MongoDB

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

People matter, results count.

Introduction

- MongoDB is one of the most popular and fastest growing open source NoSQL database.
- MongoDB is written in C++. It is fast and scalable.
- Most of the MongoDB functionalities can be accessed directly through JavaScript notation and we can make use of all of the standard JavaScript libraries with in it.
- It uses JSON for storing and manipulating the data. A JSON database returns query results that can be easily parsed, with little or no transformation, directly by JavaScript and most popular programming languages which reduces the amount of logic need to build into the application layer.
- MongoDB represents JSON documents in binary-encoded format called BSON(Binary JSON) behind the scenes. BSON extends the JSON model to provide additional data types and to be efficient for encoding and decoding within different languages.

Why MongoDB?

- MongoDB is a cross-platform, document oriented database that provides, high performance, high availability and easy scalability
- It supports a wide range of Operating Systems (Windows, OSX, Linux)
- There are drivers for nearly any language including C/C++, Python, PHP, Ruby, Perl, .NET and Node.js.
- Document Oriented Storage i.e. Data is stored in the form of JSON style documents
- Index on any attribute
- Replication & High Availability
- Auto-Sharding
- Rich Queries
- Migrations and a constantly evolving schemas can be managed easier

MongoDB key terminologies

- Database
  - Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases
- Collection
  - A *collection* may be considered as a table except there are no aligned columns.
- Document
  - Each of the entries or rows inside a collection is called a *document*. Each entry (row) can use varying dynamic schemas in key-value pairs.
  - Collections have dynamic schemas. This means that the documents within a single collection can have any number of different “shapes.”
  - Inside a collection of Users there may be one entry with First name & Last name. Then another entry with First, Last, and Middle name, along with e-mail address and date of birth.
  - Documents are basically JSON data blocks stored in memory-mapped files which behave as separate entries in collections.

SQL Terminology vs MongoDB Terminology

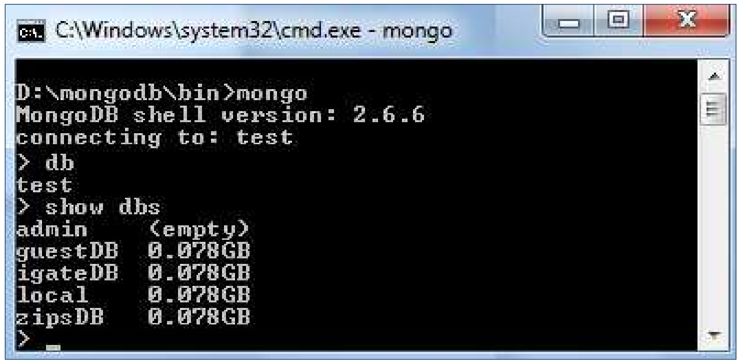
SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document or BSON document
column	field
index	index
table joins	embedded documents and linking
primary key	primary key In MongoDB primary key is automatically set to the_id field.
aggregation (e.g. group by)	aggregation pipeline.

MongoDB vs Relational Databases

- Relational databases save data in tables and rows.
- But in application development our objects are not simply tables and rows. It forces an application developer to write a mapping layer or use an ORM, to translate the object in memory and what is saved in the database. Mapping those to tables and rows can be quite a bit of pain.
- In MongoDB, there is no schema to define. There are no tables and no relationships between collections of objects.
- Every document you save in Mongo can be as flat and simple, or as complex as your application requires. This makes developer life much easier and your application code much cleaner and simpler.
- Further, two documents in the same collection may be different from each other since there is no schema governing the collection.
- Structuring a single object is clear and no complex joins

Starting the mongo shell

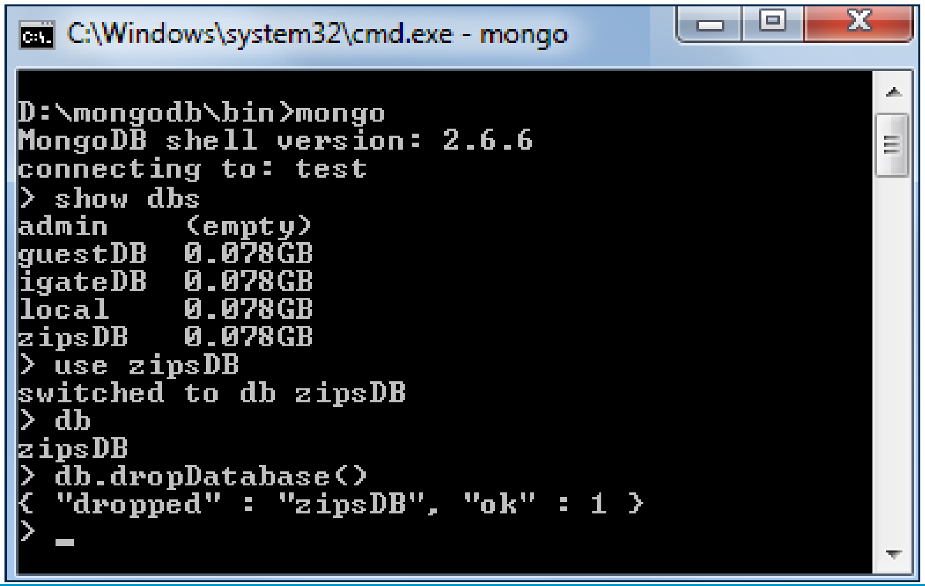
- To start the mongo shell and connect to your MongoDB instance running on localhost(127.0.0.1) with default port(27017)
  - **Go to <mongodb installation dir> :** *cd <mongodb installation dir>*
  - **To start mongo goto bin directory and type mongo ex:-** *D:\mongodb\bin>mongo*
- To check your currently selected database, use the command *db*. Default(test)
  - *db*
- To list the databases, use the command show dbs.
  - *show dbs*



```
D:\mongodb\bin>mongo
MongoDB shell version: 2.6.6
connecting to: test
> db
test
> show dbs
admin     (empty)
guestDB   0.078GB
igateDB   0.078GB
local     0.078GB
zipsDB    0.078GB
> _
```

Creating and Dropping Database

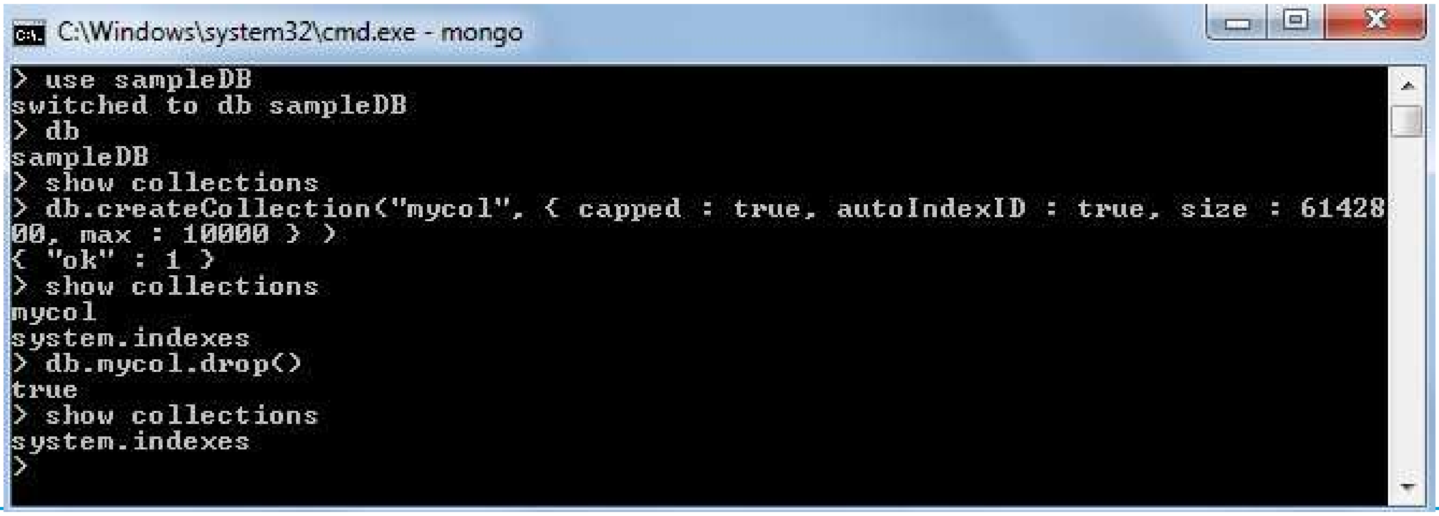
- MongoDB *use DATABASE_NAME* is used to create database.
- If the database doesn't exists it creates a new database, otherwise it will return the existing database which can be used using *db*.
  - *use DATABASE_NAME*
- To display database you need to insert at least one document into it.
- db.dropDatabase() command is used to drop a existing database.



```
D:\mongodb\bin>mongo
MongoDB shell version: 2.6.6
connecting to: test
> show dbs
admin     (empty)
guestDB   0.078GB
igateDB   0.078GB
local     0.078GB
zipsDB    0.078GB
> use zipsDB
switched to db zipsDB
> db
zipsDB
> db.dropDatabase()
{ "dropped" : "zipsDB", "ok" : 1 }
> _
```

Creating and Dropping Collection

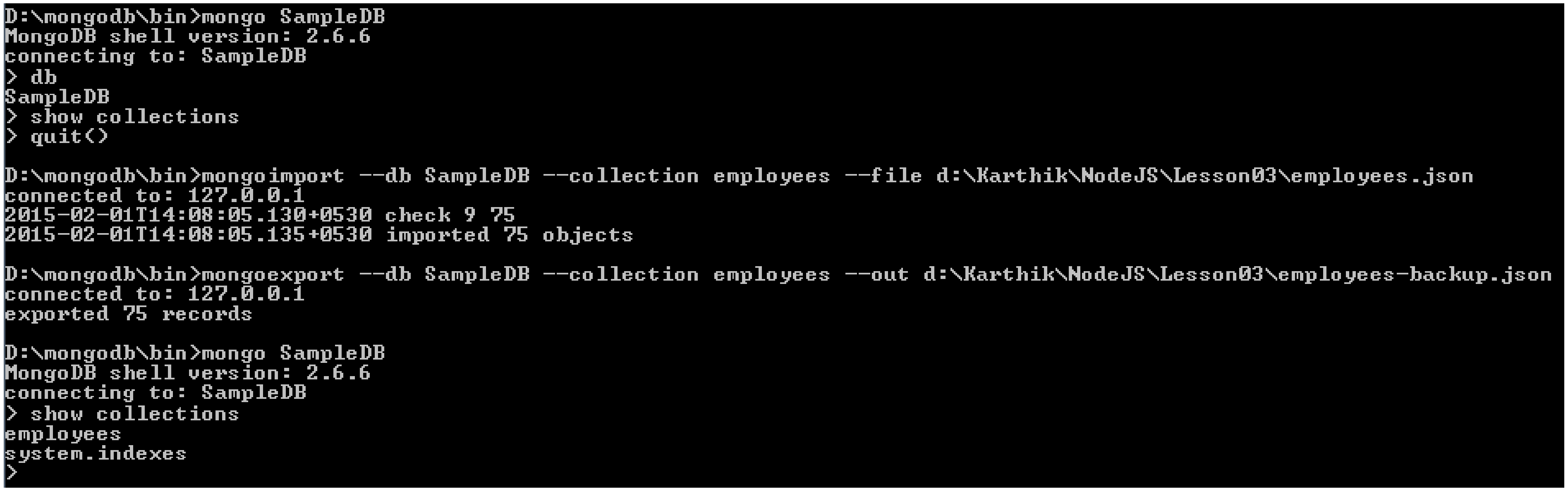
- MongoDB *db.createCollection(collectionname, options)* is used to create collection. Options parameter is optional. Following is the list of options
  - **capped** : Capped collection is a collection fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size.
  - **autoIndexID**: If true, If true, automatically create index on _id field.
  - **size**: Specifies a maximum size in bytes for a capped collection.
  - **max**: Specifies the maximum number of documents allowed in the capped collection.
- db.<collectionname>.drop() command is used to drop a collection



```
> use sampleDB
switched to db sampleDB
> db
sampleDB
> show collections
> db.createCollection("mycol", { capped : true, autoIndexID : true, size : 61428
00, max : 10000 } )
{ "ok" : 1 }
> show collections
mycol
system.indexes
> db.mycol.drop()
true
> show collections
system.indexes
>
```

Importing and Exporting Collection

- To import the collection from the file use *mongoimport* command
  - *mongoimport --db <DBName> --collection <CollectionName> --file <FileName>*
- To export the collection from the DB use *mongoexport* command
  - *mongoexport --db <DBName> --collection <CollectionName> --out <OutputFileName>*



```
D:\mongodb\bin>mongo SampleDB
MongoDB shell version: 2.6.6
connecting to: SampleDB
> db
SampleDB
> show collections
> quit()

D:\mongodb\bin>mongoimport --db SampleDB --collection employees --file d:\Karthik\NodeJS\Lesson03\employees.json
connected to: 127.0.0.1
2015-02-01T14:08:05.130+0530 check 9 75
2015-02-01T14:08:05.135+0530 imported 75 objects

D:\mongodb\bin>mongoexport --db SampleDB --collection employees --out d:\Karthik\NodeJS\Lesson03\employees-backup.json
connected to: 127.0.0.1
exported 75 records

D:\mongodb\bin>mongo SampleDB
MongoDB shell version: 2.6.6
connecting to: SampleDB
> show collections
employees
system.indexes
>
```

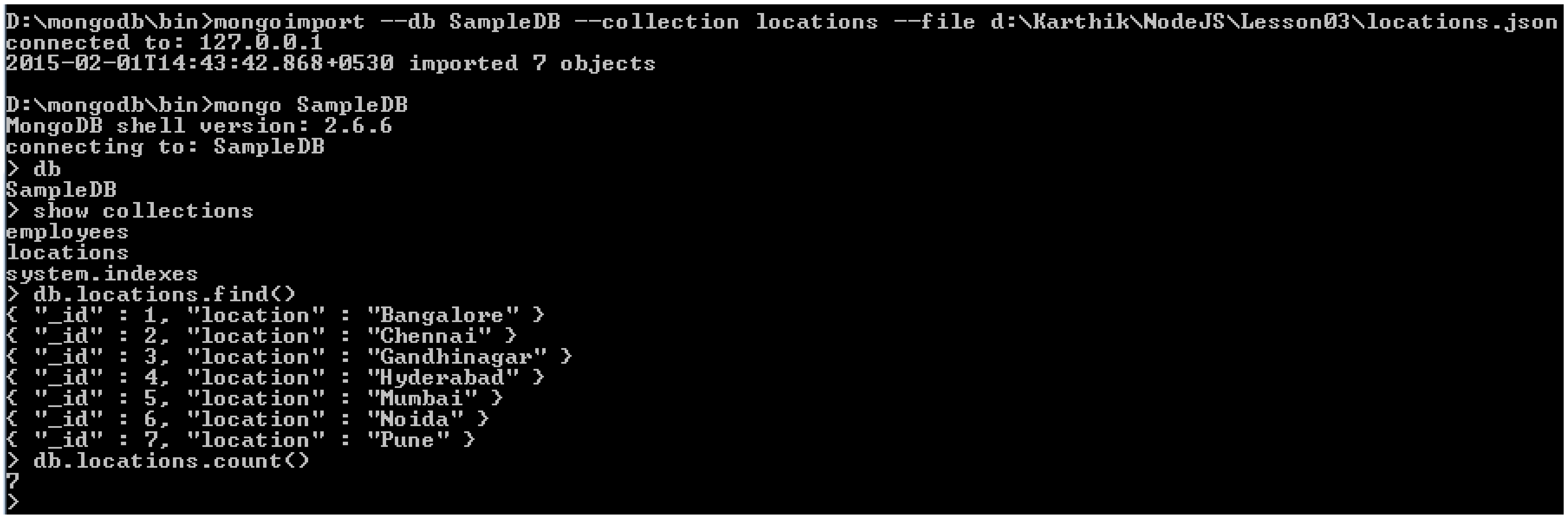
Demo

- Importing and Exporting Collections

Querying MongoDB Documents – find()

➢MongoDB's find() method is used to query data from MongoDB collection

- ***db.COLLECTION_NAME.find()***
- **db. *COLLECTION_NAME*.find().pretty()** is used to display the results in a formatted way.
- **db. *COLLECTION_NAME*.count()** is used to count the number of documents in a collection.

