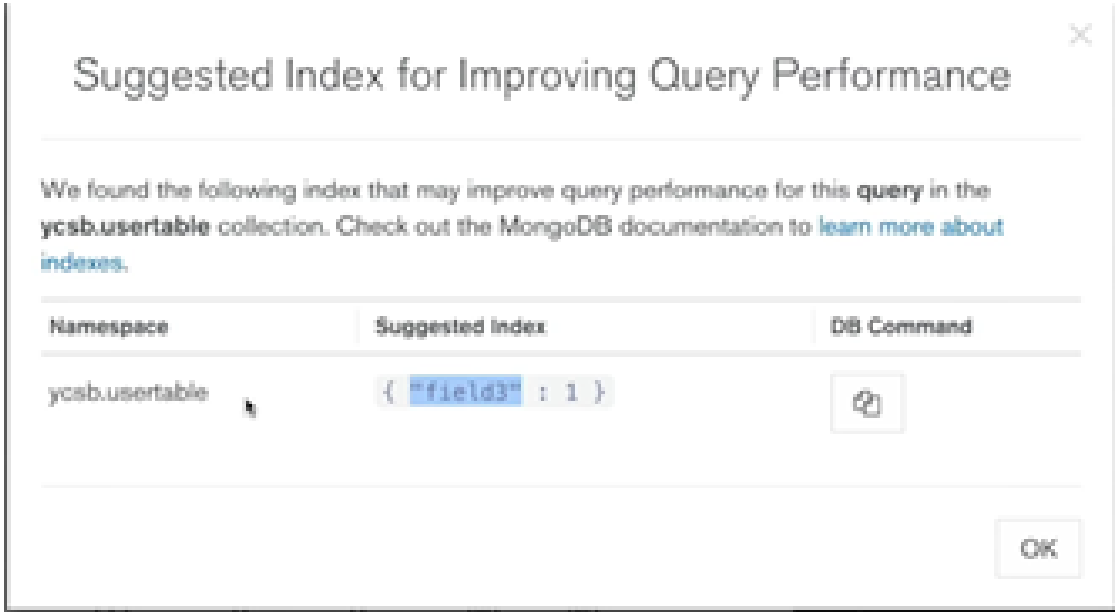
What can I do to improve performance ?

Skills/tools we need:

- Which server
- Which logfile
- Server profiling Level
- Which queries
  - Cloud/Ops Manager !!
  - mtools
  - Text processing

- Drop the query plan
- Analyze the query plan

- Which indexes get used
- Other



Sample data set: zips.json

```
> db.zips.findOne()
{
    "_id" : ObjectId("570 .. 1c1f2"),
    "city" : "ACMAR",
    "zip" : "35004",
    "loc" : {
        "y" : 33.584132,
        "x" : 86.51557
    },
    "pop" : 6055,
    "state" : "AL"
}
> db.zips.count()
29470
```

```
>db.zips.find( { "state" : "WI", "pop" :
{ "$lt" : 50 } } ).sort( { "city" : 1 } )
```

Query 5: Dumping the query plan

```
db.zips.find( { "state" : "WI",
    "pop" : { "$lt" : 50 } } ).sort(
    { "city" :
1 } ).explain("executionStats")
```

```
"winningPlan" : {
   "stage" : "SORT", "sortPattern" : { "city" : 1 },
   "inputStage" : {
   "stage" : "COLLSCAN",
     "filter" : {
       "$and" : [ "state" : { "$eq" : "WI" "pop" : { "$lt" : 50
"rejectedPlans" : [ ]
"executionStats" : {
   "nReturned" : 4,
   "executionTimeMillis" : 16,
   "totalKeysExamined" : 0,
   "totalDocsExamined" : 29470,
```

Query 5: get indexes

```
> db.zips.getIndexes()
[ {
      "v" : 1, "key" : { "_id" : 1 },
      "name" : "_id_",
      "ns" : "test_db.zips"
   } ]

db.zips.createIndex( { "state" : 1 , "pop" : 1 } )
```

