QUERYING

Query

- Return all

```
db.users.find()
```

- Filter results

```
db.users
  .find({“dept” : “sales”, “location” : “nyc”})
```

- Project keys (_id always returned)

```
db.users
  .find({“dept” : “sales”, “location” : “nyc”},
        {“name” : 1, “email” : 1})
```

Comparison

- Operators:
  - “$lt”
  - “$lte”
  - “$gt”
  - “$gte”
  - “$ne”
- Example

```
db.users.find({“age” : {“$gte” : 18, “$lte” : 30}})
start = new Date(“01/01/2007”)
db.users.find({“registered” : {“$lt” : start}})
```

60

60

Comparison

- Set membership:

```
db.raffle.find(
   {“ticket_no” :
        {“$in” : [725, 542, 390]}})
```

- Disjunction:

```
db.raffle.find(
   {“$or” :
      [ {“ticket_no” :
             {“$in” : [725, 542, 390]}},
        {“winner” : true}
      ]
   })
```

61

61

Type-Specific Queries

- Regular Expressions

```
    db.users.find({“name” : /joey?/i})
```

- Null
  - Matches “does not exist”:

```
    db.coll.find({“x” : null})
```

  - Find keys whose value is null:

```
    db.coll.find({“x” : {“$in” : [null],
                         “$exists” : true}})
```

Querying Arrays

- Array: any element can match search key
- Insertion:

```
db.food.insert(
    {“fruit” : [“apple”, ”banana”, ”peach”]})
```

- Query:

```
db.food.find({“fruit” : ”banana”})

db.food.find({“fruit” :
                {“$all” : [“apple”, ”banana”]} })

db.food.find({“fruit” : {“$size” : 3}})
```

Array Slicing

- First 10 comments:

```
db.blog.posts.findOne(criteria,
       {"comments" : {"$slice" : 10}})
```

- Last 10 comments:

```
db.blog.posts.findOne(criteria,
       {"comments" : {"$slice" : -10}})
```

- Range of comments:

```
db.blog.posts.findOne(criteria,
       {"comments" : {"$slice" : [23, 10]}})
```

offset   # elements

Querying for Embedded Keys

- Issue: embedded doc match must match the whole doc
- Example database:

```
{
  "name" : {
       "first" : "Joe",
        "last" : "Schmoe"
  },
  "age" : 45
}
```

- Query:

~~db.people.find({"name" :~~
~~{"first" : "Joe", "last" : "Schmoe"}})~~

```
db.people.find(
         {"name.first" : "Joe", "name.last" : "Schmoe"})
```

Embedded Document Matches

- Issue: author and score match should be for same list elem
- Example database:

```
{
  "content" : "...",
  "comments" :
      [ ... {“author” : “joe”, “score” : 3, ...} ...]
}
```

- Query:

~~db.blog.find({"comments" :~~
~~{"author" : "joe", "score" : {"$gte" : 5}}})~~
~~db.blog.find({"comments.author" : "joe",~~
~~"comments.score" : {"$gte" : 5}}),~~

```
db.people.find({“comments” :
             {“$elemMatch” :
             {”author" : "Joe", ”score" : {”$gte” : 5}}}})
```

66

$where Queries

- Example database:

```
db.foo.insert({"apple" : 1, "banana" : 6, "peach" : 3})
db.foo.insert({"apple" : 8, "spinach" : 4, ”banana" : 4})
```

- Query:

```
db.foo.find({“$where” : function () {
  for (var current in this) {
    for (var other in this) {
      if (current != other &&
          this[current] == this[other]) {
        return true;
      }
    }
  }
  return false;
}});
```

67

Cursors

- Assign result of database query:

```
var cursor = db.foo.find()
while (cursor.hasNext()) {
  obj = cursor.next();
  // do something
}
```

- Iterator interface:

```
Var cursor = db.people.find();
cursor.forEach(function(x) {
  print(x.name);
});
```

68

Cursor Options

- Options: `limit(), skip(), sort()`
- Add options using builder pattern

```
var cursor =
  db.people.find().sort({"x" : 1}).limit(1).skip(10);

var cursor =
  db.people.find().limit(1).sort({"x" : 1}).skip(10);

var cursor =
  db.people.find().skip(10).limit(1).sort({"x" : 1});
```

- Execute query:

```
cursor.forEach(function(x) {
  print(x.name);
});
```

69

Paginating without skip

- Avoid long skips - expensive

```
var page1 = db.foo.find(criteria).limit(100)
var page2 = db.foo.find(criteria).skip(100).limit(100)
var page3 = db.foo.find(criteria).skip(200).limit(100)
```