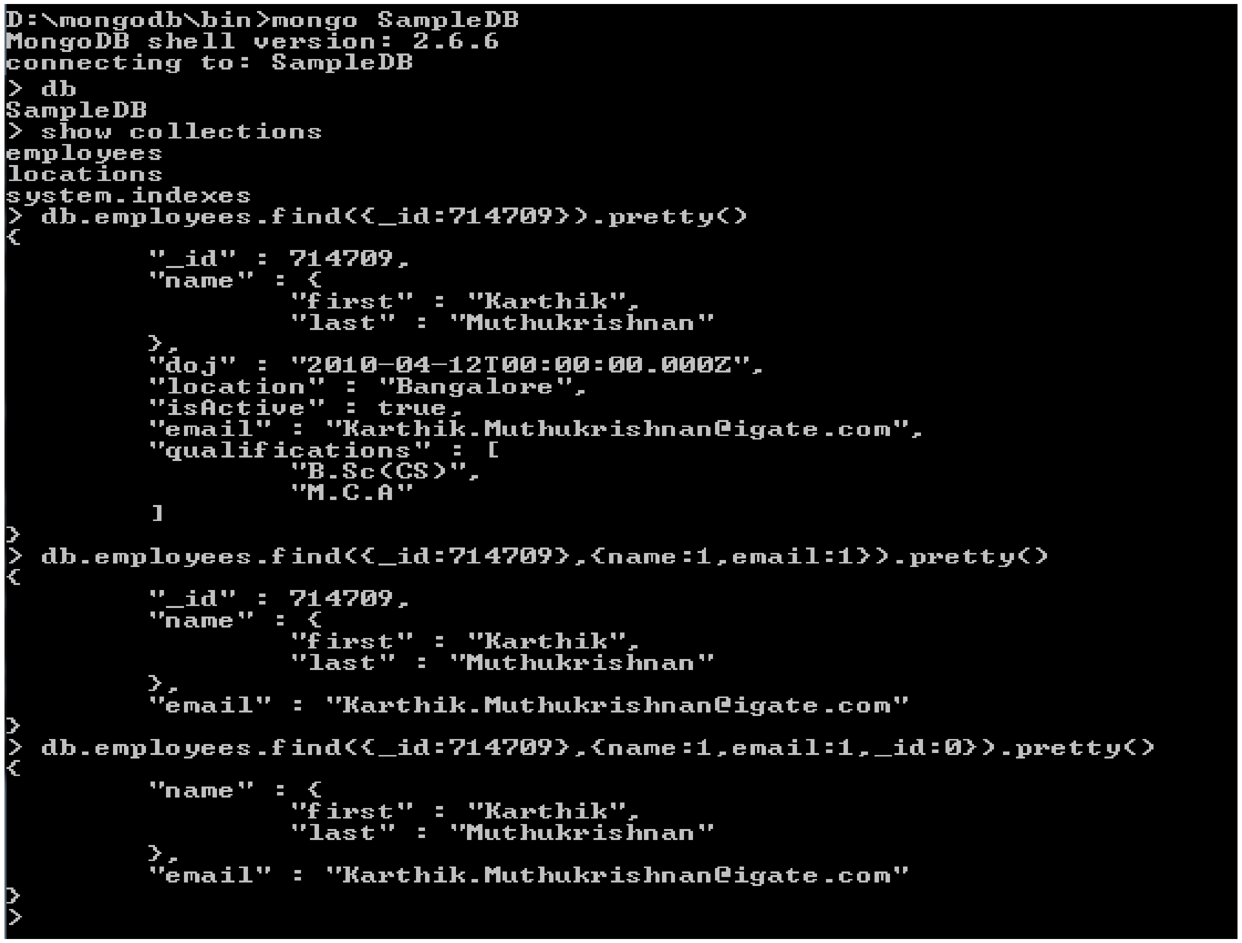


```
D:\mongodb\bin>mongoimport --db SampleDB --collection locations --file d:\Karthik\NodeJS\Lesson03\locations.json
connected to: 127.0.0.1
2015-02-01T14:43:42.868+0530 imported 7 objects

D:\mongodb\bin>mongo SampleDB
MongoDB shell version: 2.6.6
connecting to: SampleDB
> db
SampleDB
> show collections
employees
locations
system.indexes
> db.locations.find()
{ "_id" : 1, "location" : "Bangalore" }
{ "_id" : 2, "location" : "Chennai" }
{ "_id" : 3, "location" : "Gandhinagar" }
{ "_id" : 4, "location" : "Hyderabad" }
{ "_id" : 5, "location" : "Mumbai" }
{ "_id" : 6, "location" : "Noida" }
{ "_id" : 7, "location" : "Pune" }
> db.locations.count()
7
>
```

Querying MongoDB Documents – pretty()

- To print the results in formatted way use pretty()

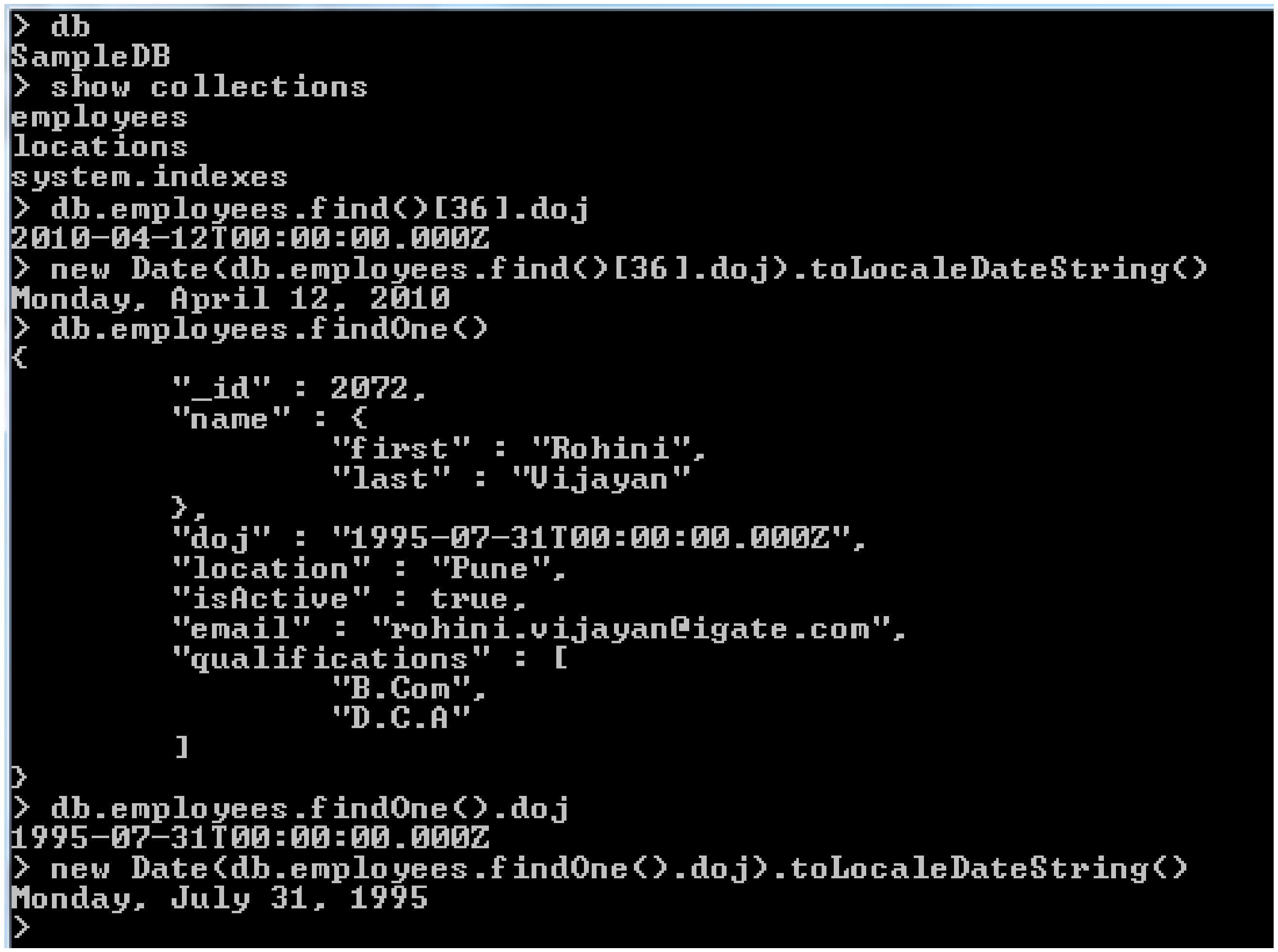


```
D:\mongodb\bin>mongo SampleDB
MongoDB shell version: 2.6.6
connecting to: SampleDB
> db
SampleDB
> show collections
employees
locations
system.indexes
> db.employees.find({_id:714709}).pretty()
{
        "_id" : 714709,
        "name" : {
                "first" : "Karthik",
                "last" : "Muthukrishnan"
        },
        "doj" : "2010-04-12T00:00:00.000Z",
        "location" : "Bangalore",
        "isActive" : true,
        "email" : "Karthik.Muthukrishnan@igate.com",
        "qualifications" : [
                "B.Sc(CS)",
                "M.C.A"
        ]
}
> db.employees.find({_id:714709},{name:1,email:1}).pretty()
{
        "_id" : 714709,
        "name" : {
                "first" : "Karthik",
                "last" : "Muthukrishnan"
        },
        "email" : "Karthik.Muthukrishnan@igate.com"
}
> db.employees.find({_id:714709},{name:1,email:1,_id:0}).pretty()
{
        "name" : {
                "first" : "Karthik",
                "last" : "Muthukrishnan"
        },
        "email" : "Karthik.Muthukrishnan@igate.com"
}
>
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Querying MongoDB Documents – findOne()

- findOne() returns a single document, where as find() returns a cursor.

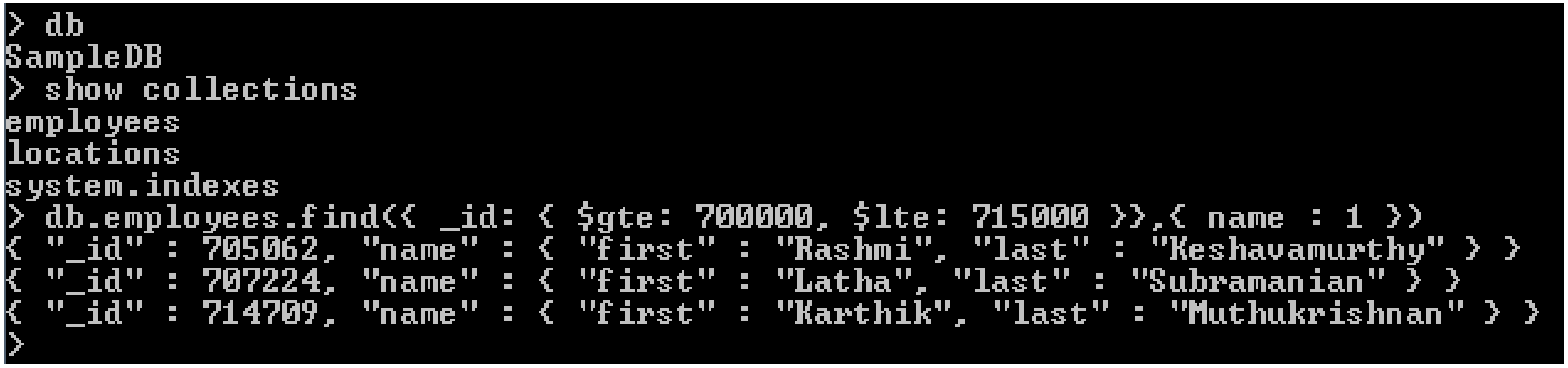


```
> db
SampleDB
> show collections
employees
locations
system.indexes
> db.employees.find()[36].doj
2010-04-12T00:00:00.000Z
> new Date(db.employees.find()[36].doj).toLocaleDateString()
Monday, April 12, 2010
> db.employees.findOne()
{
        "_id" : 2072,
        "name" : {
                "first" : "Rohini",
                "last" : "Vijayan"
        },
        "doj" : "1995-07-31T00:00:00.000Z",
        "location" : "Pune",
        "isActive" : true,
        "email" : "rohini.vijayan@igate.com",
        "qualifications" : [
                "B.Com",
                "D.C.A"
        ]
}
> db.employees.findOne().doj
1995-07-31T00:00:00.000Z
> new Date(db.employees.findOne().doj).toLocaleDateString()
Monday, July 31, 1995
>
```

Comparison Operators

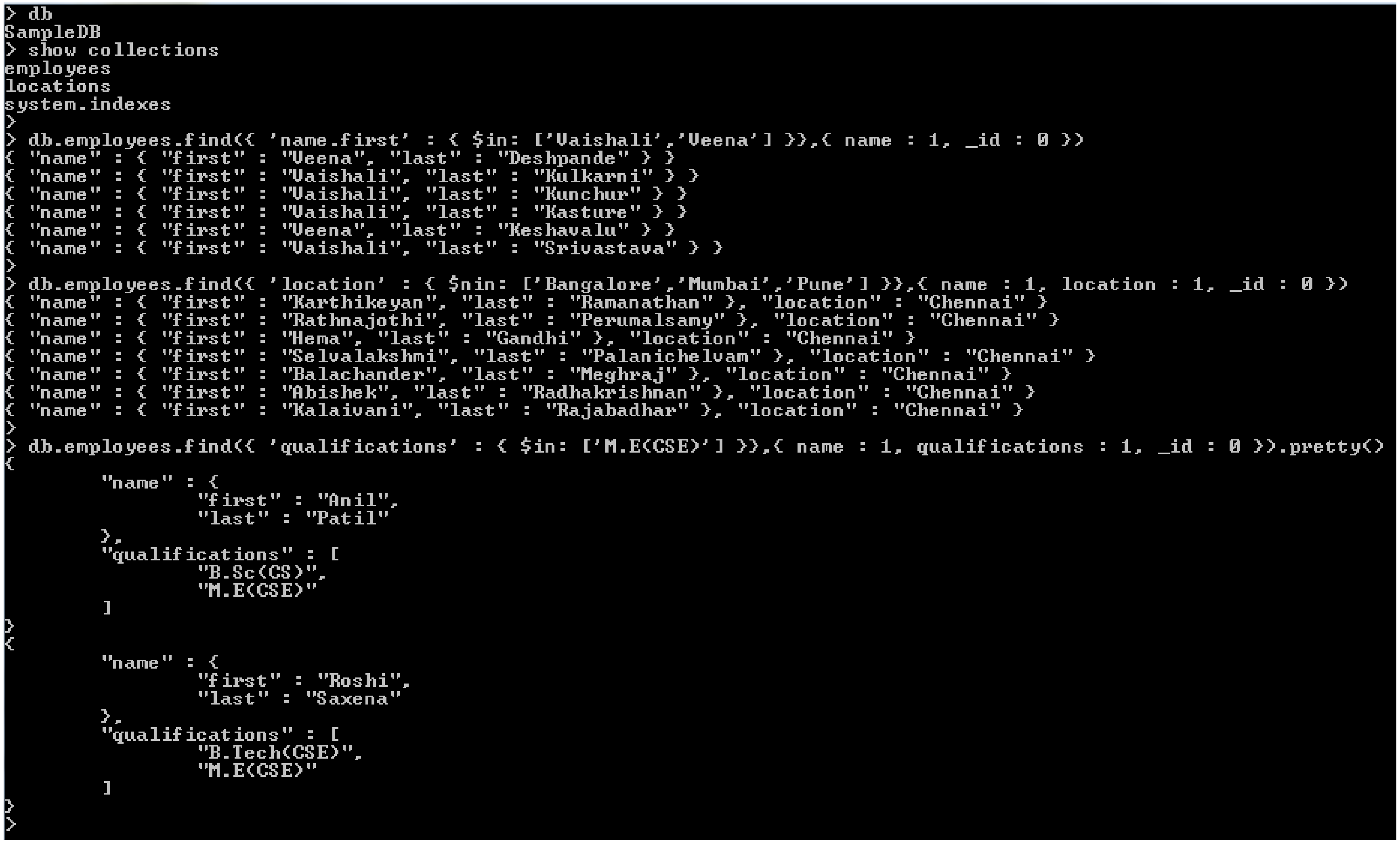
Name	Description
$gt	Matches values that are greater than the value specified in the query.
$gte	Matches values that are greater than or equal to the value specified in the query.
$in	Matches any of the values that exist in an array specified in the query.
$lt	Matches values that are less than the value specified in the query.
$le	Matches values that are less than or equal to the value specified in the query.
$ne	Matches all values that are not equal to the value specified in the query.
$nin	Matches values that do not exist in an array specified to the query.

Comparison Operators – gte & lte