Query 5: attempt 2

```
winningPlan" : {
   "stage" : "SORT", "sortPattern" : { "city" : 1 },
   "inputStage" : {
      "stage" : "FETCH",
   "inputStage" : {
      "stage" : "IXSCAN",
         "keyPattern" : { "state" : 1, "pop" : 1 },
         "indexBounds" : {
            "state" : [ "[\"WI\", \"WI\"]" ],
            "pop" : [ "[-inf.0, 50.0)" ]
"executionStats" : {
   "nReturned" : 4,
   "executionTimeMillis" : 1,
   "totalKeysExamined" : 4,
   "totalDocsExamined" : 4,
```

```
db.zips.createIndex( { "state" : 1 , "city" : 1 , "pop" : 1 } )

"winningPlan" : {
   "stage" : "SORT", "sortPattern" : { "city" : 1
   "inputStage" : {
      "stage" : "FETCH",
   "inputStage" : {
      "stage" : "IXSCAN",
         "keyPattern" : { "state" : 1, "pop" : 1
"rejectedPlans" : [
   "stage" : "FETCH",
   "inputStage" : {
      "stage" : "IXSCAN",
         "keyPattern" : { "state" : 1, "city" : 1, "pop" : 1 },
            "indexBounds" : {
                  "state" : [ "[\"WI\", \"WI\"]" ],
                  "city" : [ "[MinKey, MaxKey]" ],
                  "pop" : [ "[-inf.0, 50.0)" ]
```

Query 5: attempt 3

Query 6: (pseudo order by clause)

```
db.zips.find( { "state" : "CO" }
  ).sort( { "pop" : 1 } )
```

```
SELECT * FROM t1
WHERE col1 = 'x'
ORDER BY col2;

SELECT * FROM t1
WHERE col1 = 'x'
ORDER BY col1, col2;

SELECT * FROM t1
WHERE col1 = 'x'
ORDER BY 'x', col2;
```

Query 6: query plan

```
"winningPlan" : {
   "stage" : "FETCH",
   "inputStage" : {
      "stage" : "IXSCAN",
         "keyPattern" : { "state" : 1, "pop" : 1 },
         "indexBounds" : {
            "state" : [ "[\"CO\", \"CO\"]" ],
            "pop" : [ "[MinKey, MaxKey]" ]
"rejectedPlans" : [
   "stage" : "SORT",
      "sortPattern" : { "pop" : 1 },
   "inputStage" : {
```

```
"stage" : "FETCH",
   "inputStage" : {
      "stage" : "IXSCAN",
         "keyPattern" : { "state" : 1, "city" : 1, "pop" : 1 },
            "indexBounds" : {
               "state" : [ "[\"CO\", \"CO\"]" ],
               "city" : [ "[MinKey, MaxKey]" ],
               "pop" : [ "[MinKey, MaxKey]" ]
"executionStats" : {
   "nReturned" : 416,
   "executionTimeMillis" : 1,
   "totalKeysExamined" : 416,
   "totalDocsExamined" : 416,
```

Review the indexes we have so far

```
> db.zips.getIndexes()

_id
db.zips.createIndex( { "state" : 1 , "pop" : 1 } )
db.zips.createIndex( { "state" : 1 , "city" : 1 , "pop" : 1 } )
```

Query 7: OR topped query

```
db.zips.find( { "$or" : [ { "state" : "UT" }, { "pop" : 2 } ] } )

"winningPlan" : {
   "inputStage" : {
      "stage" : "COLLSCAN",
        "filter" : {
             "$or" : [
               "pop" : { "$eq" : 2 }
               "state" : { "$eq" : "UT" }
"rejectedPlans" : [ ]
"executionStats" : {
   "nReturned" : 215,
   "executionTimeMillis" : 22,
   "totalKeysExamined" : 0,
   "totalDocsExamined" : 29470,
```

```
SELECT * FROM t1
WHERE
   order_date = TODAY
OR
   ship_weight < 10;
```

