Instructor Notes:

Add instructor notes here.

Instructor Notes:

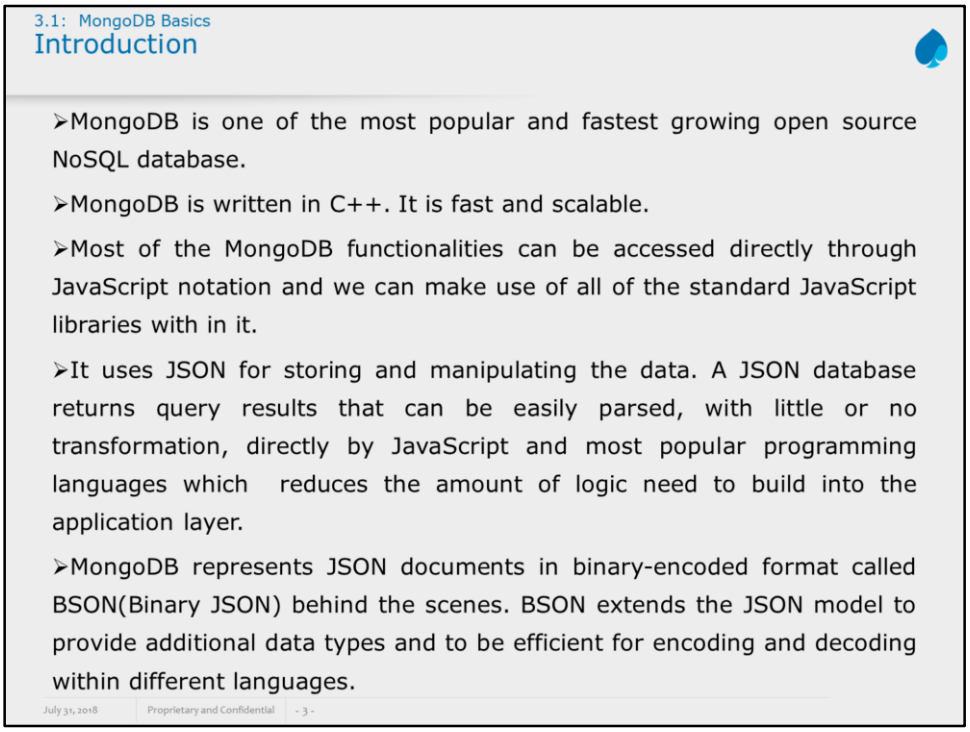
Add instructor notes here.

Lesson Objectives

Mongoose Library
Introduction to Mongoose Library
Connecting MongoDB using Mongoose Library
Object Modeling
RDBMS Schema Vs Mongoose Schema
Simple Schema
Complex Schema
Validations in Model
Finding documents
Query API

Instructor Notes:

Add instructor notes here.



The MongoDB BSON implementation is lightweight, fast and highly traversable.

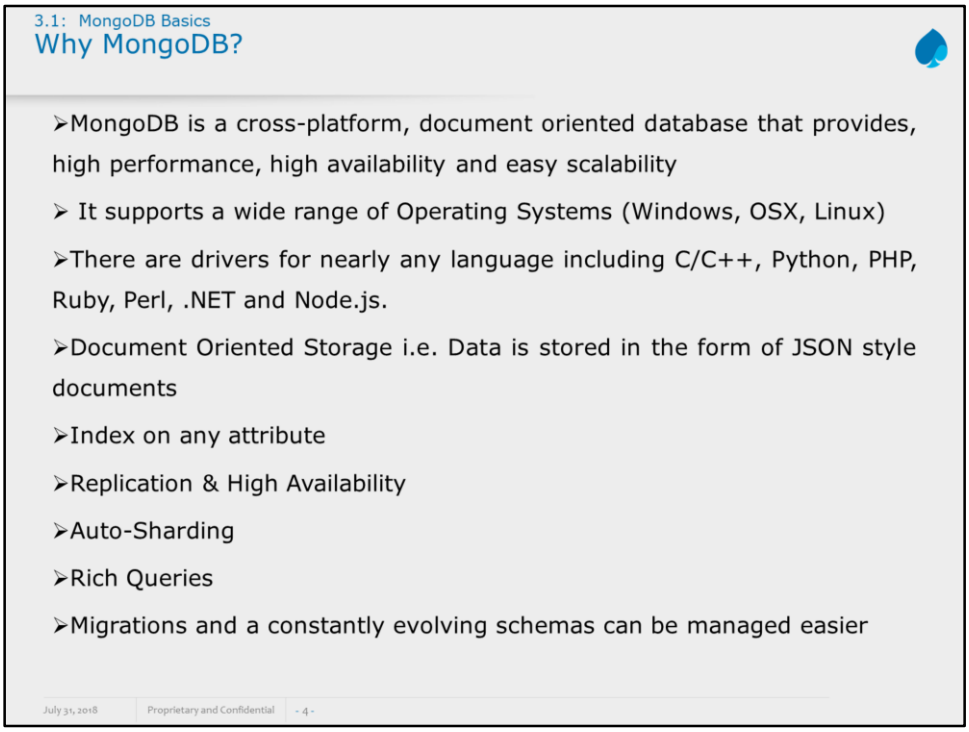
BSON is the binary encoding of JSON-like documents that MongoDB uses when storing documents in collections.

It adds support for data types like Date and binary that aren't supported in JSON.

In practice, you don't have to know much about BSON when working with MongoDB, you just need to use its data types when constructing documents.

Instructor Notes:

Add instructor notes here.



Indexing :

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and require the mongod to process a large volume of data. Indexes are special data structures, that store a small portion of the data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in index.

Replication :

Replication is the process of synchronizing data across multiple servers. Replication provides redundancy and increases data availability with multiple copies of data on different database servers, replication protects a database from the loss of a single server. Replication also allows us to recover from hardware failure and service interruptions. With additional copies of the data, you can dedicate one to disaster recovery, reporting, or backup

Auto-Sharding :

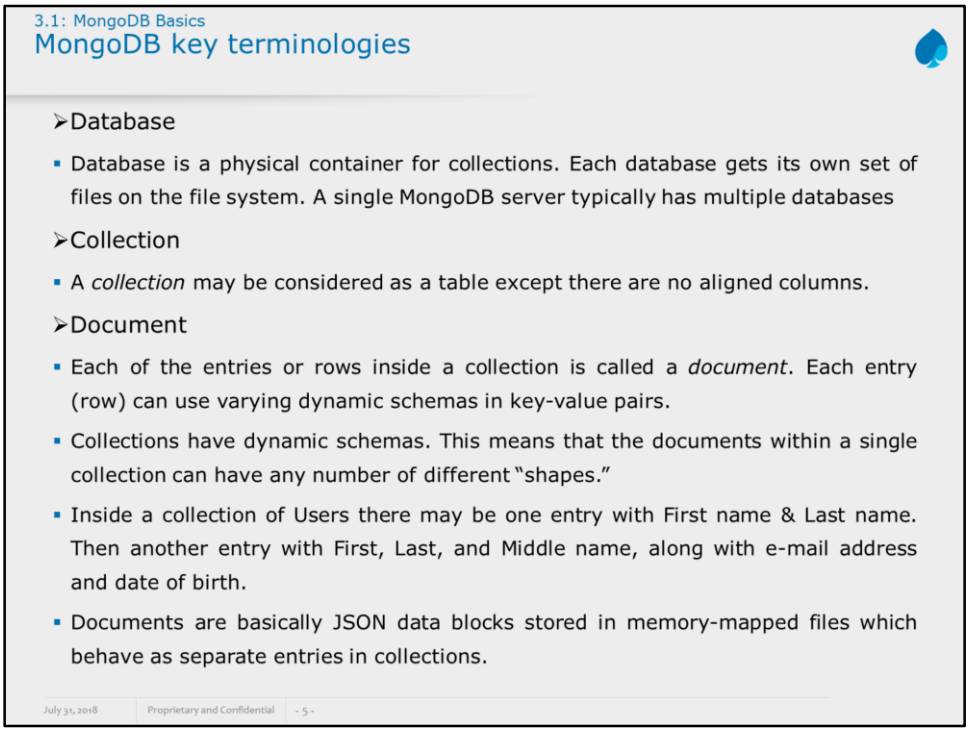
Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth. As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves the problem with horizontal scaling. With sharding, we can add more machines to support data growth and the demands of read and write operations.