```
> db
SampleDB
> show collections
employees
locations
system.indexes
> db.employees.find({ _id: { $gte: 700000, $lte: 715000 }},{ name : 1 })
{ "_id" : 705062, "name" : { "first" : "Rashmi", "last" : "Keshavamurthy" } }
{ "_id" : 707224, "name" : { "first" : "Latha", "last" : "Subramanian" } }
{ "_id" : 714709, "name" : { "first" : "Karthik", "last" : "Muthukrishnan" } }
>
```

MongoDB Comparison Operators – in & nin



```
> db
SampleDB
> show collections
employees
locations
system.indexes
>
> db.employees.find({ 'name.first' : { $in: ['Vaishali','Veena'] }},{ name : 1, _id : 0 })
{ "name" : { "first" : "Veena", "last" : "Deshpande" } }
{ "name" : { "first" : "Vaishali", "last" : "Kulkarni" } }
{ "name" : { "first" : "Vaishali", "last" : "Kunchur" } }
{ "name" : { "first" : "Vaishali", "last" : "Kasture" } }
{ "name" : { "first" : "Veena", "last" : "Keshavalu" } }
{ "name" : { "first" : "Vaishali", "last" : "Srivastava" } }
>
> db.employees.find({ 'location' : { $nin: ['Bangalore','Mumbai','Pune'] }},{ name : 1, location : 1, _id : 0 })
{ "name" : { "first" : "Karthikeyan", "last" : "Ramanathan" }, "location" : "Chennai" }
{ "name" : { "first" : "Rathnajothi", "last" : "Perumalsamy" }, "location" : "Chennai" }
{ "name" : { "first" : "Hema", "last" : "Gandhi" }, "location" : "Chennai" }
{ "name" : { "first" : "Selvalakshmi", "last" : "Palanichelvam" }, "location" : "Chennai" }
{ "name" : { "first" : "Balachander", "last" : "Meghraj" }, "location" : "Chennai" }
{ "name" : { "first" : "Abishek", "last" : "Radhakrishnan" }, "location" : "Chennai" }
{ "name" : { "first" : "Kalaivani", "last" : "Rajabadhar" }, "location" : "Chennai" }
>
> db.employees.find({ 'qualifications' : { $in: ['M.E(CSE)'] }},{ name : 1, qualifications : 1, _id : 0 }).pretty()
{
        "name" : {
                "first" : "Anil",
                "last" : "Patil"
        },
        "qualifications" : [
                "B.Sc(CS)",
                "M.E(CSE)"
        ]
}
{
        "name" : {
                "first" : "Roshi",
                "last" : "Saxena"
        },
        "qualifications" : [
                "B.Tech(CSE)",
                "M.E(CSE)"
        ]
}
>
```

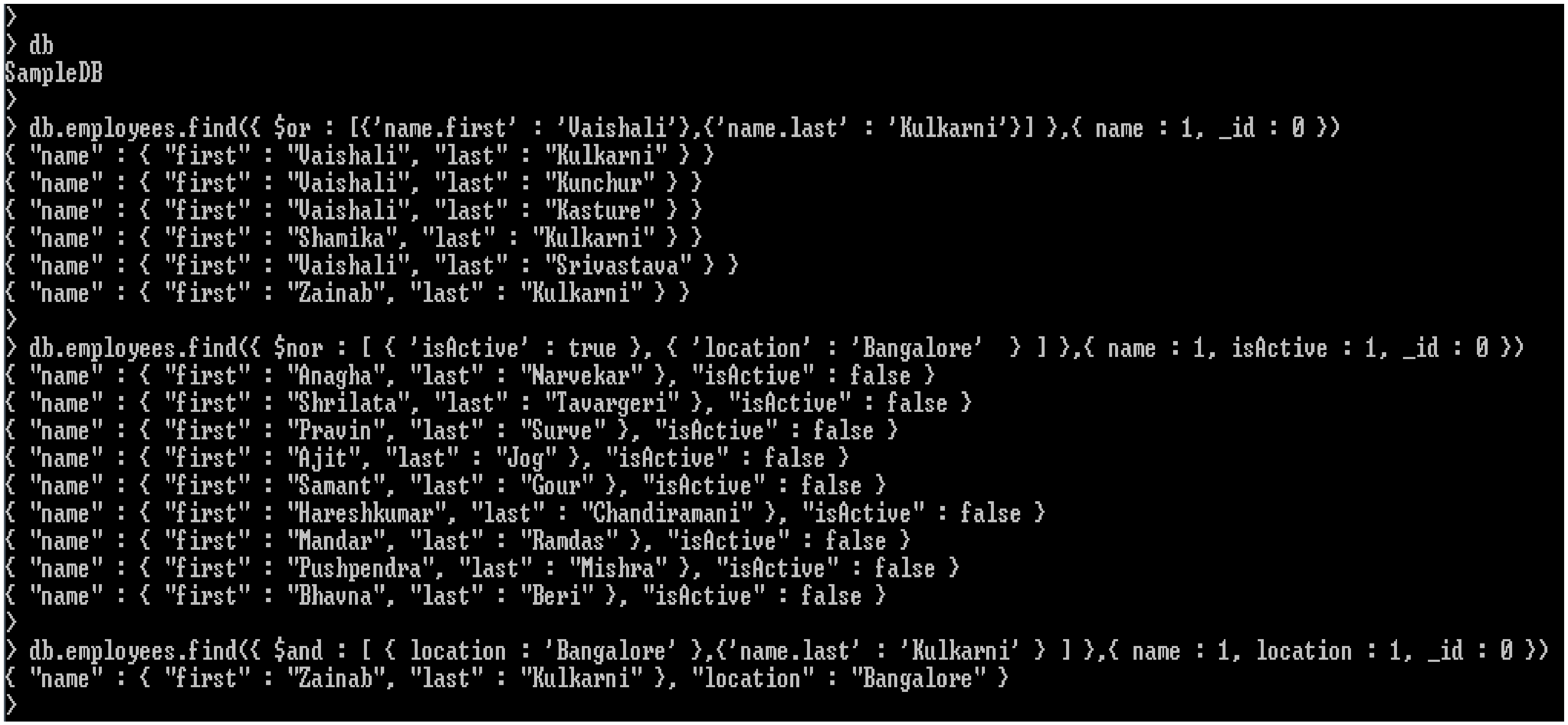
Demo

- Comparison Operators

Logical Operators

Name	Description
$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
$nor	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
$not	Inverts the effect of a query expression and returns documents that do not match the query expression.
$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

Logical Operators – or, not & and