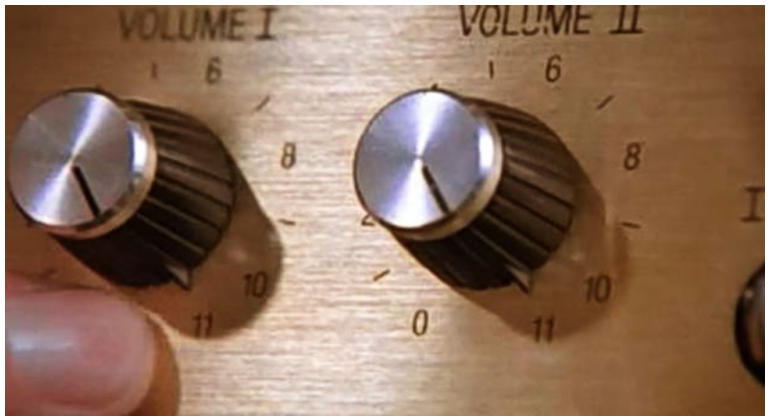
Query 7: solution

```
db.zips.createIndex( { "pop" : 1 } )

"winningPlan" : {
  "inputStage" : {
    "stage" : "FETCH",
  "inputStage" : {
    "stage" : "OR",
      "inputStages" : [
        "stage" : "IXSCAN",
          "keyPattern" : { "state" : 1, "pop" : 1 },
            "indexBounds" : {
              "state" : [ "[\"UT\", \"UT\"]" ],
              "pop" : [ "[MinKey, MaxKey]" ]
```

```
"stage" : "IXSCAN",
            "keyPattern" : { "pop" : 1 },
              "indexBounds" : {
                "pop" : [ "[2.0, 2.0]" ]
  "rejectedPlans" : [ ]
"executionStats" : {
  "nReturned" : 215,
  "executionTimeMillis" : 2,
  "totalKeysExamined" : 215,
  "totalDocsExamined" : 215,
```

Topics not previously covered



- How to tell which indexes are being used
- How to tell if an index is unique
- Smoke tests
- Covered queries
- MongoDB index types
- When do winning query plans get evacuated
- Index intersection
- Building indexes (online/offline)
- Sharding and queries, query plans
- Capped collections, tailable cursors
- Optimizer hints
- Memory limits
- Query rewrite (aggregation pipeline optimization)

- Which server
- Which logfile
- (Server profiling Level)
- Which queries
  - mtools
  - Text processing

Resources:

- The parent to this preso,
  https://github.com/farrell0/MongoDB-Developers-Notebook
- An excellent query primer (110 pages)
  http://www.redbooks.ibm.com/abstracts/sg247138.html?Open
  (Chapters 10 and 11.)
- University.MongoDB.com
- https://www.mongodb.com/consulting-test#performance_evaluation
- zips.json
  http://media.mongodb.org/zips.json
- Call Dave Lutz, at home, ... On Sunday (early)
  (512)555/1212

Backup Slides

How to tell which indexes are being used

```
db.zips.aggregate( [ { "$indexStats" : {} } ] ).pretty()
{ "name" : "pop_1",
  "key" : { "pop" : 1 },
  "host" : "rhhost00.grid:27017",
  "accesses" : {
    "ops" : NumberLong(15),
    "since" : ISODate("2016-04-19T07:13:44.546Z") } }
{ "name" : "state_1_city_1_pop_1",
  "key" : { "state" : 1, "city" : 1, "pop" : 1 },
  "host" : "rhhost00.grid:27017",
  "accesses" : {
    "ops" : NumberLong(0),
    "since" : ISODate("2016-04-19T06:49:11.765Z") } }
```

How to tell if an index is unique

```
db.t1.createIndex( { "k1" : 1 },
   { "unique" : true })
{
"createdCollectionAutomatically" : true,
"numIndexesBefore" : 1,
"numIndexesAfter" : 2,
"ok" : 1
}
```

```
> db.t1.getIndexes()
[ { "v" : 1,
   "key" : { "_id" : 1 },
   "name" : "_id_",
   "ns" : "test_db.t1“ },
{ "v" : 1,
   "unique" : true,
   "key" : { "k1" : 1 },
   "name" : "k1_1",
   "ns" : "test_db.t1“ }
]
```

Smoke tests

Every night, gather a set of statistics about your hard disk fullness, and about the performance of a set of queries that are strategic to the application.