Rich Queries :

- Arbitrary sorting on one or more fields, ascending or descending.
- Projection, i.e. retrieving only specified fields from each document.
- skip() and limit() to easily implement pagination if desired.
- explain(), which returns a wealth of information including full details on query path analysis, document and index counts, and estimated vs. actual processing time.

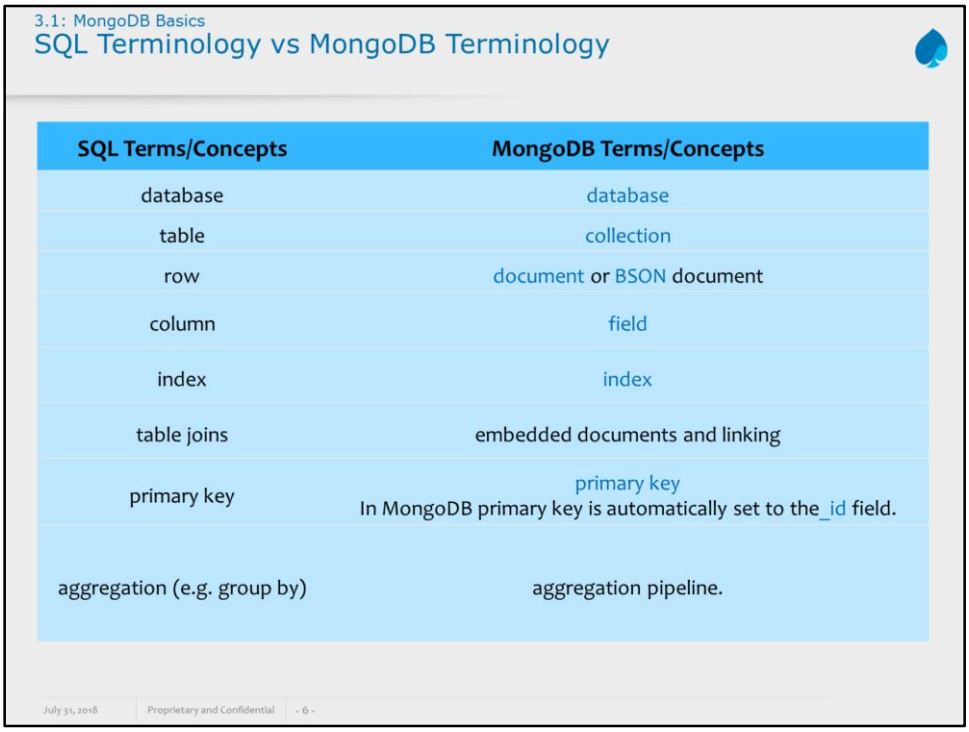
Instructor Notes:

Add instructor notes here.



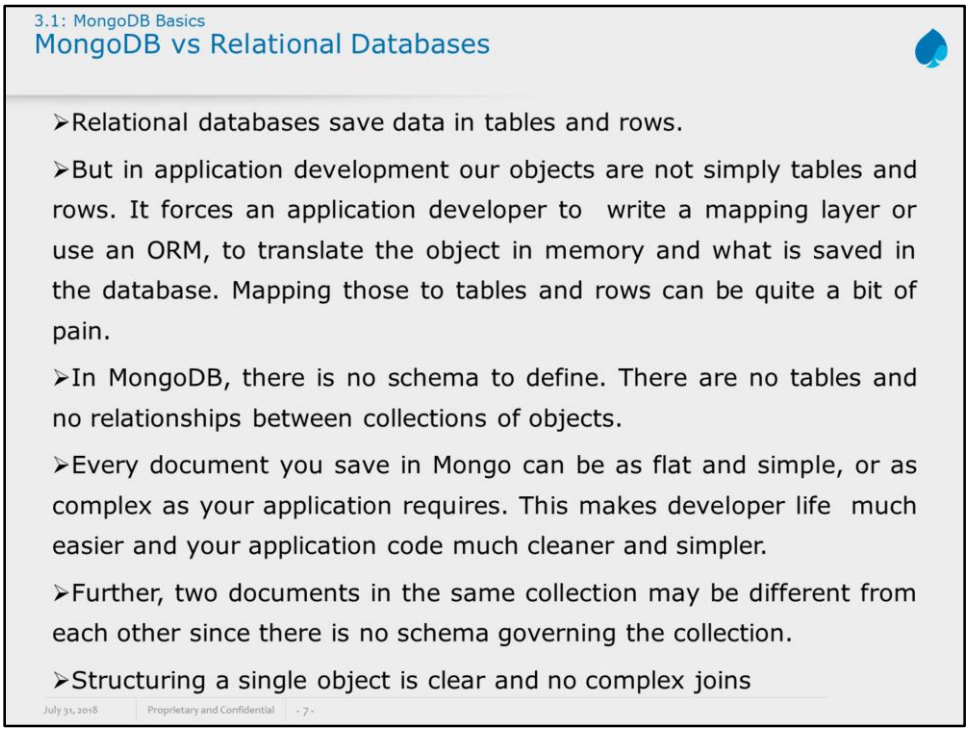
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

3.2: Getting started with MongoDB

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)

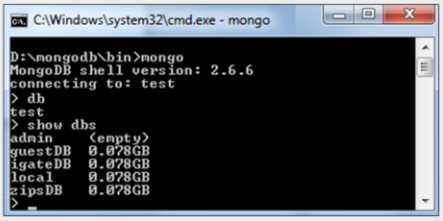
- **Go to <mongodb installation dir> :** *cd <mongodb installation dir>*
- **To start mongo goto bin directory and type mongo ex:-** *D:\mongodb\bin>mongo*

➢To check your currently selected database, use the command *db*. Default(test)

- *db*

➢To list the databases, use the command show dbs.

- *show dbs*



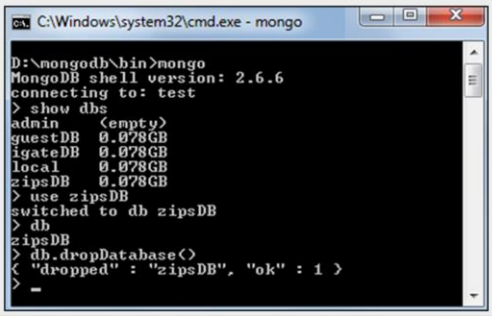
Instructor Notes:

Add instructor notes here.

3.2: Getting started with MongoDB

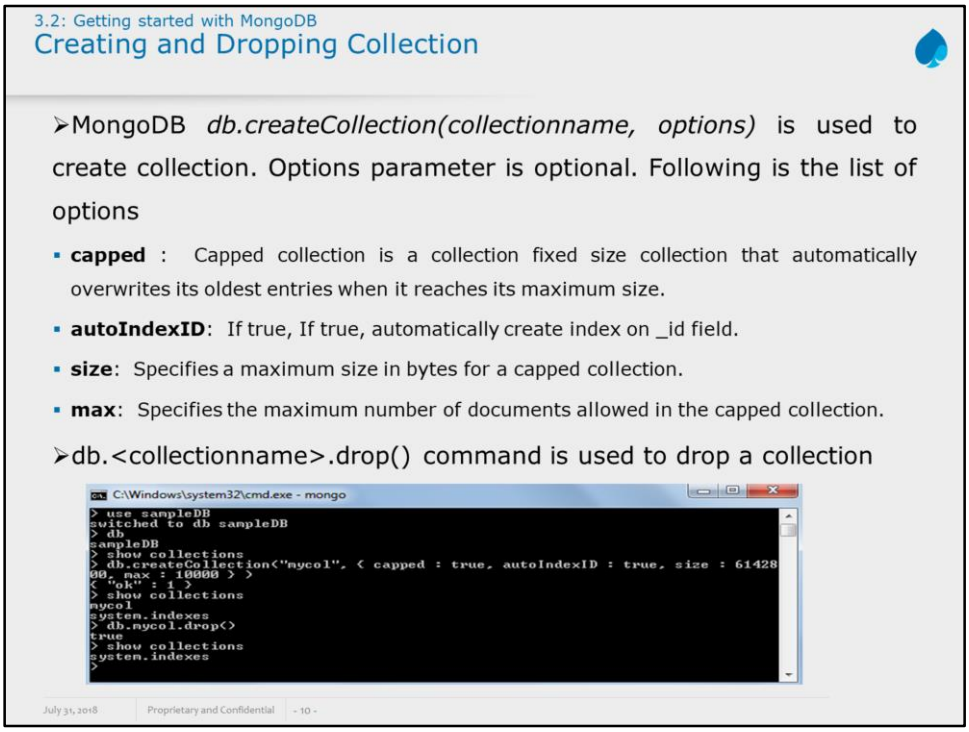
Creating and Dropping Database

- 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.