```
>
> db
SampleDB
>
> db.employees.find({ $or : [{'name.first' : 'Vaishali'},{'name.last' : 'Kulkarni'}] },{ name : 1, _id : 0 })
{ "name" : { "first" : "Vaishali", "last" : "Kulkarni" } }
{ "name" : { "first" : "Vaishali", "last" : "Kunchur" } }
{ "name" : { "first" : "Vaishali", "last" : "Kasture" } }
{ "name" : { "first" : "Shamika", "last" : "Kulkarni" } }
{ "name" : { "first" : "Vaishali", "last" : "Srivastava" } }
{ "name" : { "first" : "Zainab", "last" : "Kulkarni" } }
>
> db.employees.find({ $nor : [ { 'isActive' : true }, { 'location' : 'Bangalore'  } ] },{ name : 1, isActive : 1, _id : 0 })
{ "name" : { "first" : "Anagha", "last" : "Narvekar" }, "isActive" : false }
{ "name" : { "first" : "Shrilata", "last" : "Tavargeri" }, "isActive" : false }
{ "name" : { "first" : "Pravin", "last" : "Surve" }, "isActive" : false }
{ "name" : { "first" : "Ajit", "last" : "Jog" }, "isActive" : false }
{ "name" : { "first" : "Samant", "last" : "Gour" }, "isActive" : false }
{ "name" : { "first" : "Hareshkumar", "last" : "Chandiramani" }, "isActive" : false }
{ "name" : { "first" : "Mandar", "last" : "Ramdas" }, "isActive" : false }
{ "name" : { "first" : "Pushpendra", "last" : "Mishra" }, "isActive" : false }
{ "name" : { "first" : "Bhavna", "last" : "Beri" }, "isActive" : false }
>
> db.employees.find({ $and : [ { location : 'Bangalore' },{'name.last' : 'Kulkarni' } ] },{ name : 1, location : 1, _id : 0 })
{ "name" : { "first" : "Zainab", "last" : "Kulkarni" }, "location" : "Bangalore" }
>
```

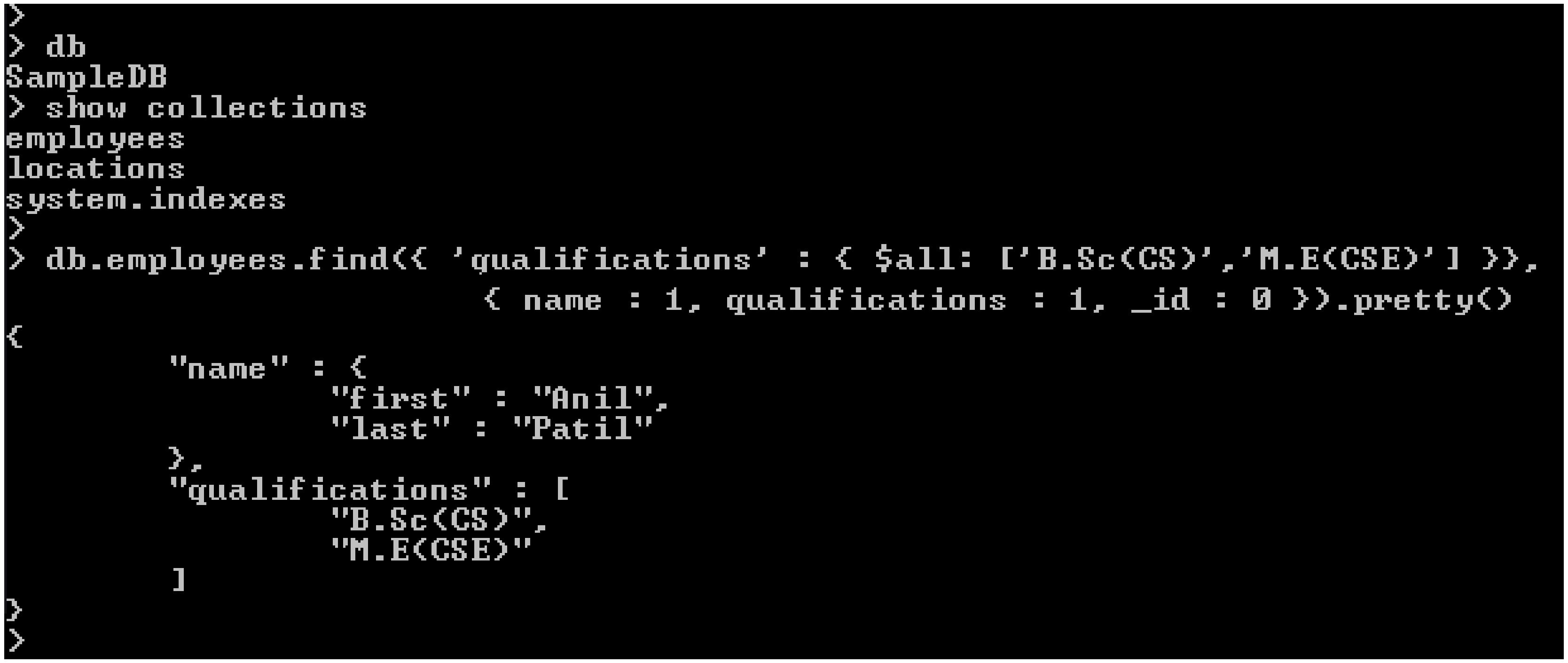
Demo

- Logical Operators

MongoDB Additional operators

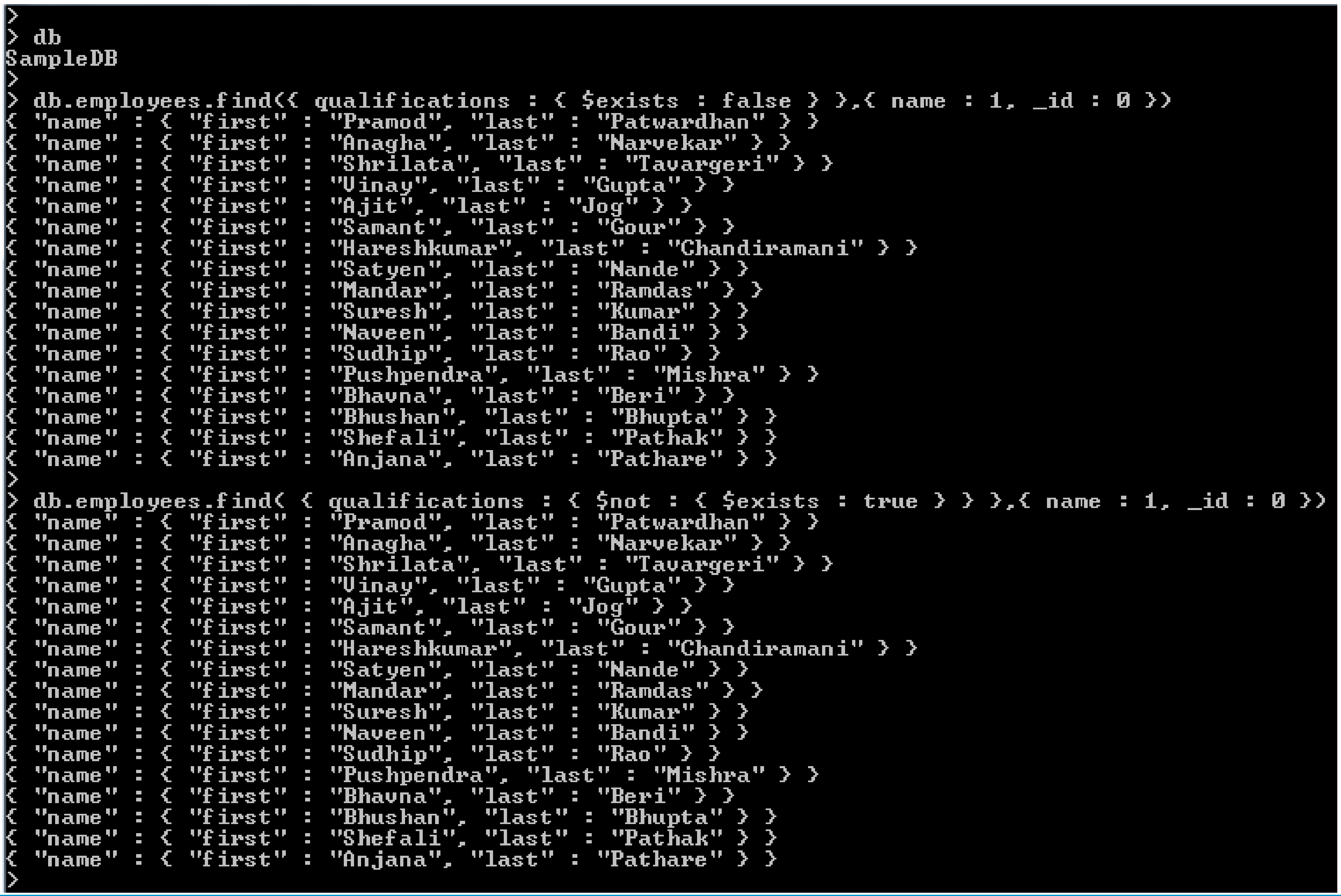
Name	Description
$all	Matches arrays that contain all elements specified in the query.
$exists	Matches documents that have the specified field.
$regex	Selects documents where values match a specified regular expression.
$where	Matches documents that satisfy a JavaScript expression.
$sort	Reorders the document stream by a specified sort key. Only the order changes; the documents remain unmodified. For each input document, outputs one document.
$limit	Passes the first n documents unmodified to the pipeline where n is the specified limit. For each input document, outputs either one document (for the first n documents) or zero documents (after the first n documents).
$skip	Skips the first n documents where n is the specified skip number and passes the remaining documents unmodified to the pipeline. For each input document, outputs either zero documents (for the first n documents) or one document (if after the first n documents).

MongoDB Additional operators - all



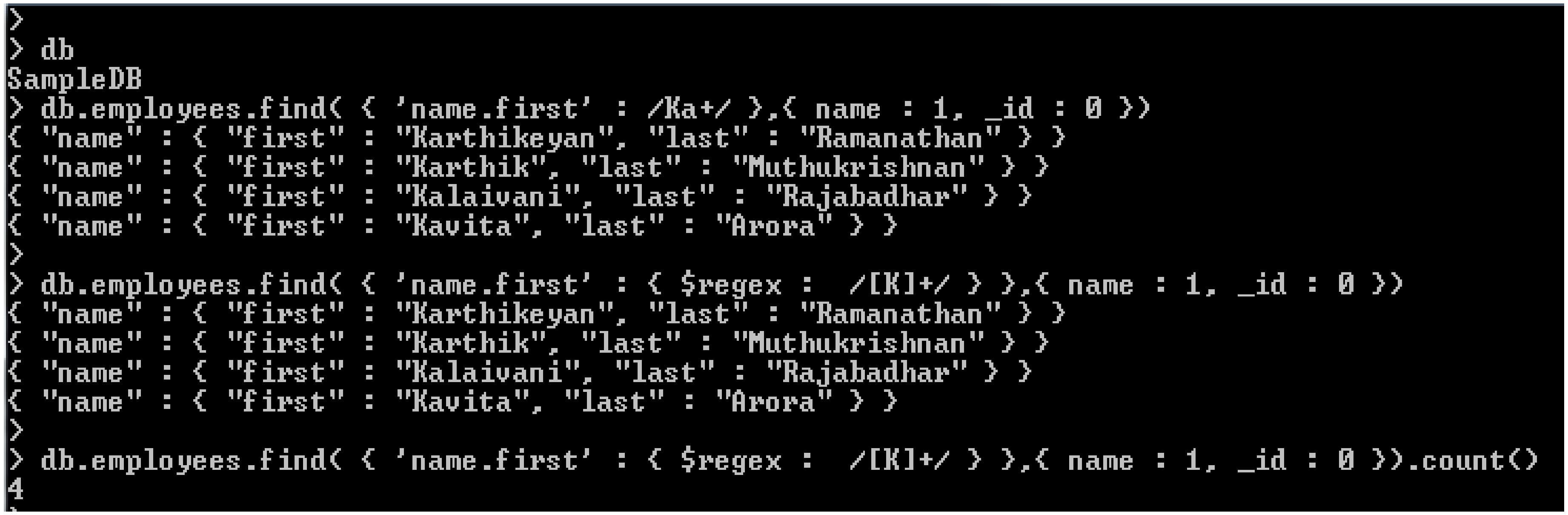
```
>
> db
SampleDB
> show collections
employees
locations
system.indexes
>
> db.employees.find({ 'qualifications' : { $all: ['B.Sc(CS)','M.E(CSE)'] }},
                        { name : 1, qualifications : 1, _id : 0 }).pretty()
{
        "name" : {
                "first" : "Anil",
                "last" : "Patil"
        },
        "qualifications" : [
                "B.Sc(CS)",
                "M.E(CSE)"
        ]
}
>
```

MongoDB Additional operators – exists & not



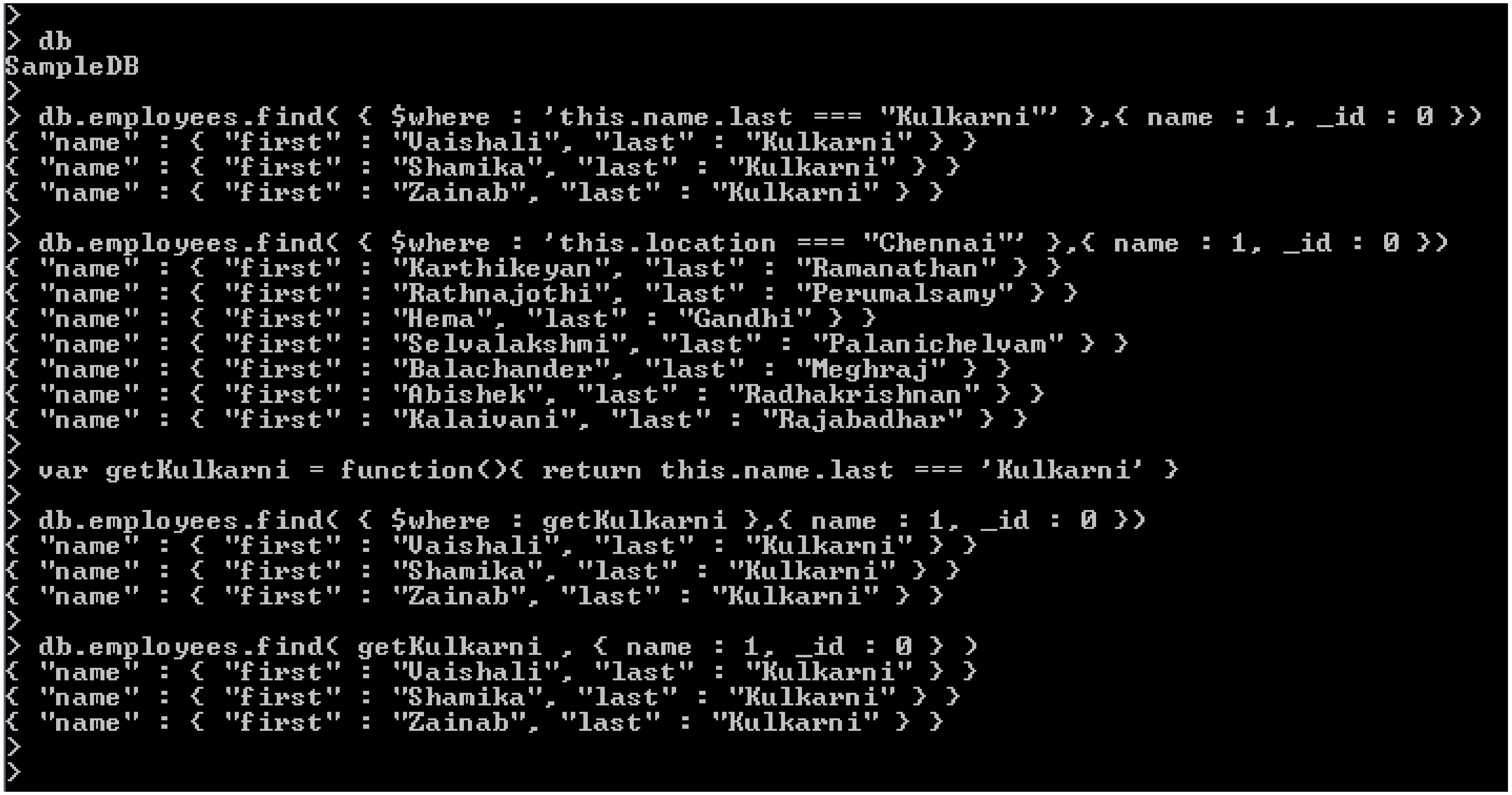
```
>
> db
SampleDB
>
> db.employees.find({ qualifications : { $exists : false } },{ name : 1, _id : 0 })
{ "name" : { "first" : "Pramod", "last" : "Patwardhan" } }
{ "name" : { "first" : "Anagha", "last" : "Narvekar" } }
{ "name" : { "first" : "Shrilata", "last" : "Tavargeri" } }
{ "name" : { "first" : "Vinay", "last" : "Gupta" } }
{ "name" : { "first" : "Ajit", "last" : "Jog" } }
{ "name" : { "first" : "Samant", "last" : "Gour" } }
{ "name" : { "first" : "Hareshkumar", "last" : "Chandiramani" } }
{ "name" : { "first" : "Satyen", "last" : "Nande" } }
{ "name" : { "first" : "Mandar", "last" : "Ramdas" } }
{ "name" : { "first" : "Suresh", "last" : "Kumar" } }
{ "name" : { "first" : "Naveen", "last" : "Bandi" } }
{ "name" : { "first" : "Sudhip", "last" : "Rao" } }
{ "name" : { "first" : "Pushpendra", "last" : "Mishra" } }
{ "name" : { "first" : "Bhavna", "last" : "Beri" } }
{ "name" : { "first" : "Bhushan", "last" : "Bhupta" } }
{ "name" : { "first" : "Shefali", "last" : "Pathak" } }
{ "name" : { "first" : "Anjana", "last" : "Pathare" } }
>
> db.employees.find( { qualifications : { $not : { $exists : true } } },{ name : 1, _id : 0 })
{ "name" : { "first" : "Pramod", "last" : "Patwardhan" } }
{ "name" : { "first" : "Anagha", "last" : "Narvekar" } }
{ "name" : { "first" : "Shrilata", "last" : "Tavargeri" } }
{ "name" : { "first" : "Vinay", "last" : "Gupta" } }
{ "name" : { "first" : "Ajit", "last" : "Jog" } }
{ "name" : { "first" : "Samant", "last" : "Gour" } }
{ "name" : { "first" : "Hareshkumar", "last" : "Chandiramani" } }
{ "name" : { "first" : "Satyen", "last" : "Nande" } }
{ "name" : { "first" : "Mandar", "last" : "Ramdas" } }
{ "name" : { "first" : "Suresh", "last" : "Kumar" } }
{ "name" : { "first" : "Naveen", "last" : "Bandi" } }
{ "name" : { "first" : "Sudhip", "last" : "Rao" } }
{ "name" : { "first" : "Pushpendra", "last" : "Mishra" } }
{ "name" : { "first" : "Bhavna", "last" : "Beri" } }
{ "name" : { "first" : "Bhushan", "last" : "Bhupta" } }
{ "name" : { "first" : "Shefali", "last" : "Pathak" } }
{ "name" : { "first" : "Anjana", "last" : "Pathare" } }
>
```

MongoDB Additional operators - regex



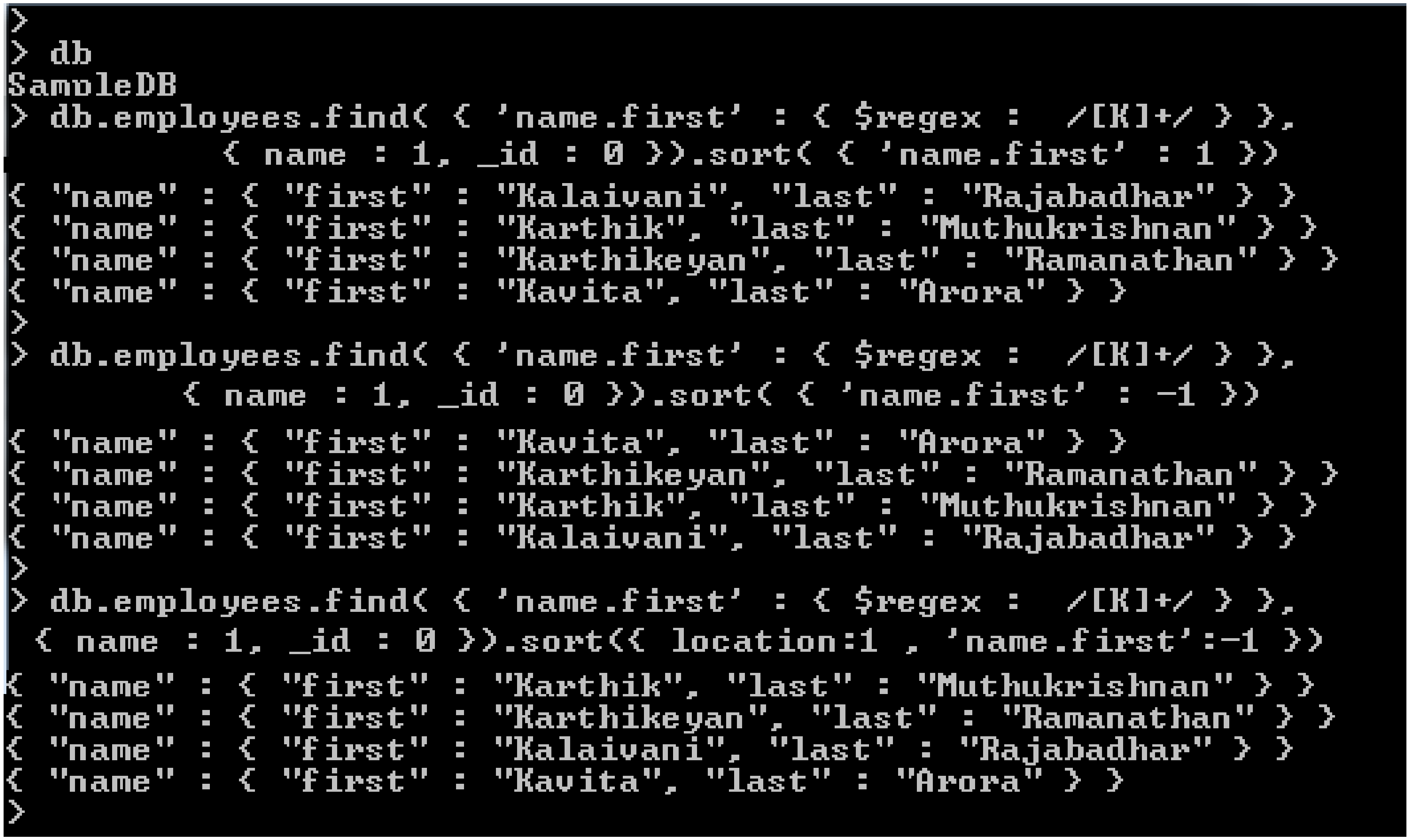
```
>
> db
SampleDB
> db.employees.find( { 'name.first' : /Ka+/ },{ name : 1, _id : 0 })
{ "name" : { "first" : "Karthikeyan", "last" : "Ramanathan" } }
{ "name" : { "first" : "Karthik", "last" : "Muthukrishnan" } }
{ "name" : { "first" : "Kalaivani", "last" : "Rajabadhar" } }
{ "name" : { "first" : "Kavita", "last" : "Arora" } }
>
> db.employees.find( { 'name.first' : { $regex :  /[K]+/ } },{ name : 1, _id : 0 })
{ "name" : { "first" : "Karthikeyan", "last" : "Ramanathan" } }
{ "name" : { "first" : "Karthik", "last" : "Muthukrishnan" } }
{ "name" : { "first" : "Kalaivani", "last" : "Rajabadhar" } }
{ "name" : { "first" : "Kavita", "last" : "Arora" } }
>
> db.employees.find( { 'name.first' : { $regex :  /[K]+/ } },{ name : 1, _id : 0 }).count()
4
```

MongoDB Additional operators - where



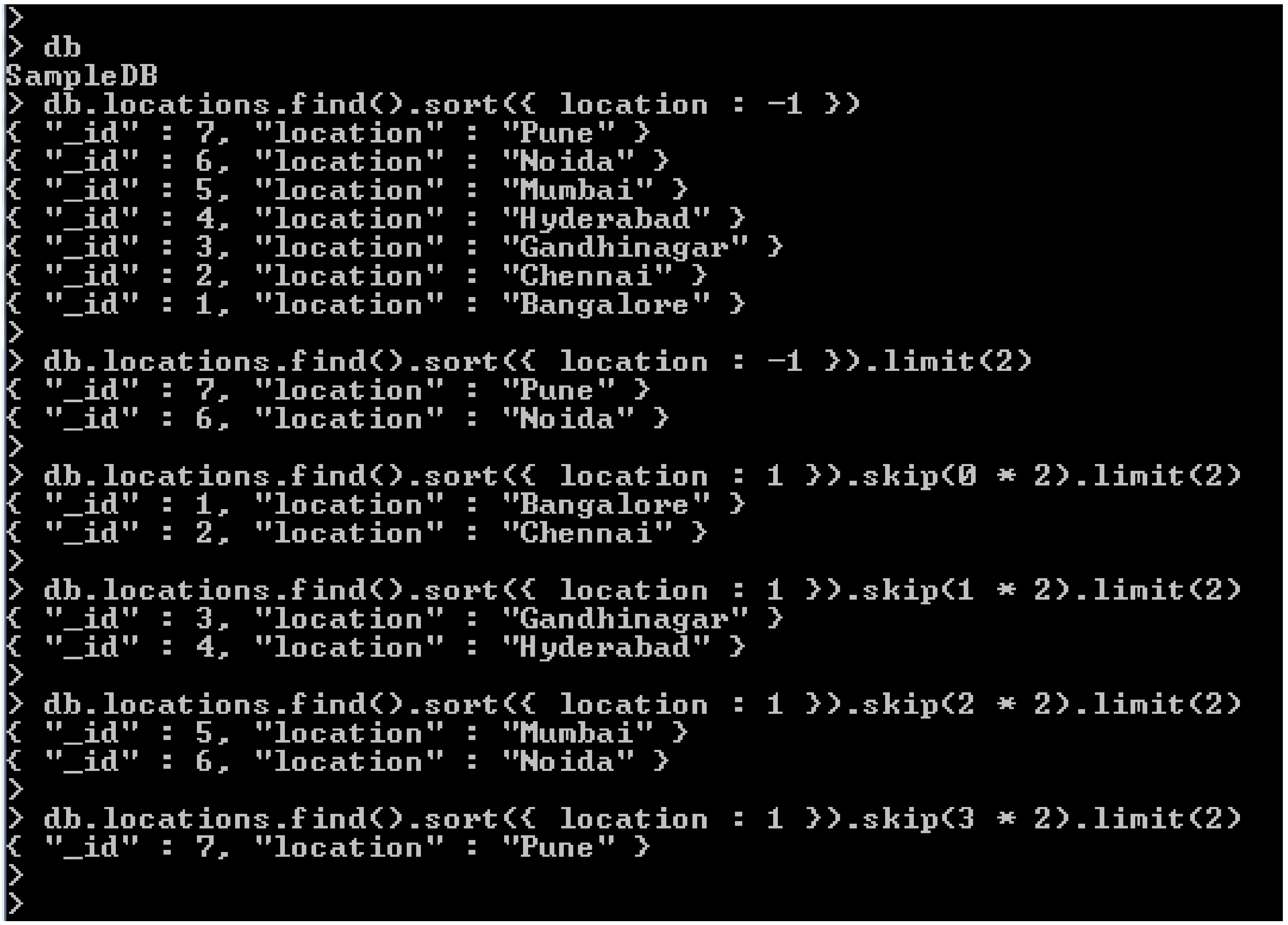
```
>
> db
SampleDB
>
> db.employees.find( { $where : 'this.name.last === "Kulkarni"' },{ name : 1, _id : 0 })
{ "name" : { "first" : "Vaishali", "last" : "Kulkarni" } }
{ "name" : { "first" : "Shamika", "last" : "Kulkarni" } }
{ "name" : { "first" : "Zainab", "last" : "Kulkarni" } }
>
> db.employees.find( { $where : 'this.location === "Chennai"' },{ name : 1, _id : 0 })
{ "name" : { "first" : "Karthikeyan", "last" : "Ramanathan" } }
{ "name" : { "first" : "Rathnajothi", "last" : "Perumalsamy" } }
{ "name" : { "first" : "Hema", "last" : "Gandhi" } }
{ "name" : { "first" : "Selvalakshmi", "last" : "Palanichelvam" } }
{ "name" : { "first" : "Balachander", "last" : "Meghraj" } }
{ "name" : { "first" : "Abishek", "last" : "Radhakrishnan" } }
{ "name" : { "first" : "Kalaivani", "last" : "Rajabadhar" } }
>
> var getKulkarni = function(){ return this.name.last === 'Kulkarni' }
>
> db.employees.find( { $where : getKulkarni },{ name : 1, _id : 0 })
{ "name" : { "first" : "Vaishali", "last" : "Kulkarni" } }
{ "name" : { "first" : "Shamika", "last" : "Kulkarni" } }
{ "name" : { "first" : "Zainab", "last" : "Kulkarni" } }
>
> db.employees.find( getKulkarni , { name : 1, _id : 0 } )
{ "name" : { "first" : "Vaishali", "last" : "Kulkarni" } }
{ "name" : { "first" : "Shamika", "last" : "Kulkarni" } }
{ "name" : { "first" : "Zainab", "last" : "Kulkarni" } }
>
>
```

MongoDB Additional operators - sort