For queries we wish to record-

- The number of documents returned
- The winning query plan
- Elapsed time, disk and memory consumed
- Other

```
> db.zips.find( { "pop" : { "$lt" : 200 } },
  { "_id" : 0, "pop" : 1 } ).sort(
  { "pop" : -1 } ).explain()
"winningPlan" : {
  "stage" : "PROJECTION",
  "transformBy" : {
    "_id" : 0,
    "pop" : 1 },
  "inputStage" : {
    "stage" : "IXSCAN",
      "keyPattern" : { "pop" : 1 },
      "indexBounds" : {
        "pop" : [ "(200.0, -inf.0]" ]
  "rejectedPlans" : [ ]
}
```

Covered queries

When does the winning plan get evacuated

In short, the cached query plan is re-evaluated if:

- The collection receives 1000 or more writes
- An index is added or dropped
- A reindex operation is performed
- mongod is restarted
- You run a query with explain

Index intersection

```
db.zips.find( { "$or" : [ { "state" : "UT" }, { "pop" : 2 } ] } )
```

```
db.zips.find( { "city" : "EAST TROY", "zip" : 53120 } )
```

Building indexes

```
db.zips.createIndex( { “zip” : 1 }, { “background” : true } )
```

Capped collections

```
db.createCollection("my_collection",
{ capped : true, size : 5242880,
max : 5000 } )
```

```
from pymongo import Connection
import time

db = Connection().my_db
coll = db.my_collection
cursor = coll.find(tailable=True)
while cursor.alive:
    try:
        doc = cursor.next()
        print doc
    except StopIteration:
        time.sleep(1)
```