Instructor Notes:

Add instructor notes here.



MongoDB supports many datatypes whose list is given below:

String : This is most commonly used datatype to store the data. String in mongodb must be UTF-8 valid.

Integer : This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.

Boolean : This type is used to store a boolean (true/ false) value.

Double : This type is used to store floating point values.

Min/ Max keys : This type is used to compare a value against the lowest and highest BSON elements.

Arrays : This type is used to store arrays or list or multiple values into one key.

Timestamp : ctimestamp. This can be handy for recording when a document has been modified or added.

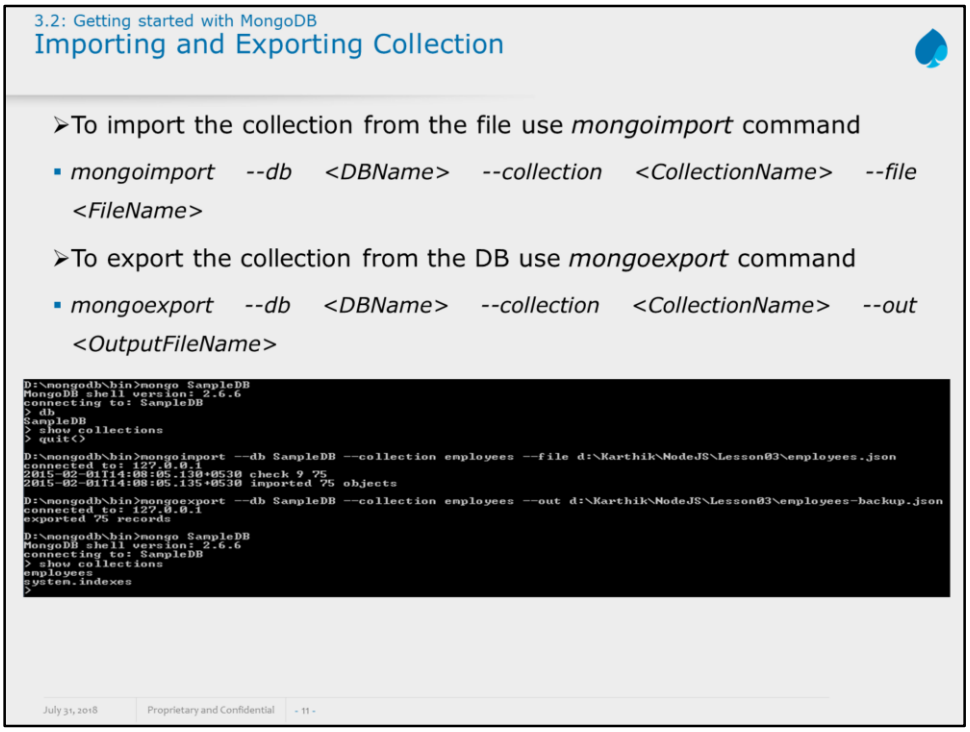
Object : This datatype is used for embedded documents.

Null : This type is used to store a Null value.

Symbol : This datatype is used identically to a string however, it's generally reserved for languages that use a specific symbol type.

Instructor Notes:

Add instructor notes here.



Date : This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.

Object ID : This datatype is used to store the document's ID.

Binary data : This datatype is used to store binay data.

Code : This datatype is used to store JavaScript code into document.

Regular expression : This datatype is used to store regular expression

Instructor Notes:

Add instructor notes here.

3.3: MongoDB Queries

Querying MongoDB Documents – find()

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

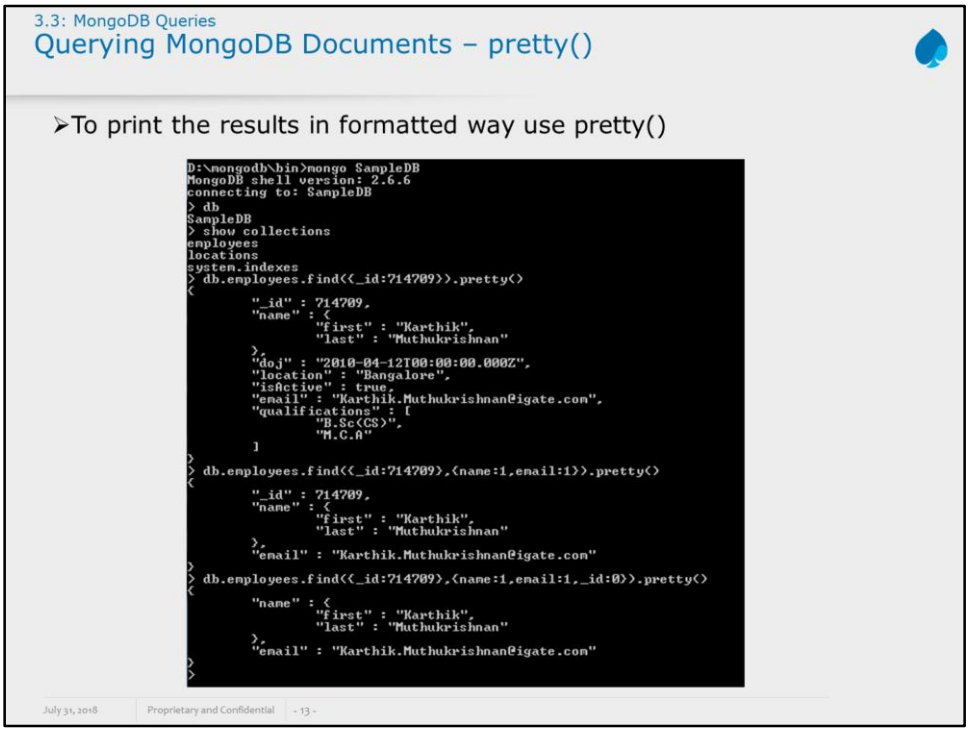
- ***db.COLLECTION_NAME.find()***
- **db. *COLLECTION_NAME*.find().pretty()** is used to display the results in a formatted way.
- **db. *COLLECTION_NAME*.count()** is used to count the number of documents in a collection.

```
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
>
```

Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

3.3: MongoDB Queries

Querying MongoDB Documents – findOne()

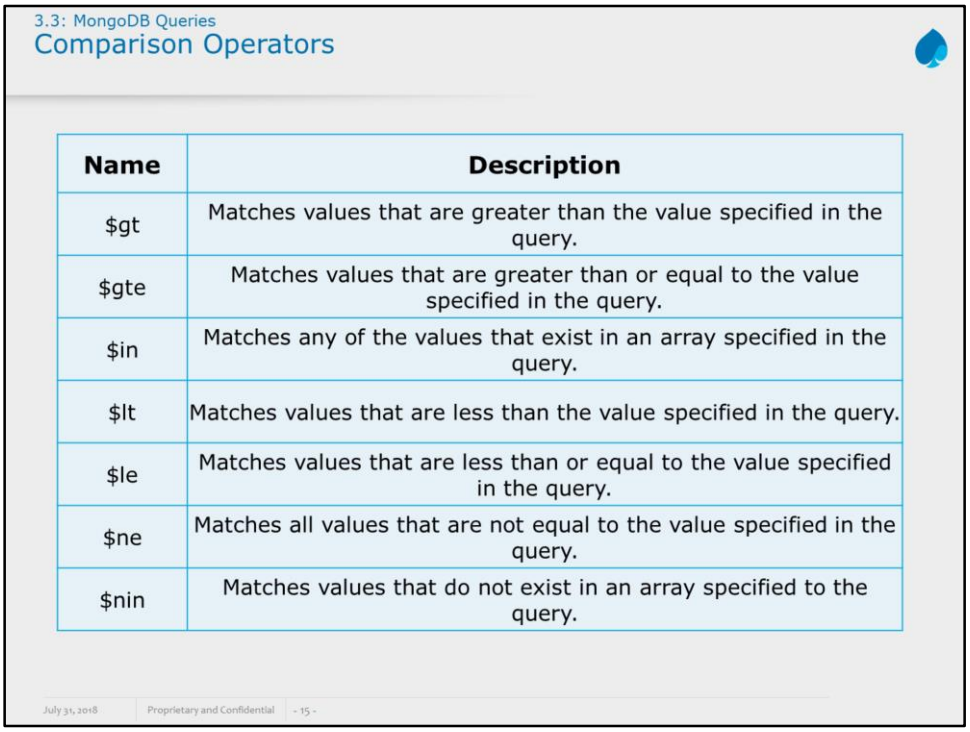
➢findOne() returns a single document, where as find() returns a cursor.

```
> 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
>
```