```
>
> db
SampleDB
> db.employees.find( { 'name.first' : { $regex :  /[K]+/ } },
           { name : 1, _id : 0 }).sort( { 'name.first' : 1 })
{ "name" : { "first" : "Kalaivani", "last" : "Rajabadhar" } }
{ "name" : { "first" : "Karthik", "last" : "Muthukrishnan" } }
{ "name" : { "first" : "Karthikeyan", "last" : "Ramanathan" } }
{ "name" : { "first" : "Kavita", "last" : "Arora" } }
>
> db.employees.find( { 'name.first' : { $regex :  /[K]+/ } },
         { name : 1, _id : 0 }).sort( { 'name.first' : -1 })
{ "name" : { "first" : "Kavita", "last" : "Arora" } }
{ "name" : { "first" : "Karthikeyan", "last" : "Ramanathan" } }
{ "name" : { "first" : "Karthik", "last" : "Muthukrishnan" } }
{ "name" : { "first" : "Kalaivani", "last" : "Rajabadhar" } }
>
> db.employees.find( { 'name.first' : { $regex :  /[K]+/ } },
 { name : 1, _id : 0 }).sort({ location:1 , 'name.first':-1 })
{ "name" : { "first" : "Karthik", "last" : "Muthukrishnan" } }
{ "name" : { "first" : "Karthikeyan", "last" : "Ramanathan" } }
{ "name" : { "first" : "Kalaivani", "last" : "Rajabadhar" } }
{ "name" : { "first" : "Kavita", "last" : "Arora" } }
>
```

MongoDB Additional operators – limit & skip

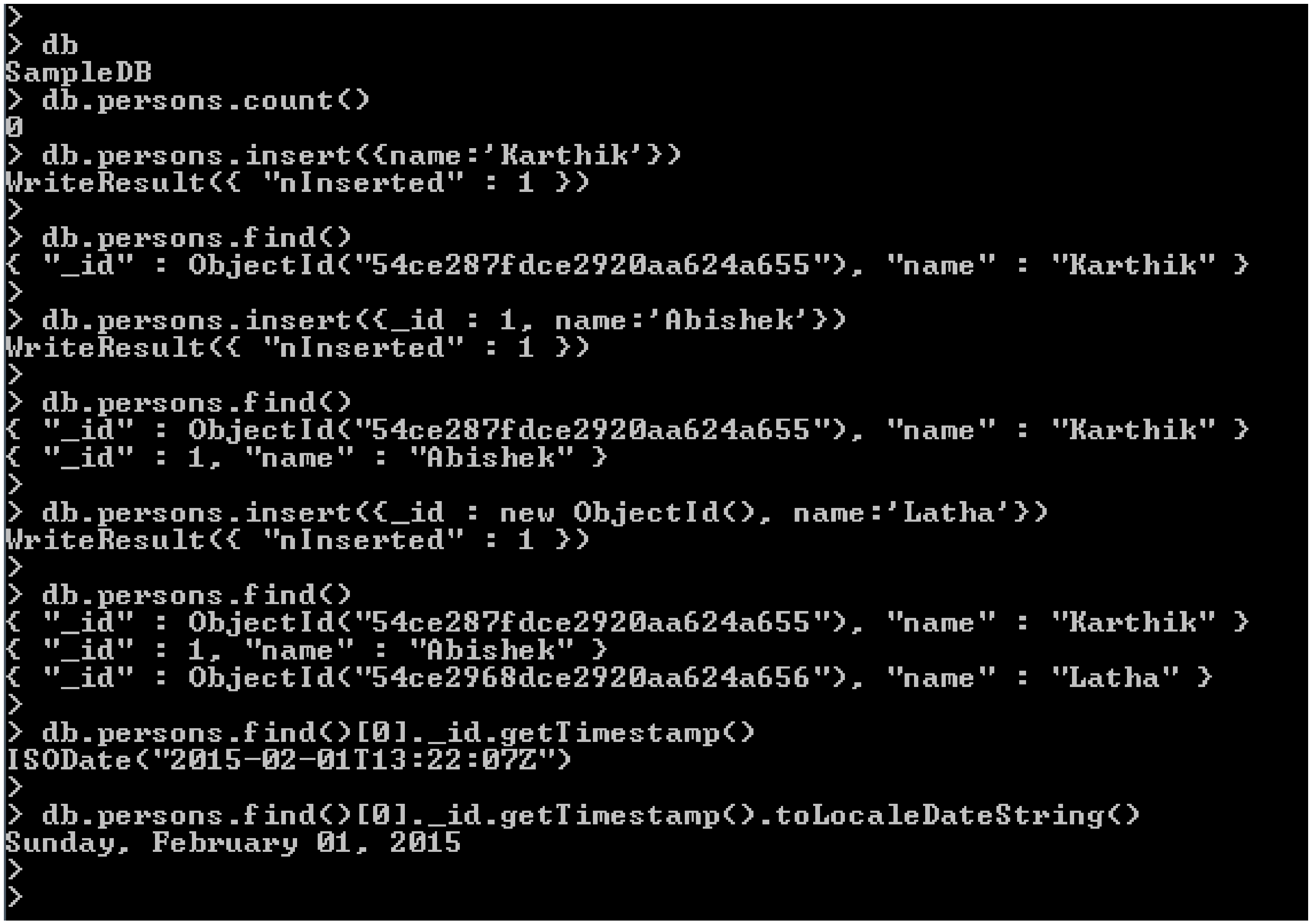


```
>
> db
SampleDB
> db.locations.find().sort({ location : -1 })
{ "_id" : 7, "location" : "Pune" }
{ "_id" : 6, "location" : "Noida" }
{ "_id" : 5, "location" : "Mumbai" }
{ "_id" : 4, "location" : "Hyderabad" }
{ "_id" : 3, "location" : "Gandhinagar" }
{ "_id" : 2, "location" : "Chennai" }
{ "_id" : 1, "location" : "Bangalore" }
>
> db.locations.find().sort({ location : -1 }).limit(2)
{ "_id" : 7, "location" : "Pune" }
{ "_id" : 6, "location" : "Noida" }
>
> db.locations.find().sort({ location : 1 }).skip(0 * 2).limit(2)
{ "_id" : 1, "location" : "Bangalore" }
{ "_id" : 2, "location" : "Chennai" }
>
> db.locations.find().sort({ location : 1 }).skip(1 * 2).limit(2)
{ "_id" : 3, "location" : "Gandhinagar" }
{ "_id" : 4, "location" : "Hyderabad" }
>
> db.locations.find().sort({ location : 1 }).skip(2 * 2).limit(2)
{ "_id" : 5, "location" : "Mumbai" }
{ "_id" : 6, "location" : "Noida" }
>
> db.locations.find().sort({ location : 1 }).skip(3 * 2).limit(2)
{ "_id" : 7, "location" : "Pune" }
>
>
```

Demo

- Additional Operators

Inserting Document(s)



```
>
> db
SampleDB
> db.persons.count()
0
> db.persons.insert({name:'Karthik'})
WriteResult({ "nInserted" : 1 })
>
> db.persons.find()
{ "_id" : ObjectId("54ce287fdce2920aa624a655"), "name" : "Karthik" }
>
> db.persons.insert({_id : 1, name:'Abishek'})
WriteResult({ "nInserted" : 1 })
>
> db.persons.find()
{ "_id" : ObjectId("54ce287fdce2920aa624a655"), "name" : "Karthik" }
{ "_id" : 1, "name" : "Abishek" }
>
> db.persons.insert({_id : new ObjectId(), name:'Latha'})
WriteResult({ "nInserted" : 1 })
>
> db.persons.find()
{ "_id" : ObjectId("54ce287fdce2920aa624a655"), "name" : "Karthik" }
{ "_id" : 1, "name" : "Abishek" }
{ "_id" : ObjectId("54ce2968dce2920aa624a656"), "name" : "Latha" }
>
> db.persons.find()[0]._id.getTimestamp()
ISODate("2015-02-01T13:22:07Z")
>
> db.persons.find()[0]._id.getTimestamp().toLocaleDateString()
Sunday, February 01, 2015
>
>
```

ObjectId

- In MongoDB, documents stored in a collection require a unique _id field that acts as a primary key.
- MongoDB uses ObjectIds as the default value for the _id field, if the _id field is not specified by the user.
- ObjectId is a 12-byte BSON type constructed using:
  - 4-byte value representing the seconds since the Unix time
  - 3-byte machine identifier
  - 2-byte process id
  - 3-byte counter, starting with a random value.
- In the mongo shell, you can access the creation time of the ObjectId, using the getTimestamp() method. Sorting on an _id field that stores ObjectId values is roughly equivalent to sorting by creation time.
- To generate a new ObjectId, use the ObjectId() constructor with no argument

Demo

- Inserting Documents

Updating Document(s)

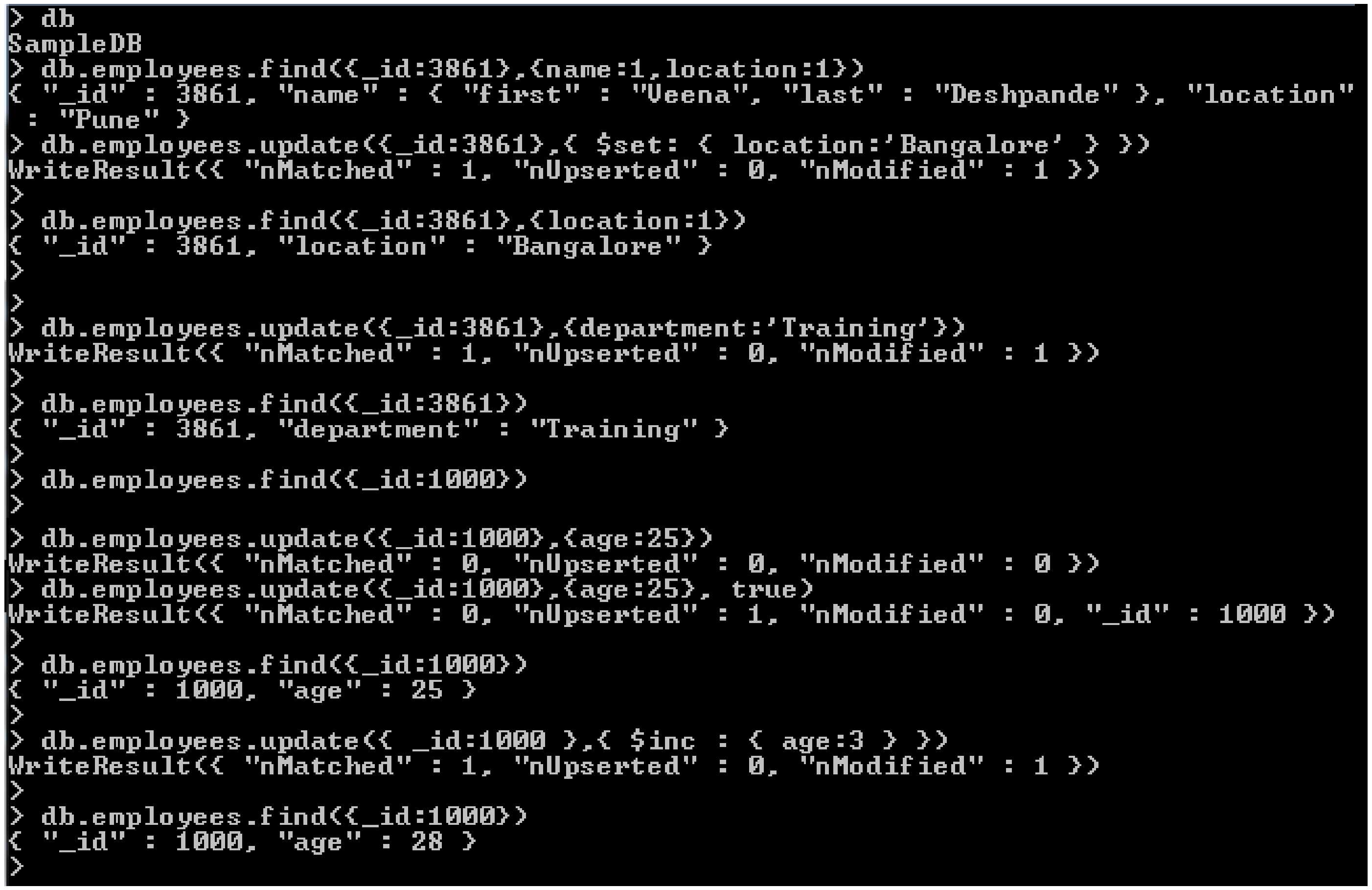
- db.collection.update() modifies an existing document or documents in a collection. The method can modify specific fields of an existing document or documents or replace an existing document entirely, depending on the update parameter.
- By default, the update() method updates a single document