mongoDB

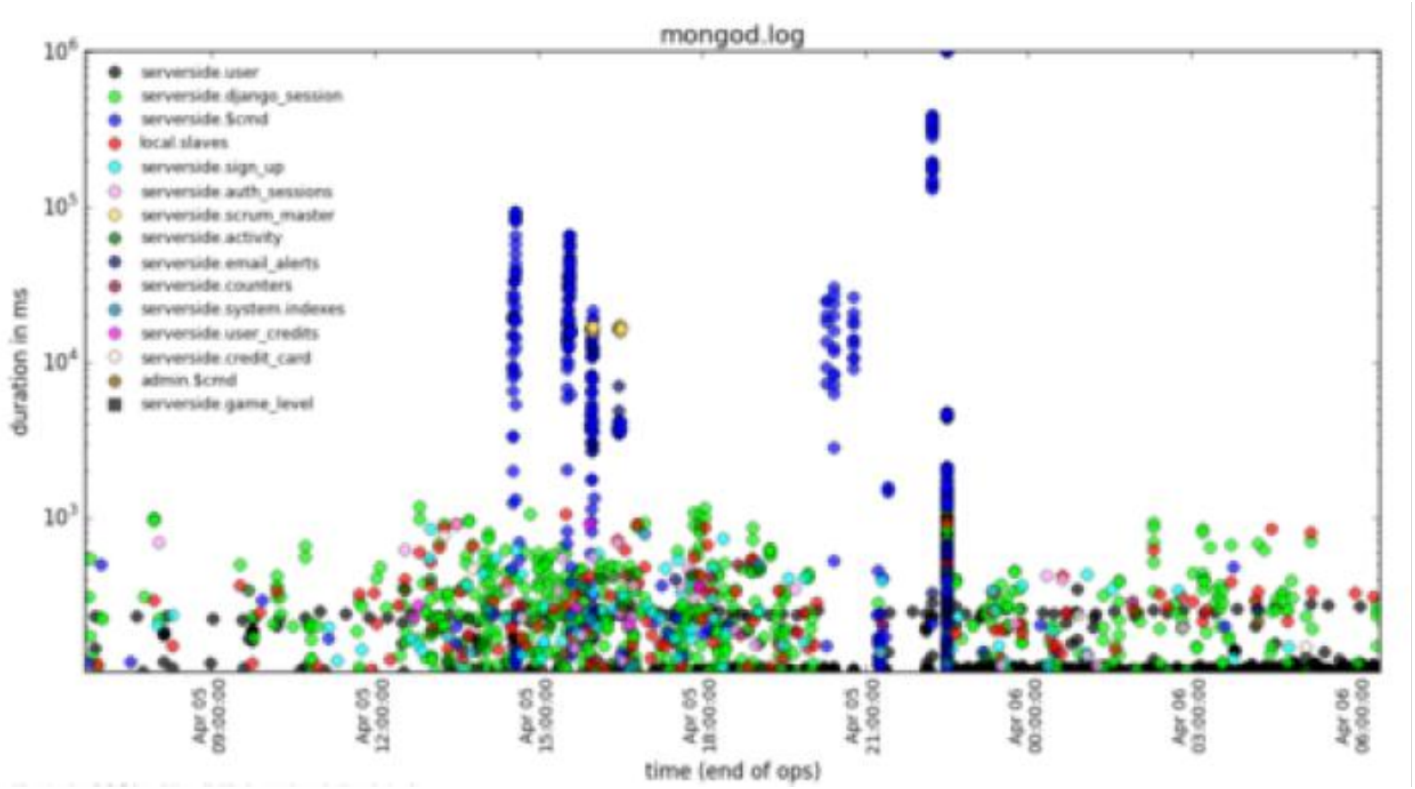
Memory limits: 32 MB, 100 MB

```
"executionStages" : {
   "stage" : "SORT",
   "nReturned" : 1,
   "executionTimeMillisEstimate" : 60,
      …
   "sortPattern" : { "City" : 1 },
     "memUsage" : 120,
     "memLimit" : 33554432,
```

```
db.zips.aggregate([
   { "$group" :
      { "_id"       : "$state",
        //
      "totalPop"  : { "$sum" : "$pop" },
      "cityCount" : { "$sum" : 1 }
      }
   } ,
   { "$sort" : { "_id" : 1 } }
   ],
   { "allowDiskUse" : true }
)
```

mongoDB

mtools: mplotquery



But first:

Y/N I have more than 24 months experience with SQL

Y/N I have more than 6 months experience with MongoDB

Y/N I have dropped a MongoDB explain plan, understood it, made changes, and was happy

Y/N Puppies scare me

Two more examples: Queries 8 and 9

find()  **aggregate()**

- optimizer hints
- $lookup()

Query 8: automatic query rewrite

```
> db.zips.aggregate(
...    [
...       { "$sort" :
...          { "state" : 1 }
...       },
...       {
...       "$match" :
...          { "state" : { "$gt" : "M" } }
...       }
...    ],
...    { "explain" : true } )
```

Query 8: Explain plan

```
"stages" : [
  "$cursor" : {
    "sort" : { "state" : 1 },
"winningPlan" : {
  "stage" : "FETCH",
    "inputStage" : {
      "stage" : "IXSCAN",
        "indexName" : "state_1_city_1",
          "indexBounds" : {
            "state" : [ "(\"M\", {})" ],
            "city" : [ "[MinKey, MaxKey]" ]
"rejectedPlans" : [ ]
```

Query 9: Optimizer hints

```
> db.zips.find( { "city" : "EAST TROY" }).hint(
   { "zip" : 1} ).explain("executionStats")

      "winningPlan" : {
        "stage" : "FETCH",
        "filter" : { "city" : { "$eq" : "EAST TROY" } },
        "inputStage" : {
          "stage" : "IXSCAN", "keyPattern" : { "zip" : 1 },
          "indexBounds" : { "zip" : [ "[MinKey, MaxKey]" ] }
      "rejectedPlans" : [ ]
    "executionStats" : {
      "nReturned" : 1,
      "executionTimeMillis" : 28,
      "totalKeysExamined" : 29470,
      "totalDocsExamined" : 29470,
```