July 31, 2018

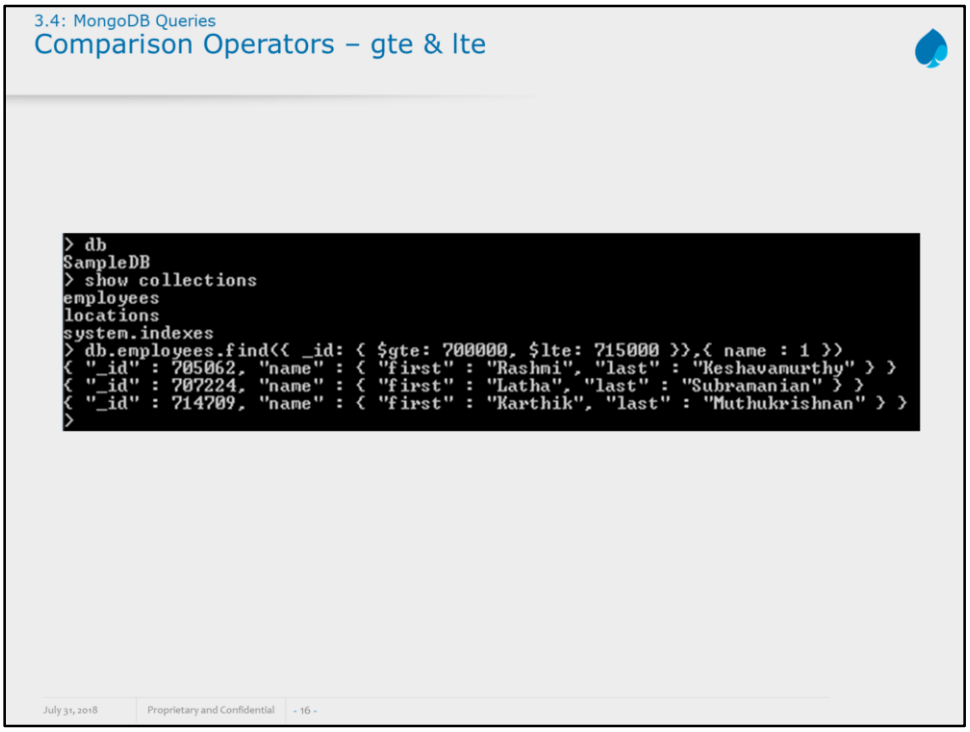
Instructor Notes:

Add instructor notes here.



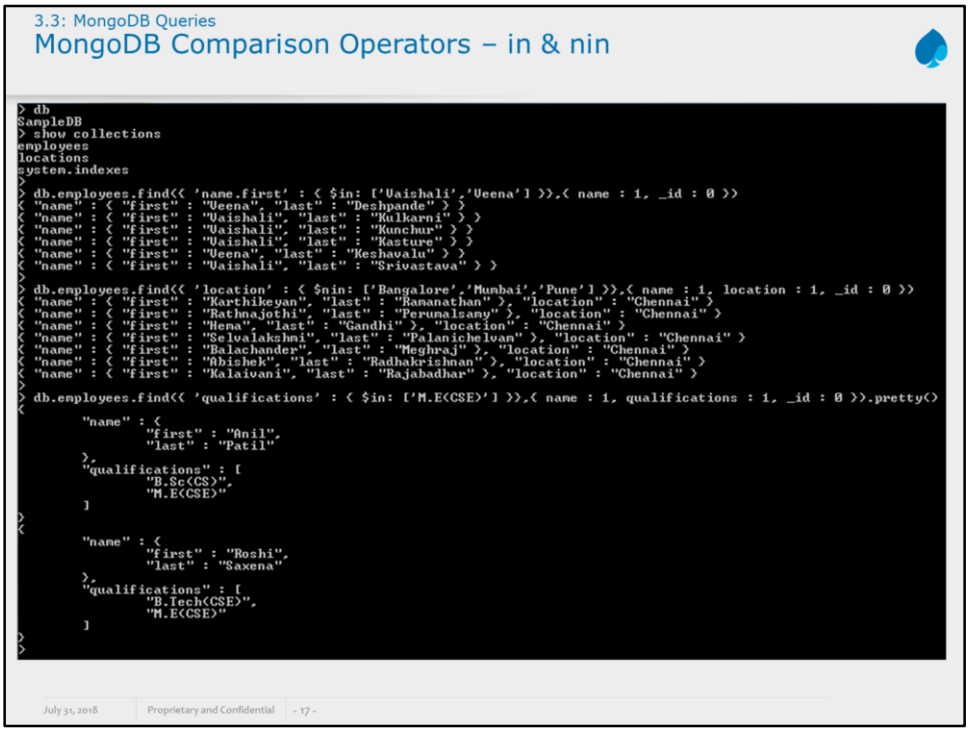
Instructor Notes:

Add instructor notes here.



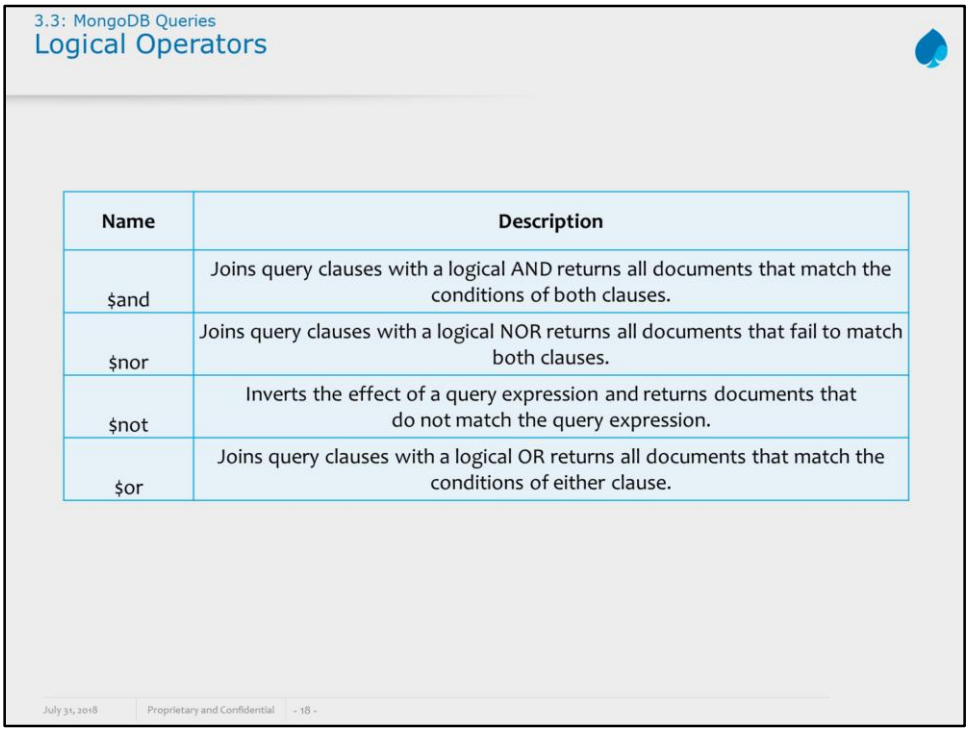
Instructor Notes:

Add instructor notes here.