```
db.collection.update(
  <query>,
  <update>,
  {
    upsert: <boolean>,
    multi: <boolean>
  })
```

- **upsert** : creates a new document when no document matches the query criteria.
- **multi** : updates multiple documents that meet the query criteria.

Updating Document(s)

- $inc field update operator is used to increment and the $set field update operator is used to replace the value of the field.

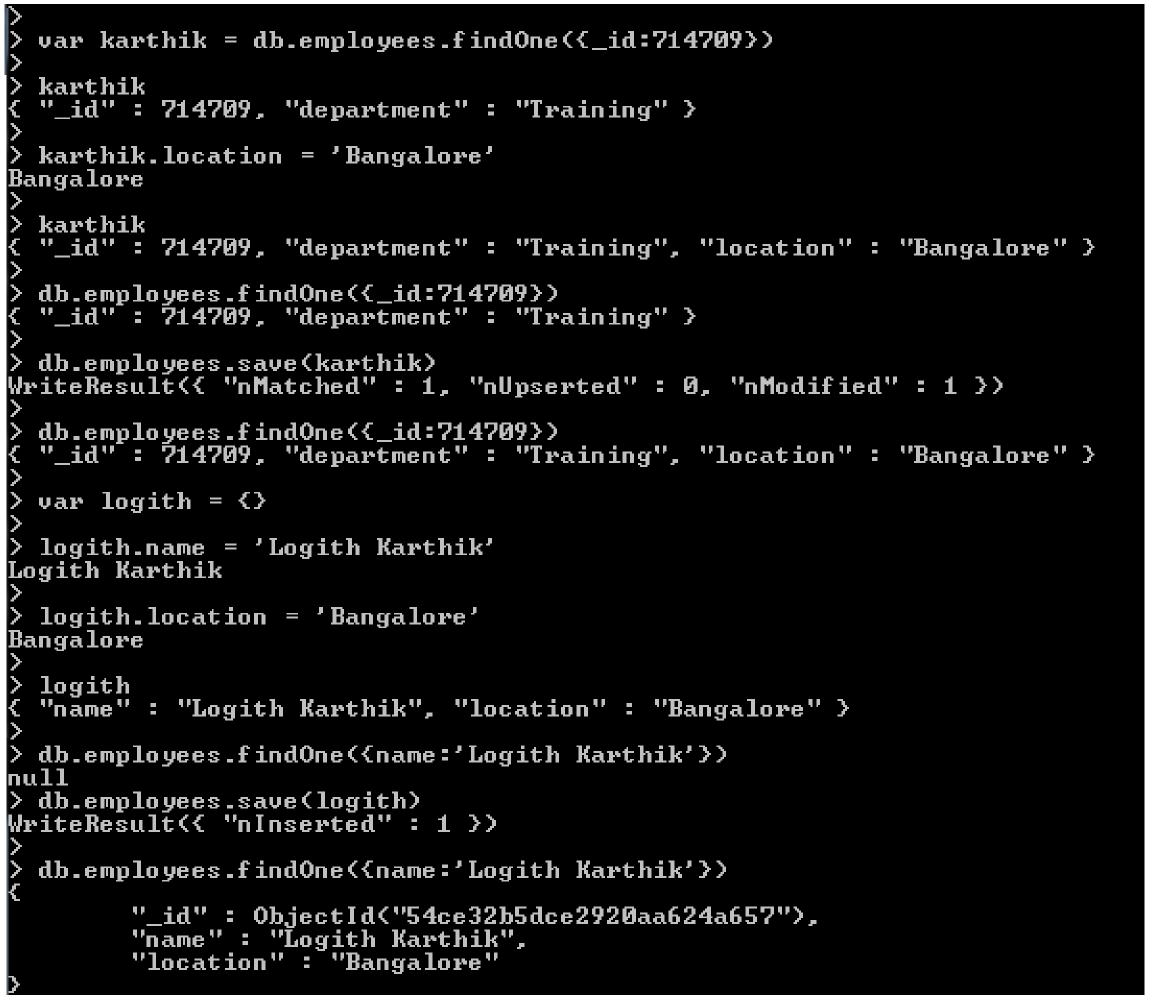


```
> db
SampleDB
> db.employees.find({_id:3861},{name:1,location:1})
{ "_id" : 3861, "name" : { "first" : "Veena", "last" : "Deshpande" }, "location"
 : "Pune" }
> db.employees.update({_id:3861},{ $set: { location:'Bangalore' } })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.employees.find({_id:3861},{location:1})
{ "_id" : 3861, "location" : "Bangalore" }
>
>
> db.employees.update({_id:3861},{department:'Training'})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.employees.find({_id:3861})
{ "_id" : 3861, "department" : "Training" }
>
> db.employees.find({_id:1000})
>
> db.employees.update({_id:1000},{age:25})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
> db.employees.update({_id:1000},{age:25}, true)
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 1000 })
>
> db.employees.find({_id:1000})
{ "_id" : 1000, "age" : 25 }
>
> db.employees.update({ _id:1000 },{ $inc : { age:3 } })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.employees.find({_id:1000})
{ "_id" : 1000, "age" : 28 }
>
```

Updating Document - save

- db.collection.save()

  updates an existing document (when _id is present already) or inserts a new document (with new ObjectId)



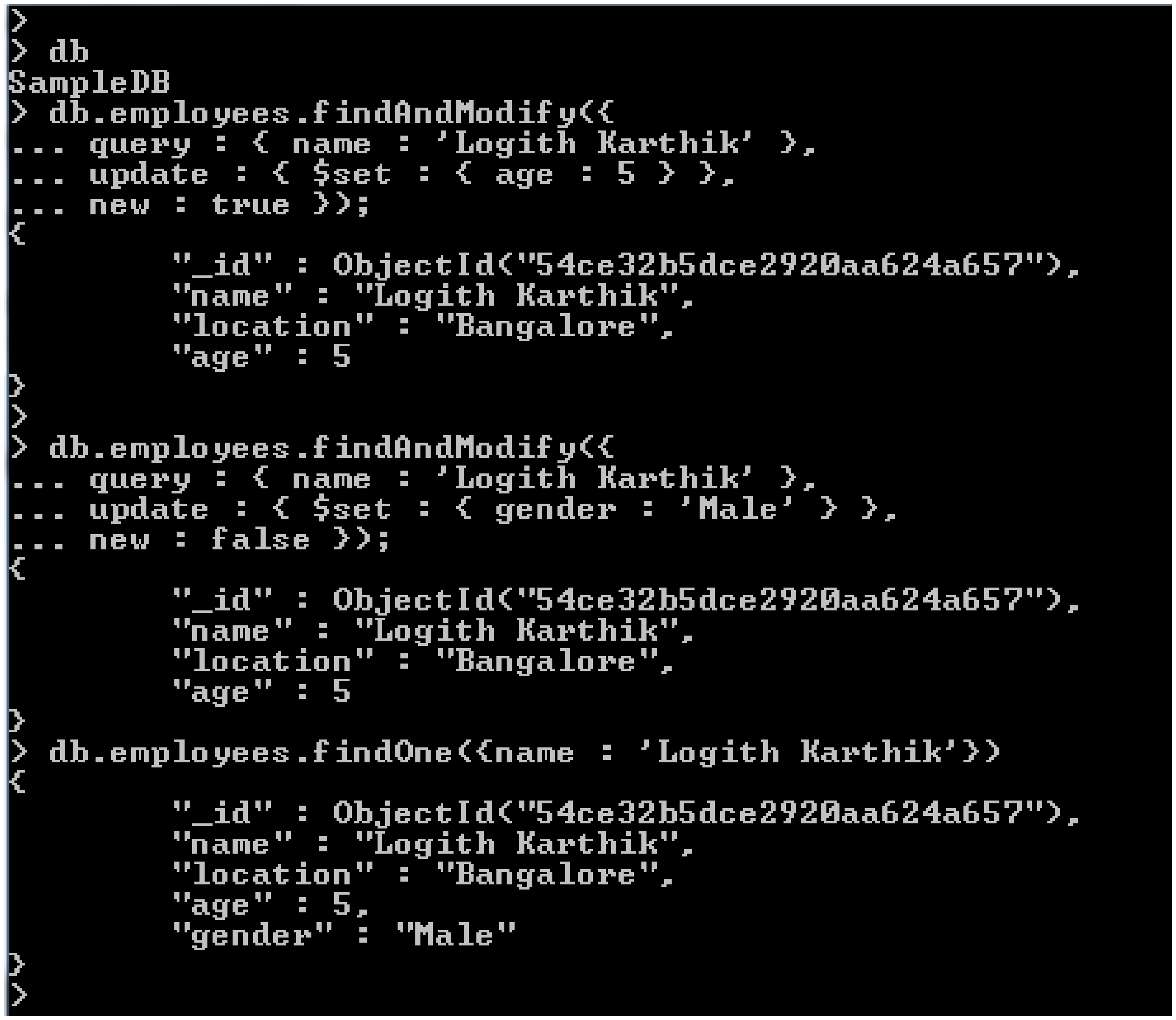
```
>
> var karthik = db.employees.findOne({_id:714709})
>
> karthik
{ "_id" : 714709, "department" : "Training" }
>
> karthik.location = 'Bangalore'
Bangalore
>
> karthik
{ "_id" : 714709, "department" : "Training", "location" : "Bangalore" }
>
> db.employees.findOne({_id:714709})
{ "_id" : 714709, "department" : "Training" }
>
> db.employees.save(karthik)
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.employees.findOne({_id:714709})
{ "_id" : 714709, "department" : "Training", "location" : "Bangalore" }
>
> var logith = {}
>
> logith.name = 'Logith Karthik'
Logith Karthik
>
> logith.location = 'Bangalore'
Bangalore
>
> logith
{ "name" : "Logith Karthik", "location" : "Bangalore" }
>
> db.employees.findOne({name:'Logith Karthik'})
null
> db.employees.save(logith)
WriteResult({ "nInserted" : 1 })
>
> db.employees.findOne({name:'Logith Karthik'})
{
        "_id" : ObjectId("54ce32b5dce2920aa624a657"),
        "name" : "Logith Karthik",
        "location" : "Bangalore"
}
```

Updating Document - findAndModify

- db.collection.findAndModify(<document>) modifies and returns a single document. By default, the returned document does not include the modifications made on the update. To return the document with the modifications made on the update, use the new option.
- The findAndModify() method has the following form

```
db.collection.findAndModify({
   query: <document>,
   sort: <document>,
   remove: <boolean>,
   update: <document>,
   new: <boolean>,
   fields: <document>,
   upsert: <boolean>
});
```

Updating Document - findAndModify



```
>
> db
SampleDB
> db.employees.findAndModify({
... query : { name : 'Logith Karthik' },
... update : { $set : { age : 5 } },
... new : true });
{
        "_id" : ObjectId("54ce32b5dce2920aa624a657"),
        "name" : "Logith Karthik",
        "location" : "Bangalore",
        "age" : 5
}
>
> db.employees.findAndModify({
... query : { name : 'Logith Karthik' },
... update : { $set : { gender : 'Male' } },
... new : false });
{
        "_id" : ObjectId("54ce32b5dce2920aa624a657"),
        "name" : "Logith Karthik",
        "location" : "Bangalore",
        "age" : 5
}
> db.employees.findOne({name : 'Logith Karthik'})
{
        "_id" : ObjectId("54ce32b5dce2920aa624a657"),
        "name" : "Logith Karthik",
        "location" : "Bangalore",
        "age" : 5,
        "gender" : "Male"
}
>
```

Auto-Incrementing Sequence Field

- In MongoDB, by default we cannot use an auto-increment pattern for the _id field, or any field, because it does not scale for databases with large numbers of documents. Typically the default value ObjectId is more ideal for the _id.
- But still we can create an auto-Incrementing sequence field by creating a collection on our own to maintain the sequence.
- By calling generateSequence('test') will create a document under counters collection and maintains the sequence for the same.