- Alternative: Keep track of current position via key

```
var page1 = db.foo.find().sort({"date" : -1}).limit(100)
var latest = null; // display first page
while (page1.hasNext()) {
  latest = page1.next();
  display(latest);
}
// get next page
var page2 =
  db.foo.find({"date" : {"$gt" : latest.date}});
page2.sort({"date" : -1}).limit(100);
```

70

Wrapped Queries

- Plain query

```
var cursor = db.foo.find({"foo" : "bar"})
```

- Wrapping

```
var cursor = db.foo.find({"foo" : "bar"}).sort({“x” : 1})
```

- Other options

```
$maxscan : integer
$min : document
$max : document
$hint : document
$explain : boolean
$snapshot : boolean
```

71

$snapshot for Consistent Result

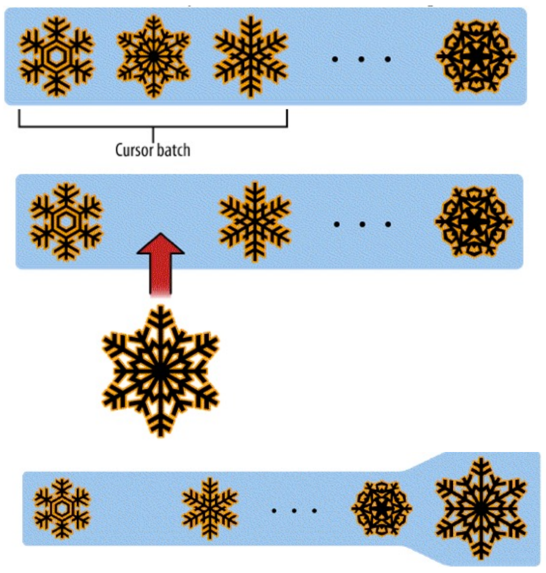
- Typical scenario:

```
cursor = db.foo.find();
while (cursor.hasNext()) {
  var doc = cursor.next();
  doc = process(doc);
  db.foo.save(doc);
}
```

72

72

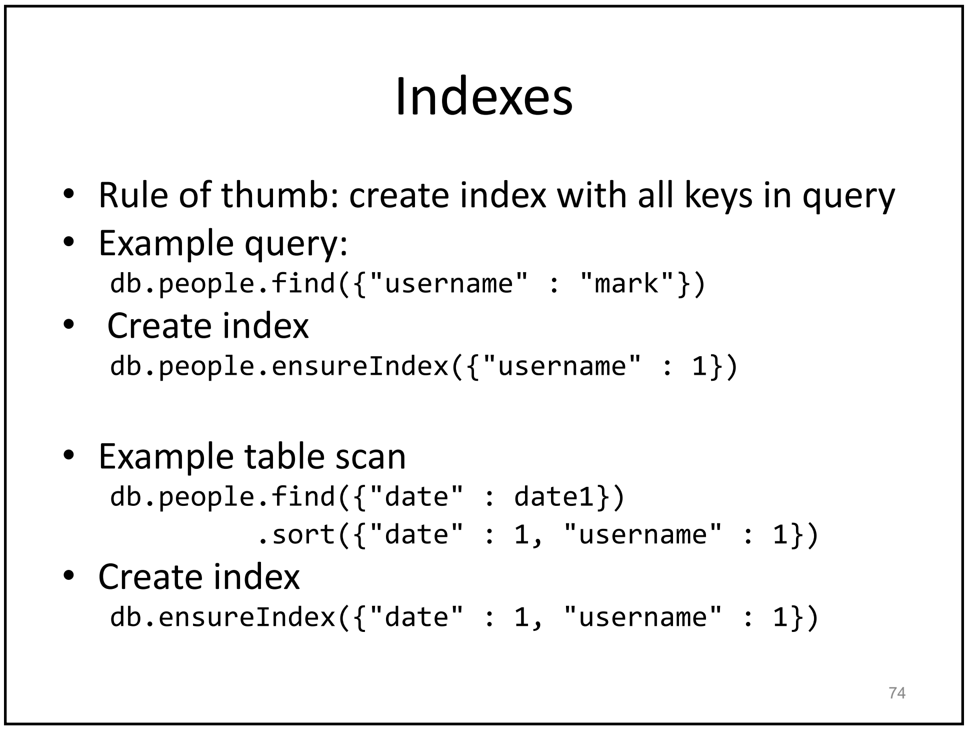
$snapshot for Consistent Result



```
cursor =
  db.foo
    .find()
    .snapshot();
```

73

73



Indexes

- Rule of thumb: create index with all keys in query
- Example query:
```
db.people.find({"username" : "mark"})
```
- Create index
```
db.people.ensureIndex({"username" : 1})
```

- Example table scan
```
db.people.find({"date" : date1})
         .sort({"date" : 1, "username" : 1})
```
- Create index
```
db.ensureIndex({"date" : 1, "username" : 1})
```

74

74