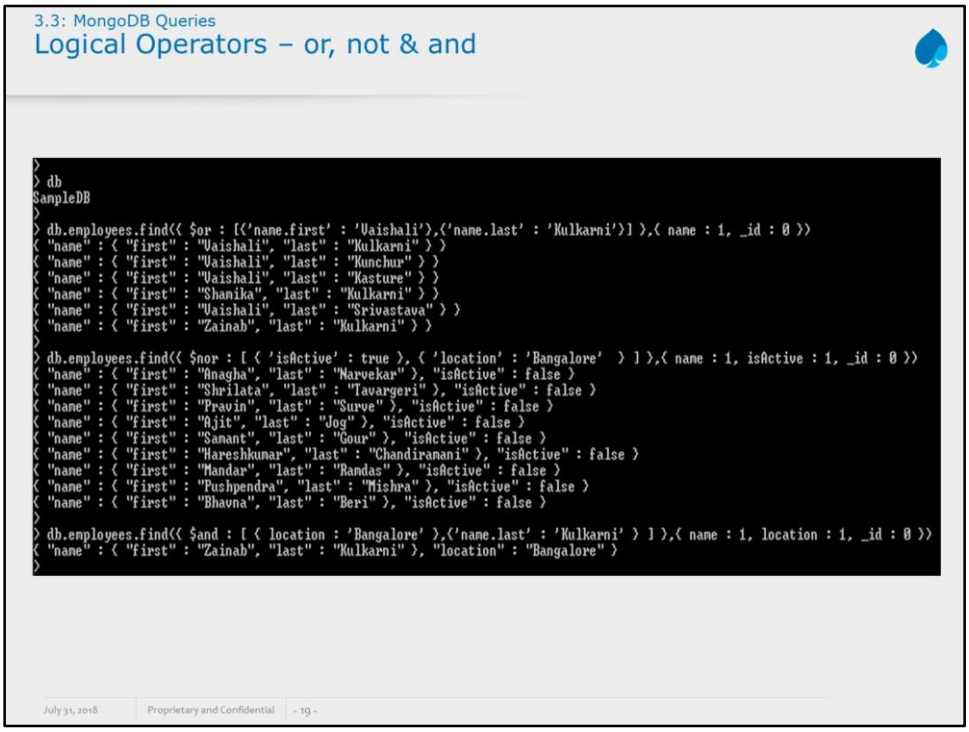
Instructor Notes:

Add instructor notes here.



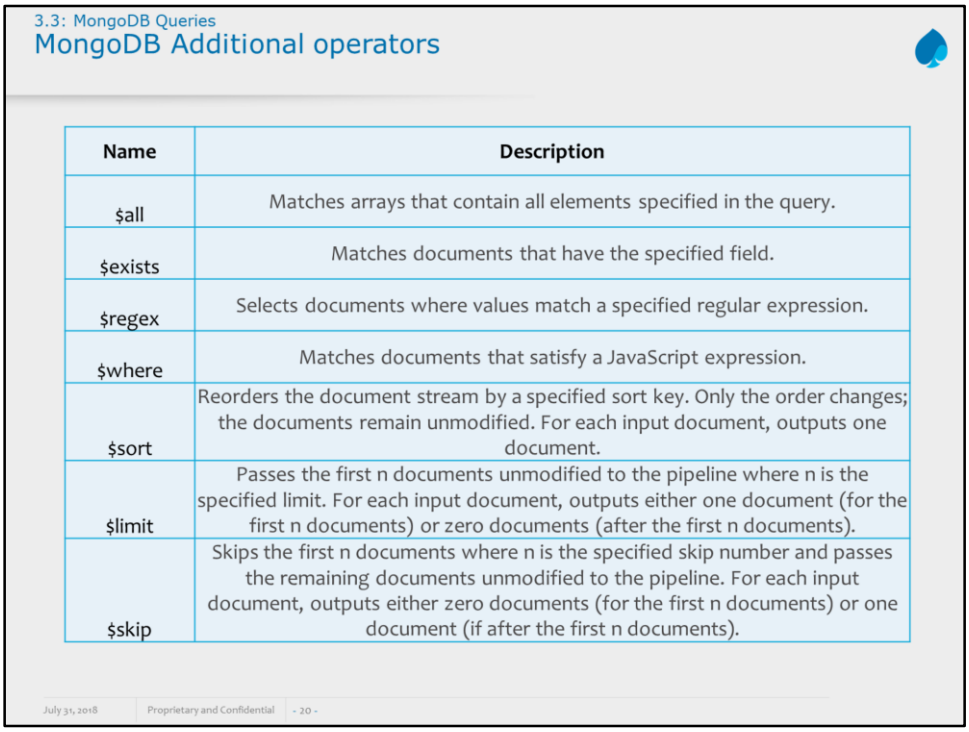
Instructor Notes:

Add instructor notes here.



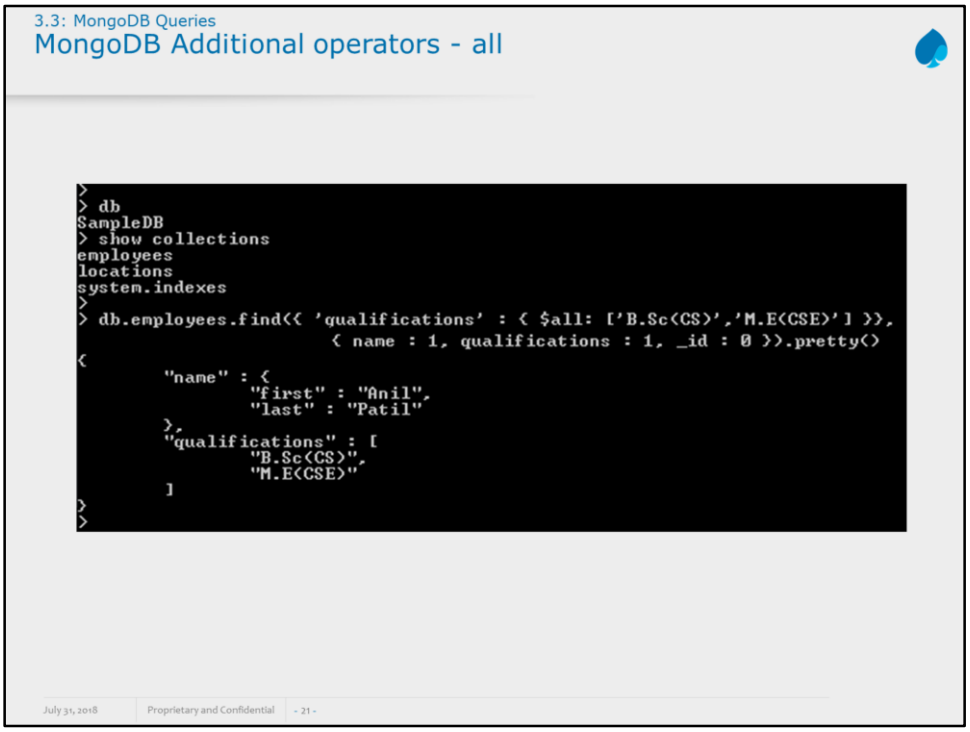
Instructor Notes:

Add instructor notes here.



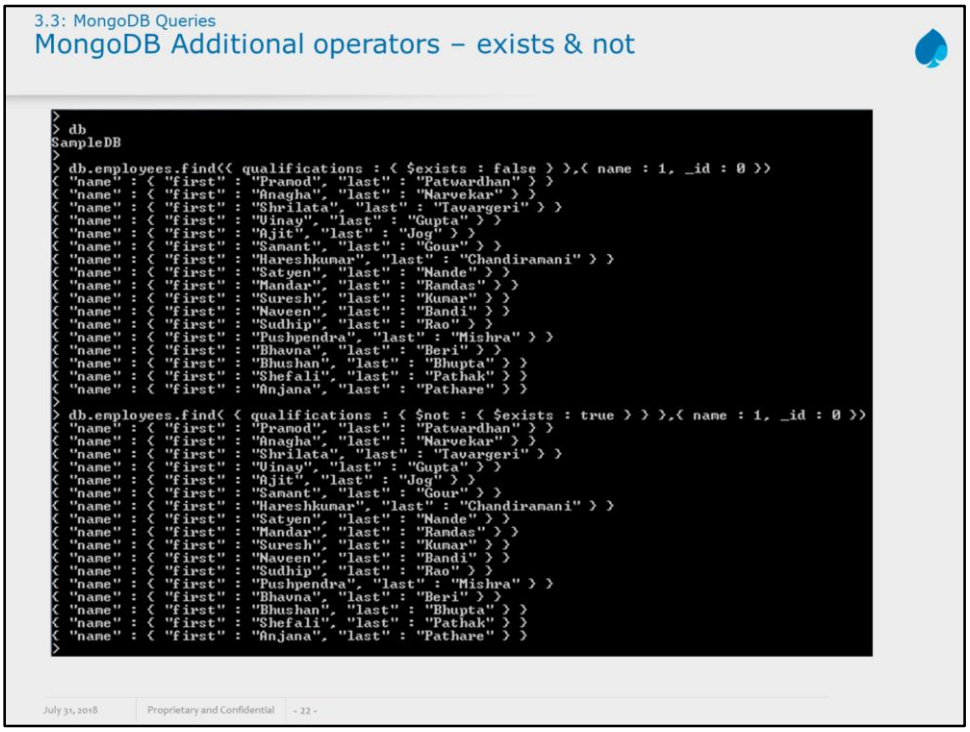
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