```
function generateSequence(name) {
  var ret = db.counters.findAndModify(
       {
         query: { _id: name },
         update: { $inc: { seq: 1 } },
         new: true,
         upsert:true
       }
  );
  return ret.seq;
}
```

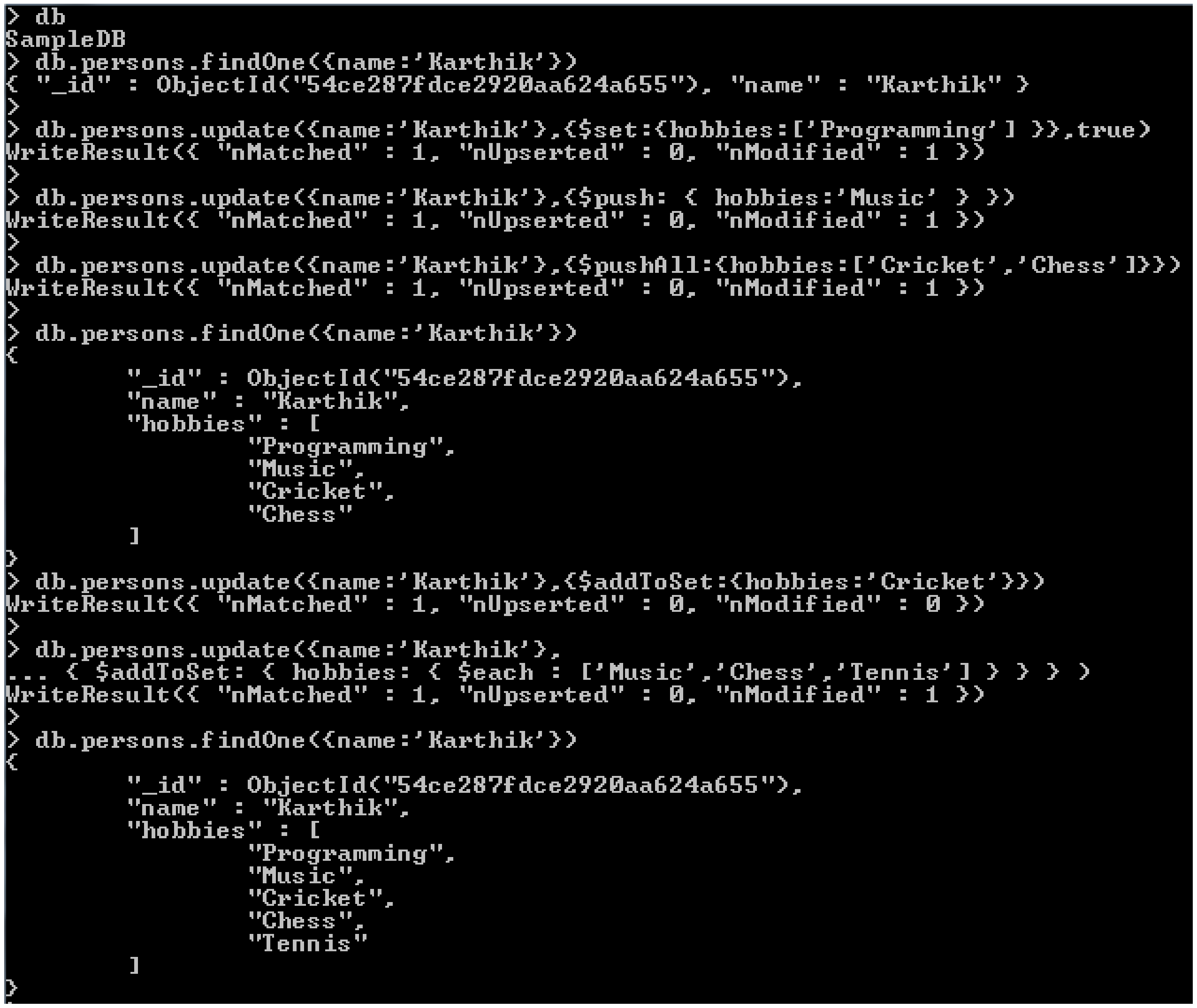
Demo

- Creating Auto Increment Field

Array Update Operators

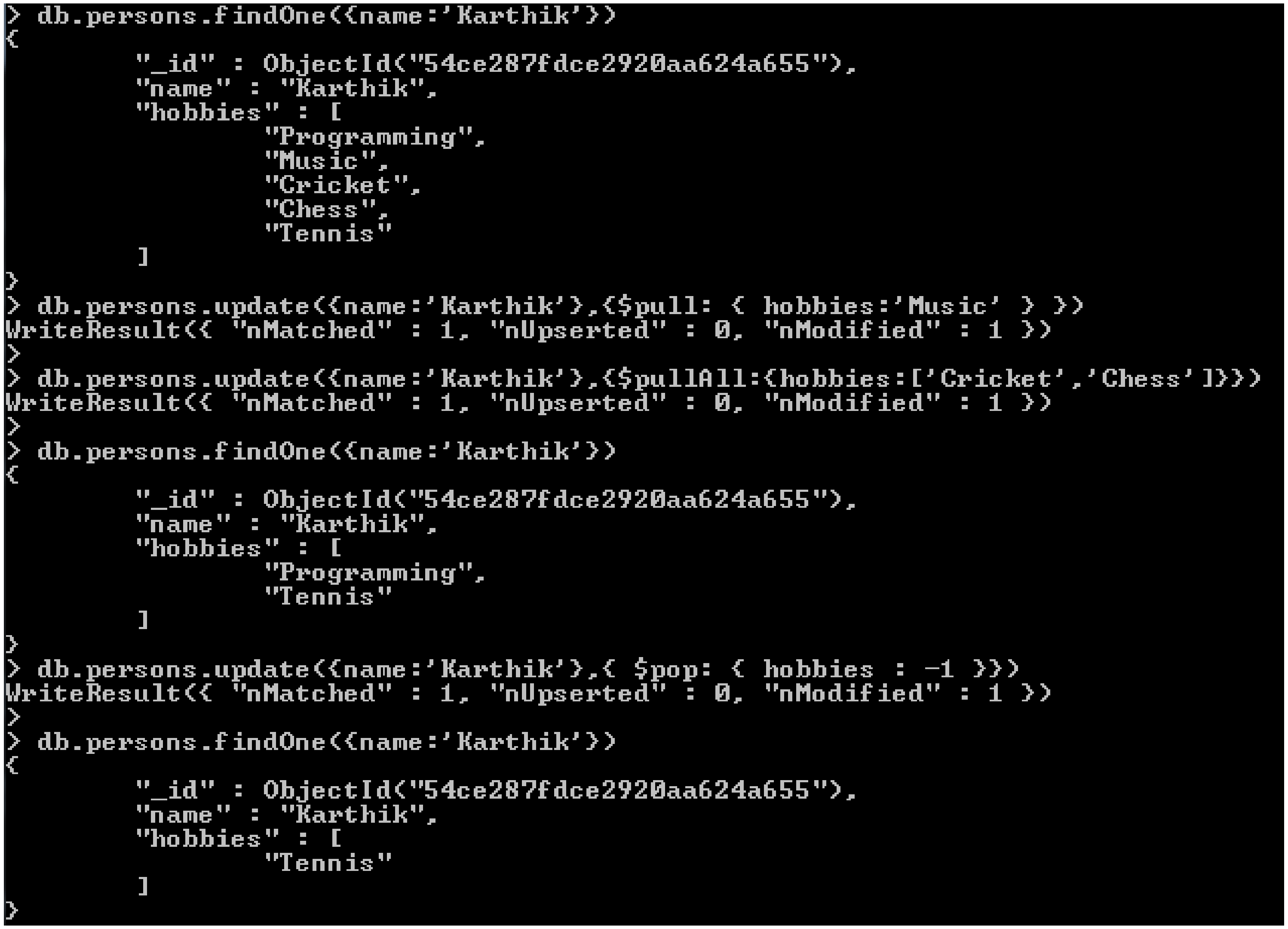
Name	Description
$	Acts as a placeholder to update the first element that matches the query condition in an update.
$addToSet	Adds elements to an array only if they do not already exist in the set.
$pop	Removes the first or last item of an array.
$pullAll	Removes all matching values from an array.
$pull	Removes all array elements that match a specified query.
$pushAll	Deprecated. Adds several items to an array.
$push	Adds an item to an array.

Array Update Operators – Adding array items



```
> db
SampleDB
> db.persons.findOne({name:'Karthik'})
{ "_id" : ObjectId("54ce287fdce2920aa624a655"), "name" : "Karthik" }
>
> db.persons.update({name:'Karthik'},{$set:{hobbies:['Programming'] }},true)
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.persons.update({name:'Karthik'},{$push: { hobbies:'Music' } })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.persons.update({name:'Karthik'},{$pushAll:{hobbies:['Cricket','Chess']}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.persons.findOne({name:'Karthik'})
{
        "_id" : ObjectId("54ce287fdce2920aa624a655"),
        "name" : "Karthik",
        "hobbies" : [
                "Programming",
                "Music",
                "Cricket",
                "Chess"
        ]
}
> db.persons.update({name:'Karthik'},{$addToSet:{hobbies:'Cricket'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
>
> db.persons.update({name:'Karthik'},
... { $addToSet: { hobbies: { $each : ['Music','Chess','Tennis'] } } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.persons.findOne({name:'Karthik'})
{
        "_id" : ObjectId("54ce287fdce2920aa624a655"),
        "name" : "Karthik",
        "hobbies" : [
                "Programming",
                "Music",
                "Cricket",
                "Chess",
                "Tennis"
        ]
}
```

Array Update Operators – Removing array items



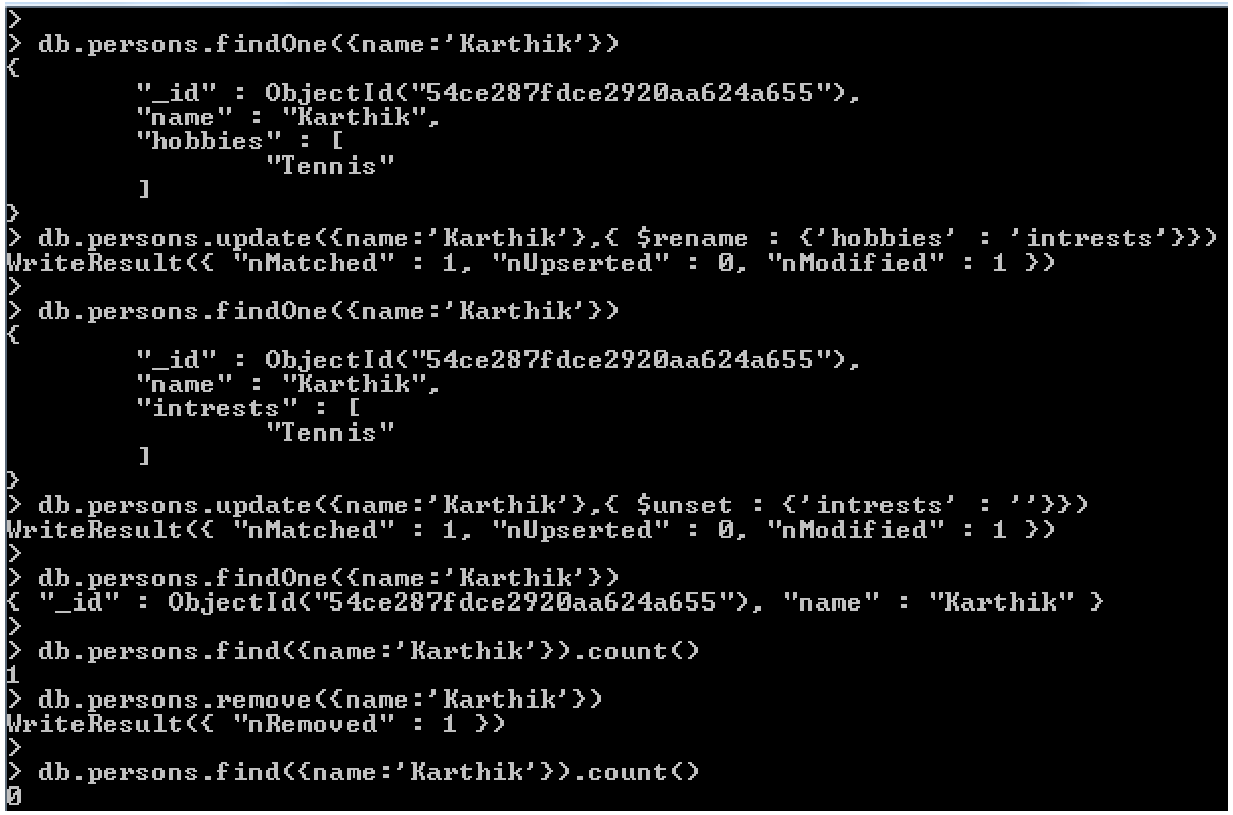
```
> db.persons.findOne({name:'Karthik'})
{
        "_id" : ObjectId("54ce287fdce2920aa624a655"),
        "name" : "Karthik",
        "hobbies" : [
                "Programming",
                "Music",
                "Cricket",
                "Chess",
                "Tennis"
        ]
}
> db.persons.update({name:'Karthik'},{$pull: { hobbies:'Music' } })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.persons.update({name:'Karthik'},{$pullAll:{hobbies:['Cricket','Chess']}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.persons.findOne({name:'Karthik'})
{
        "_id" : ObjectId("54ce287fdce2920aa624a655"),
        "name" : "Karthik",
        "hobbies" : [
                "Programming",
                "Tennis"
        ]
}
> db.persons.update({name:'Karthik'},{ $pop: { hobbies : -1 }})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.persons.findOne({name:'Karthik'})
{
        "_id" : ObjectId("54ce287fdce2920aa624a655"),
        "name" : "Karthik",
        "hobbies" : [
                "Tennis"
        ]
}
```

Demo

- Working with Array items

Renaming and Deleting fields and documents

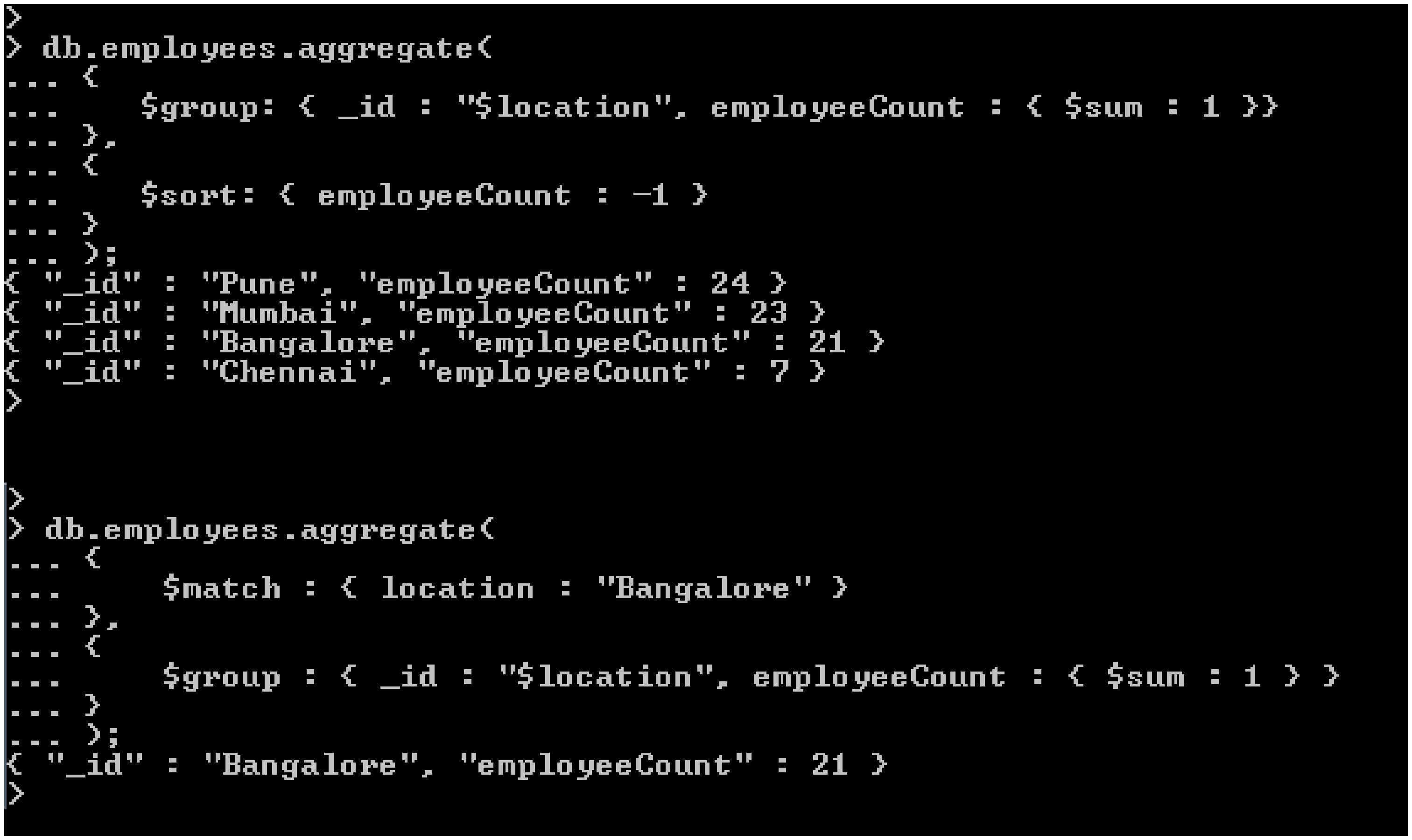
- The $unset field update operator is used to delete a particular field.
- The $rename field update operator updates the name of a field
- db.collection.remove() is used to remove documents from a collection