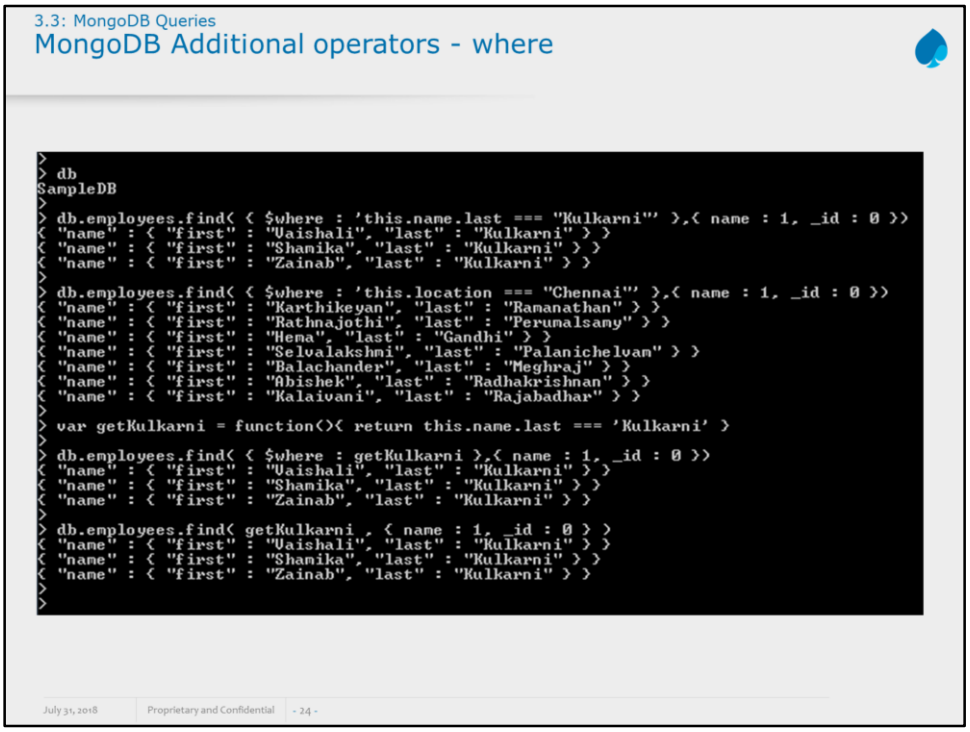


Instructor Notes:

Add instructor notes here.

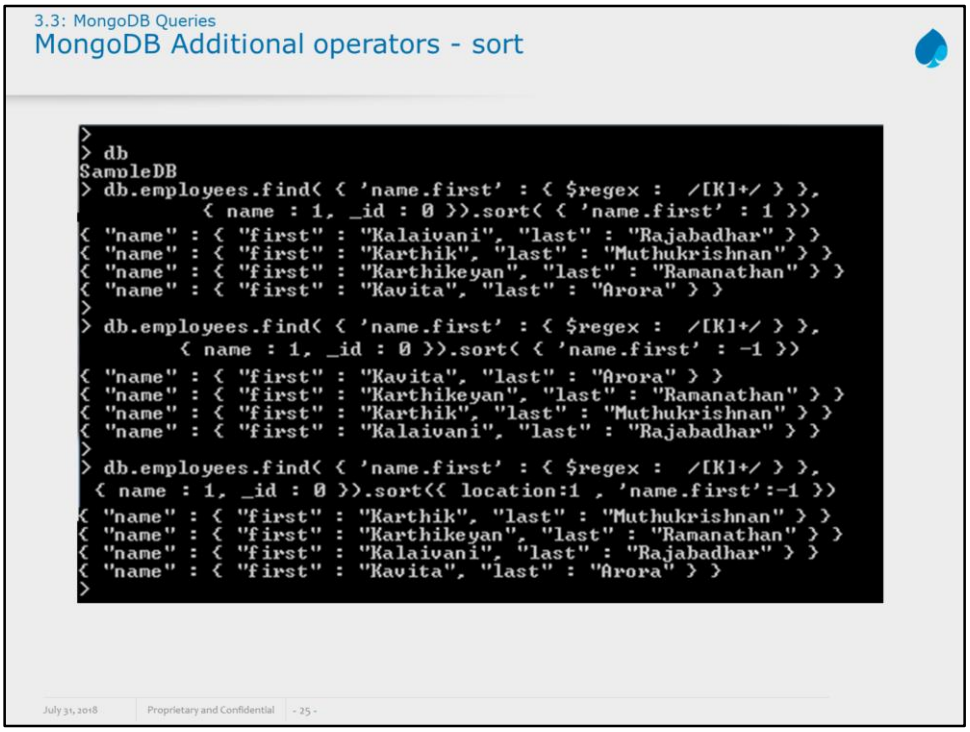
Instructor Notes:

Add instructor notes here.



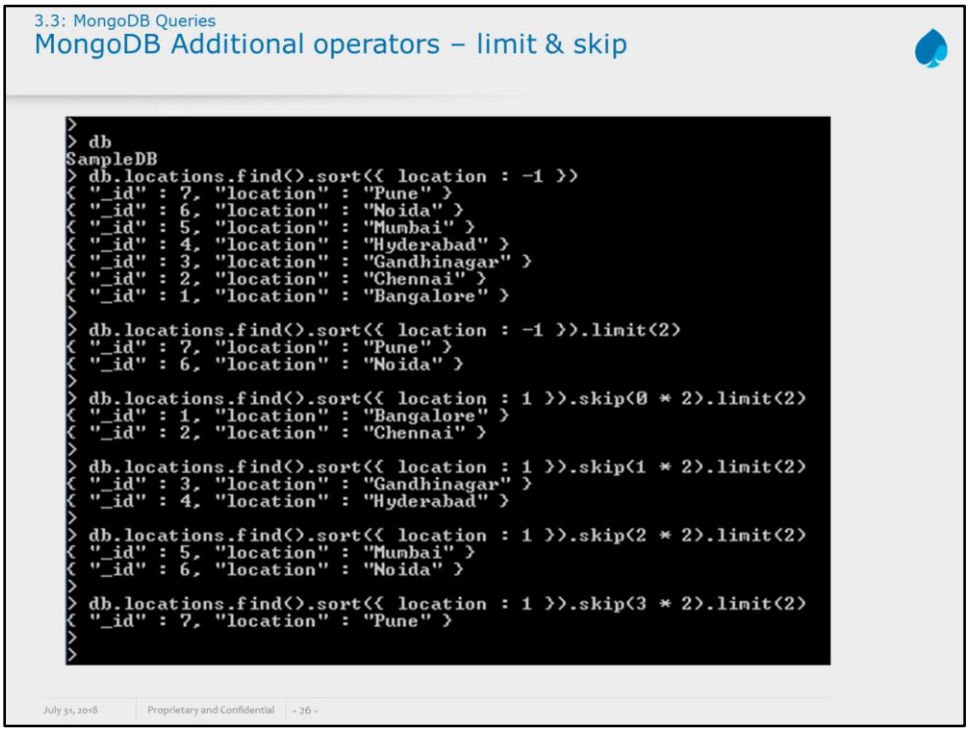
Instructor Notes:

Add instructor notes here.



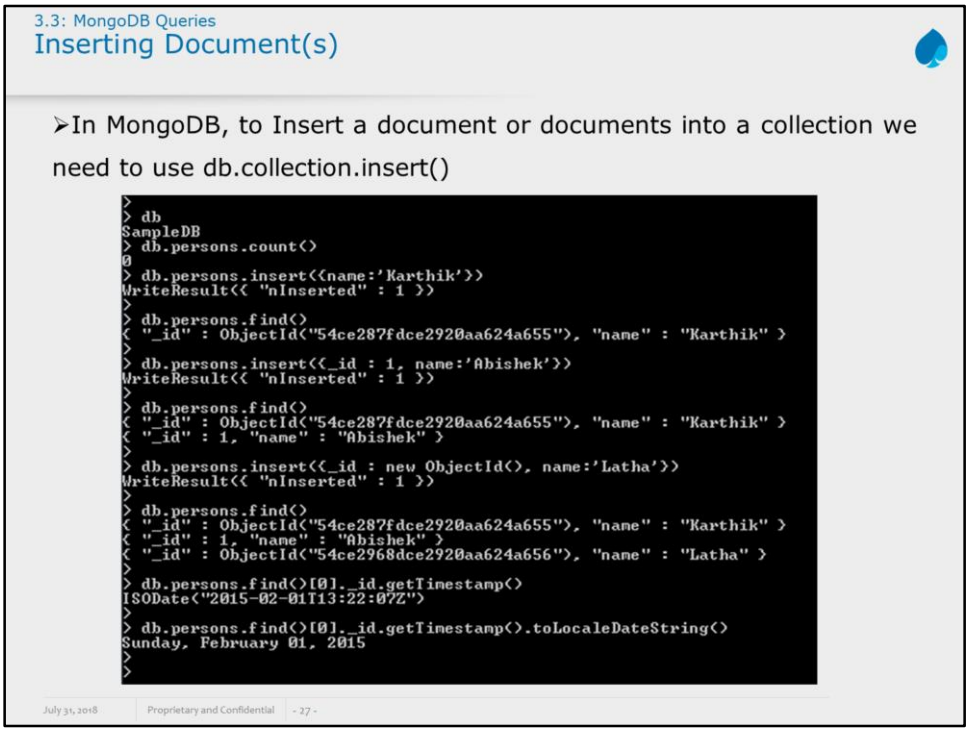
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

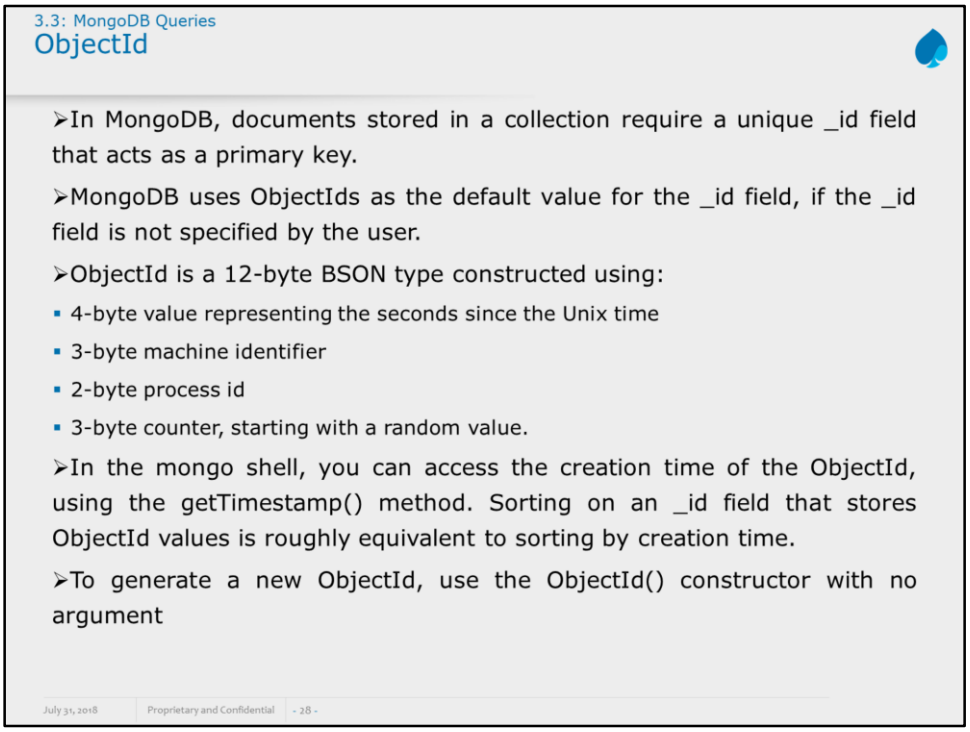


To Insert Multiple Documents

```
db.products.insert(
  [
    { _id: 11, item: "pencil", qty: 50, type: "no.2" },
    { item: "pen", qty: 20 },
    { item: "eraser", qty: 25 }
  ]
)
```

Instructor Notes:

Add instructor notes here.



To access the creation time

> db.guests.find()[0]._id.getTimestamp()
ISODate("2015-01-31T06:46:15Z")

To print the time in current Locale

> db.guests.find()[0]._id.getTimestamp().toLocaleDateString()
Saturday, January 31, 2015

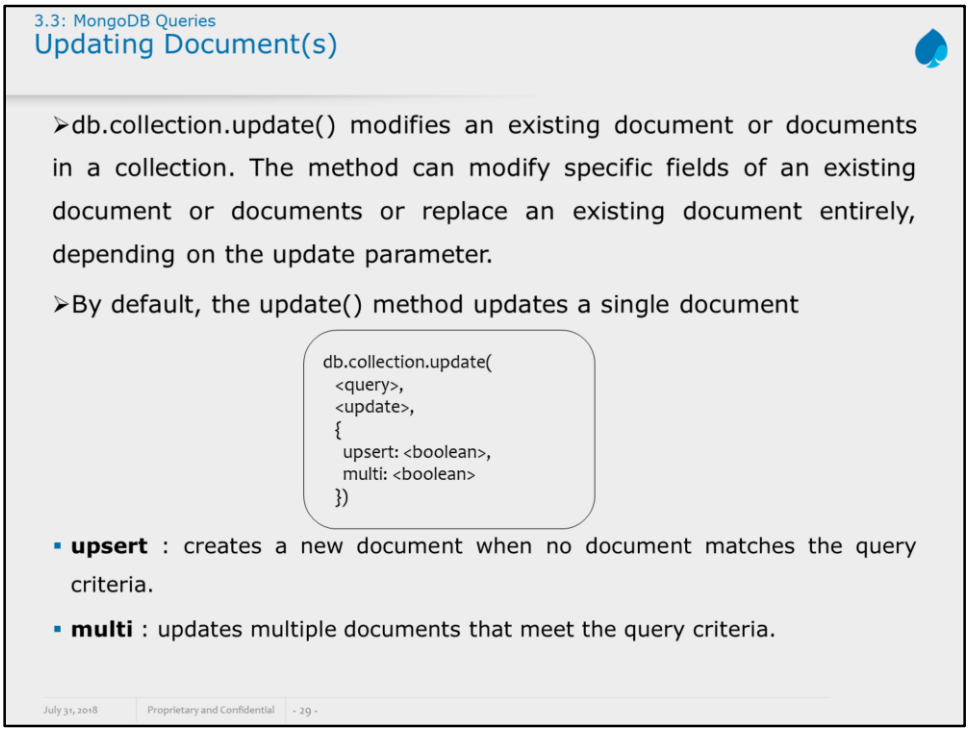
> db.guests.find()[0]._id.getTimestamp().toLocaleTimeString()
12:16:15

To create a new ObjectId

> new ObjectId()
ObjectId("54cc7fd2935e1001cd8a4d06")

Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

3.3: MongoDB Queries

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 }
>
```

Instructor Notes:

Add instructor notes here.

3.3: MongoDB Queries

Updating Document - save

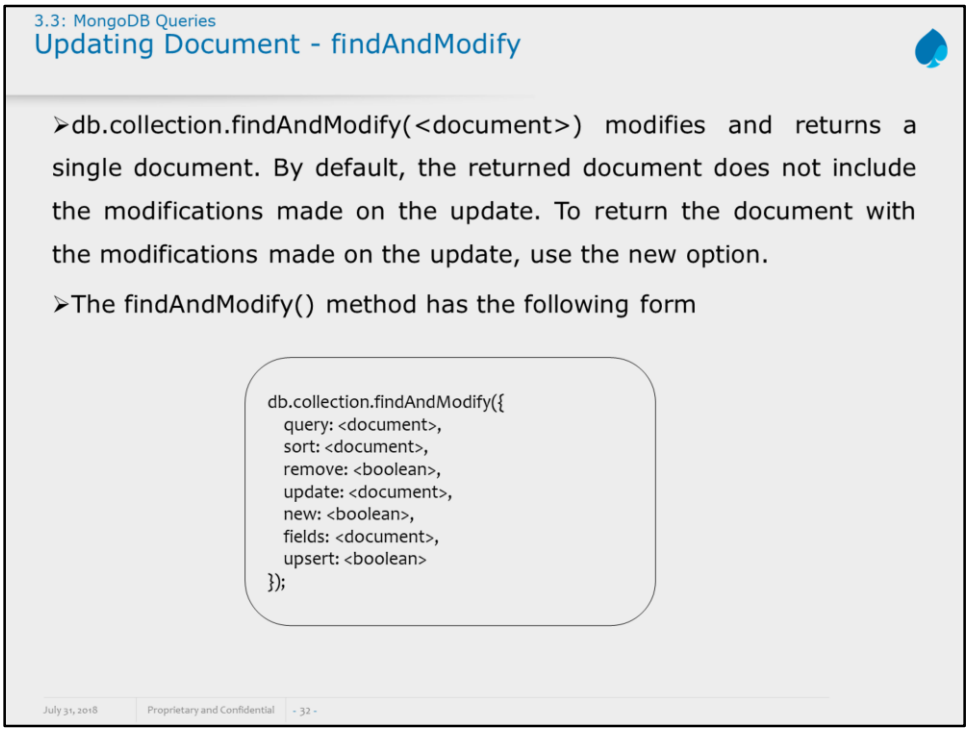
➢db.collection.save() updates an existing document (when _id is present already) or inserts a new document (with new ObjectId)