```
>
> db.persons.findOne({name:'Karthik'})
{
        "_id" : ObjectId("54ce287fdce2920aa624a655"),
        "name" : "Karthik",
        "hobbies" : [
                "Tennis"
        ]
}
> db.persons.update({name:'Karthik'},{ $rename : {'hobbies' : 'intrests'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.persons.findOne({name:'Karthik'})
{
        "_id" : ObjectId("54ce287fdce2920aa624a655"),
        "name" : "Karthik",
        "intrests" : [
                "Tennis"
        ]
}
> db.persons.update({name:'Karthik'},{ $unset : {'intrests' : ''}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.persons.findOne({name:'Karthik'})
{ "_id" : ObjectId("54ce287fdce2920aa624a655"), "name" : "Karthik" }
>
> db.persons.find({name:'Karthik'}).count()
1
> db.persons.remove({name:'Karthik'})
WriteResult({ "nRemoved" : 1 })
>
> db.persons.find({name:'Karthik'}).count()
0
```

Aggregation

- db.collection.aggregate() calculates aggregate values for the data in a collection.

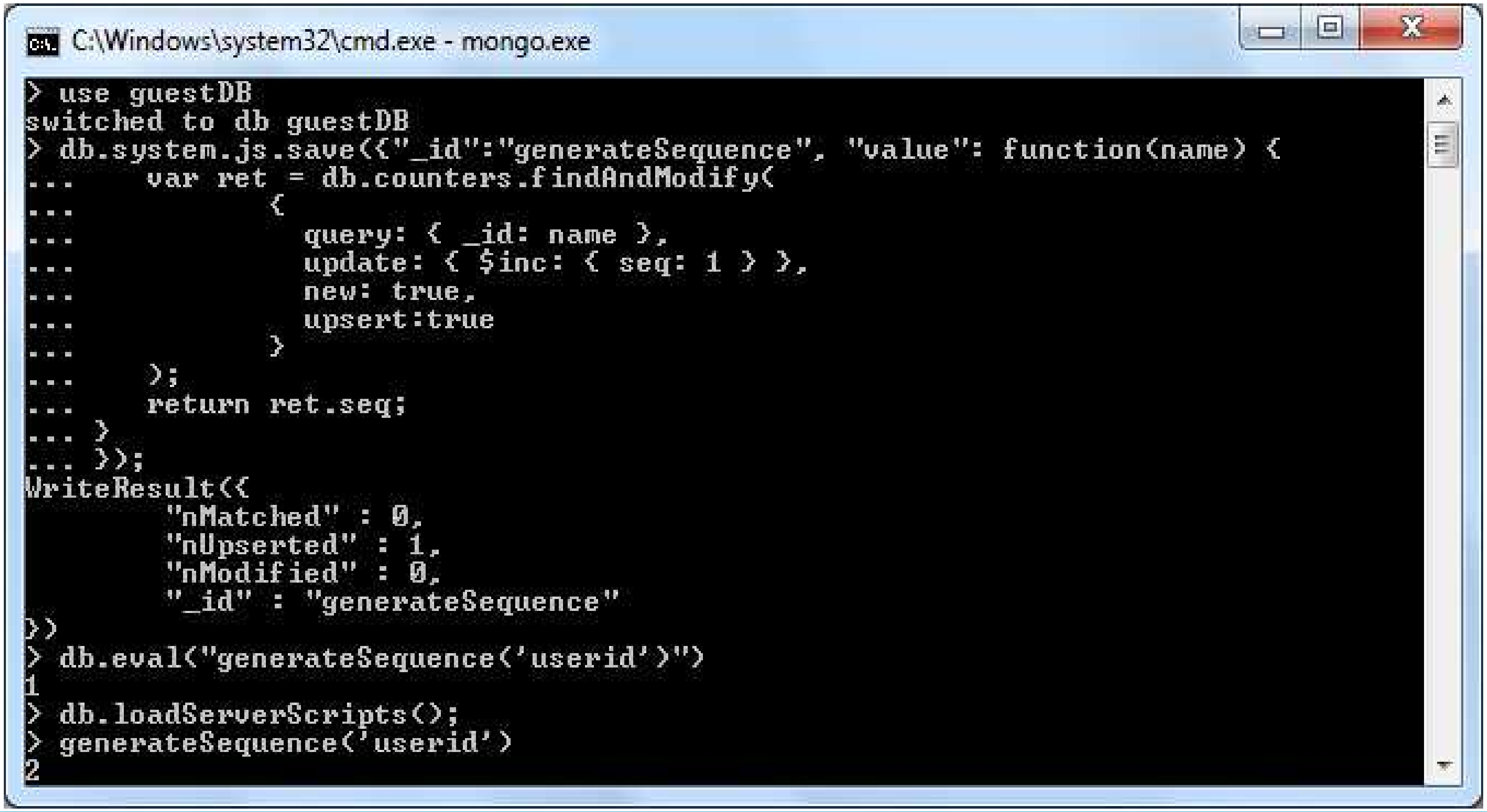


```
>
> db.employees.aggregate(
... {
...     $group: { _id : "$location", employeeCount : { $sum : 1 }}
... },
... {
...     $sort: { employeeCount : -1 }
... }
... );
{ "_id" : "Pune", "employeeCount" : 24 }
{ "_id" : "Mumbai", "employeeCount" : 23 }
{ "_id" : "Bangalore", "employeeCount" : 21 }
{ "_id" : "Chennai", "employeeCount" : 7 }
>
```

```
>
> db.employees.aggregate(
... {
...      $match : { location : "Bangalore" }
... },
... {
...      $group : { _id : "$location", employeeCount : { $sum : 1 } }
... }
... );
{ "_id" : "Bangalore", "employeeCount" : 21 }
>
```

Stored JavaScript

Do not store application logic in the database. There are performance limitations to running JavaScript inside of MongoDB.



C:\Windows\system32\cmd.exe - mongo.exe

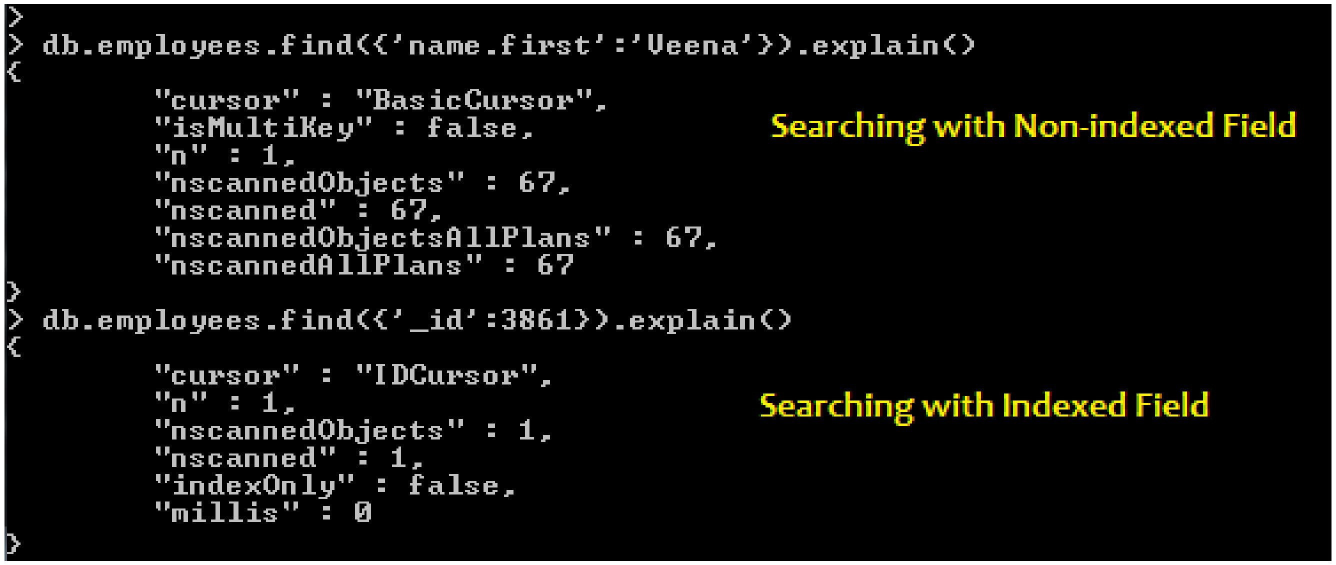
```
> use guestDB
switched to db guestDB
> db.system.js.save({"_id":"generateSequence", "value": function(name) {
...     var ret = db.counters.findAndModify(
...             {
...               query: { _id: name },
...               update: { $inc: { seq: 1 } },
...               new: true,
...               upsert:true
...             }
...     );
...     return ret.seq;
... }
... });
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : "generateSequence"
})
> db.eval("generateSequence('userid')")
1
> db.loadServerScripts();
> generateSequence('userid')
2
```

Demo

- Stored JavaScript

Indexing

- Indexing is an important part of database management.
- In MongoDB, if we do a query with non-indexed field, it uses "BasicCursor". BasicCursor indicates a full collection scan where as "BtreeCursor" indicates that the query used is an index field.
- explain() method returns a document that describes the process used to return the query results. MongoDB stores its indexes in system.indexes collection. We can view the indexes of DB using *db.system.indexes.find()*



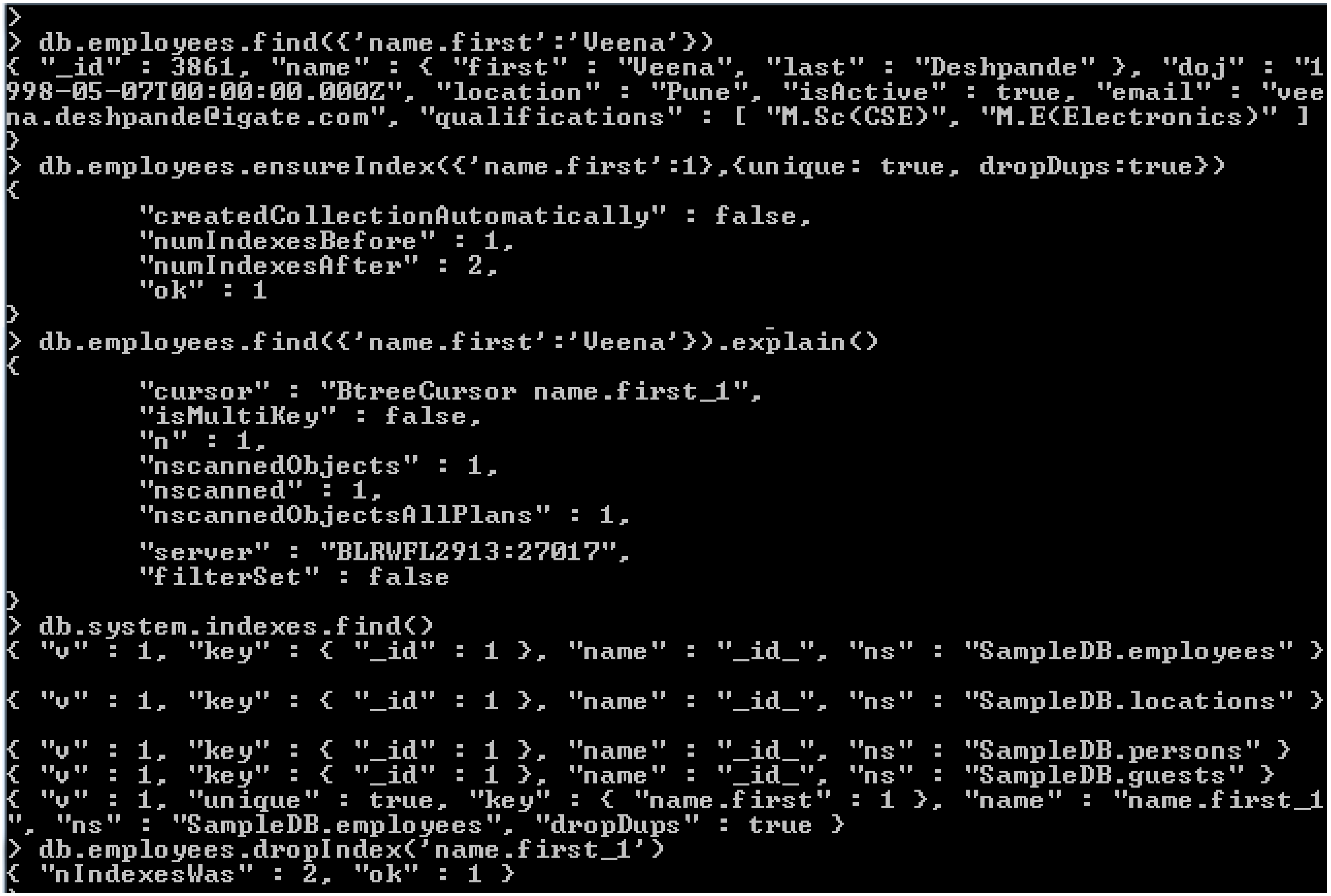
```
>
> db.employees.find({'name.first':'Veena'}).explain()
{
        "cursor" : "BasicCursor",
        "isMultiKey" : false,
        "n" : 1,
        "nscannedObjects" : 67,
        "nscanned" : 67,
        "nscannedObjectsAllPlans" : 67,
        "nscannedAllPlans" : 67
}
> db.employees.find({'_id':3861}).explain()
{
        "cursor" : "IDCursor",
        "n" : 1,
        "nscannedObjects" : 1,
        "nscanned" : 1,
        "indexOnly" : false,
        "millis" : 0
}
```

Searching with Non-indexed Field
Searching with Indexed Field

Indexing

- To create an index on the specified field if the index does not already exist.
  - db.<collection>.ensureIndex({<field>:1(asc)/-1(desc)})
- To create a unique index so that the collection will not accept insertion of documents where the index key or keys match an existing value in the index.
  - db.<collection>.ensureIndex({<field>:1(asc)/-1(desc)},{unique: true})
- To creates a unique index on a field that may have duplicates
  - db.<collection>.ensureIndex({<field>:1(asc)/-1(desc)},{unique: true, dropDups:true})
- To create an Index on a Multiple Fields
  - db.collection.ensureIndex({<field1>:1(asc)/-1(desc), <field2>:1(asc)/-1(desc) })
- To create index which only references the documents with specified field
  - db.employees.ensureIndex({nonexistfield:1},{sparse:true});
- To drop index
  - db.employees.dropIndex('<indexname>')

Indexing



```
>
> db.employees.find({'name.first':'Veena'})
{ "_id" : 3861, "name" : { "first" : "Veena", "last" : "Deshpande" }, "doj" : "1
998-05-07T00:00:00.000Z", "location" : "Pune", "isActive" : true, "email" : "vee
na.deshpande@igate.com", "qualifications" : [ "M.Sc(CSE)", "M.E(Electronics)" ]
}
> db.employees.ensureIndex({'name.first':1},{unique: true, dropDups:true})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
> db.employees.find({'name.first':'Veena'}).explain()
{
        "cursor" : "BtreeCursor name.first_1",
        "isMultiKey" : false,
        "n" : 1,
        "nscannedObjects" : 1,
        "nscanned" : 1,
        "nscannedObjectsAllPlans" : 1,

        "server" : "BLRWFL2913:27017",
        "filterSet" : false
}
> db.system.indexes.find()
{ "v" : 1, "key" : { "_id" : 1 }, "name" : "_id_", "ns" : "SampleDB.employees" }

{ "v" : 1, "key" : { "_id" : 1 }, "name" : "_id_", "ns" : "SampleDB.locations" }

{ "v" : 1, "key" : { "_id" : 1 }, "name" : "_id_", "ns" : "SampleDB.persons" }
{ "v" : 1, "key" : { "_id" : 1 }, "name" : "_id_", "ns" : "SampleDB.guests" }
{ "v" : 1, "unique" : true, "key" : { "name.first" : 1 }, "name" : "name.first_1
", "ns" : "SampleDB.employees", "dropDups" : true }
> db.employees.dropIndex('name.first_1')
{ "nIndexesWas" : 2, "ok" : 1 }
```

Demo

- Working with Indexing