```
>
> 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"
}
```

July 31, 2018 Proprietary and Confidential - 31 -

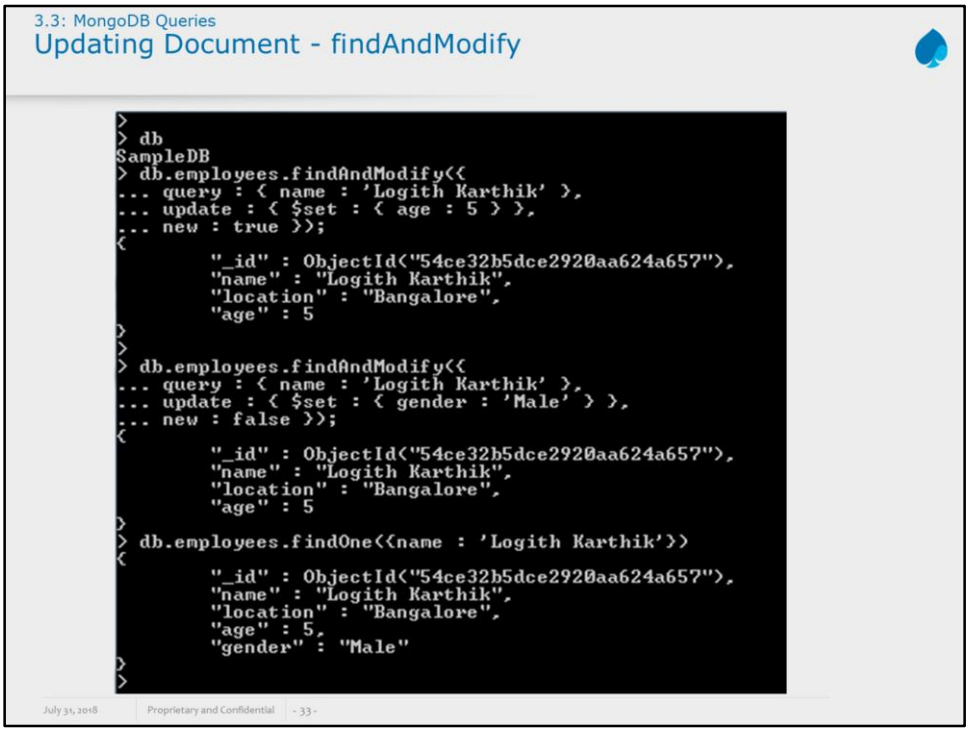
Instructor Notes:

Add instructor notes here.



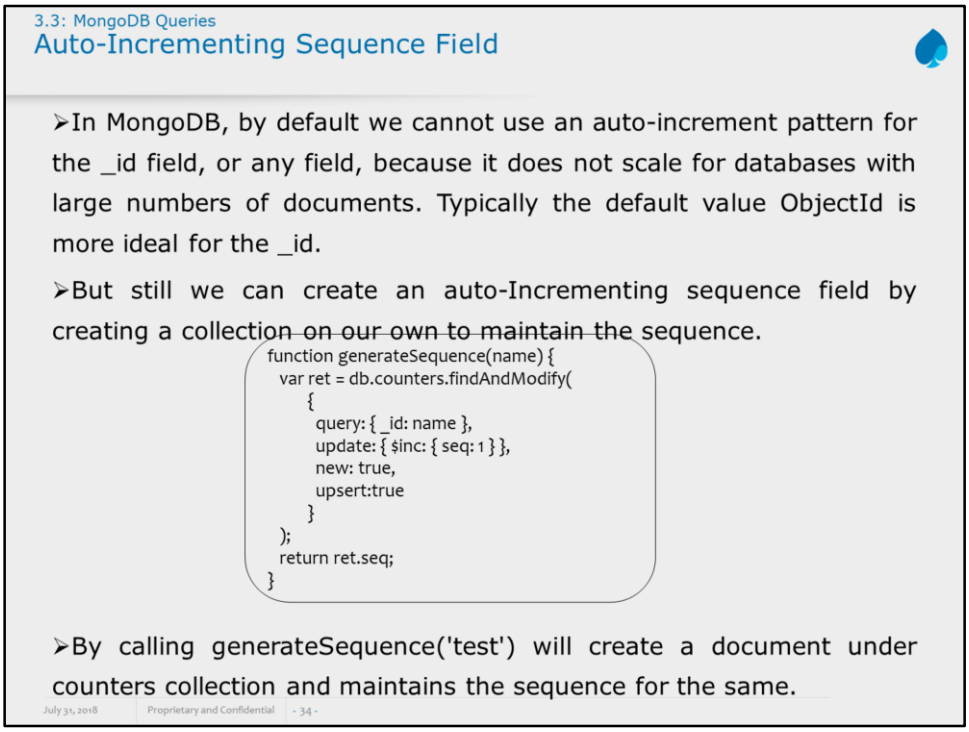
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

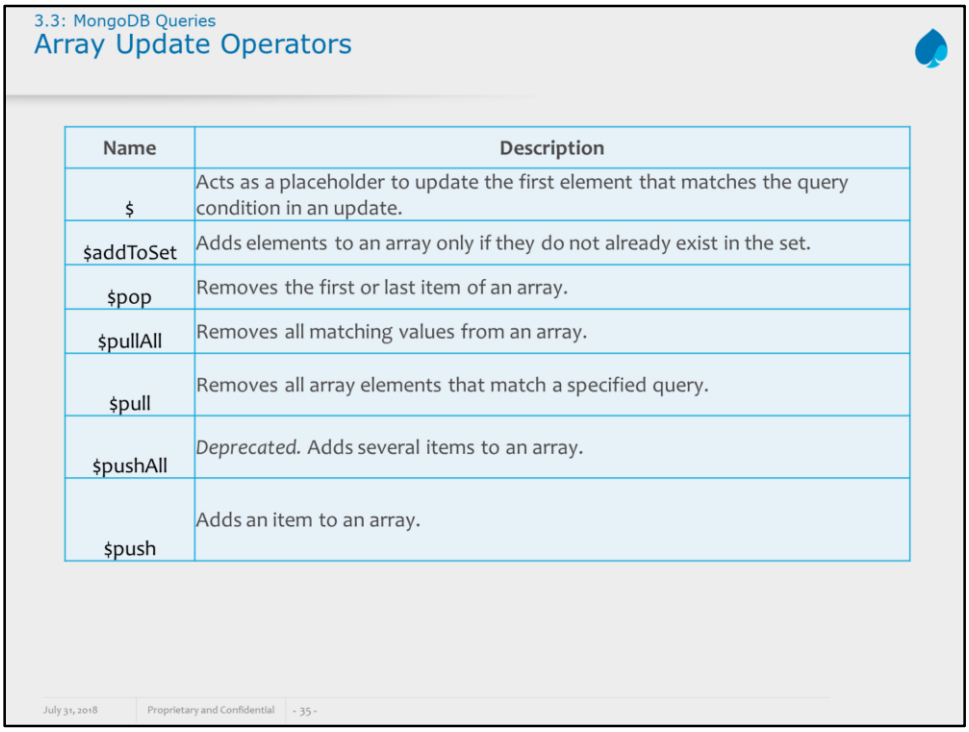



```
> function generateSequence(name) {
  var ret = db.counters.findAndModify(
        {
          query: { _id: name },
          update: { $inc: { seq: 1 } },
          new: true,
          upsert:true
        }
   );
   return ret.seq;
 }
> generateSequence('test')
1
> generateSequence('test')
2
> generateSequence('product')
1
> generateSequence('product')
2
> db.departments.insert({_id:generateSequence('department'),name:'IT'})
> db.departments.insert({_id:generateSequence('department'),name:'Training'})
> db.departments.find()
{ "_id" : 1, "name" : "IT" }
{ "_id" : 2, "name" : "Training" }

> db.counters.find()
{ "_id" : "test", "seq" : 2 }
{ "_id" : "product", "seq" : 2 }
{ "_id" : "department", "seq" : 2 }
```

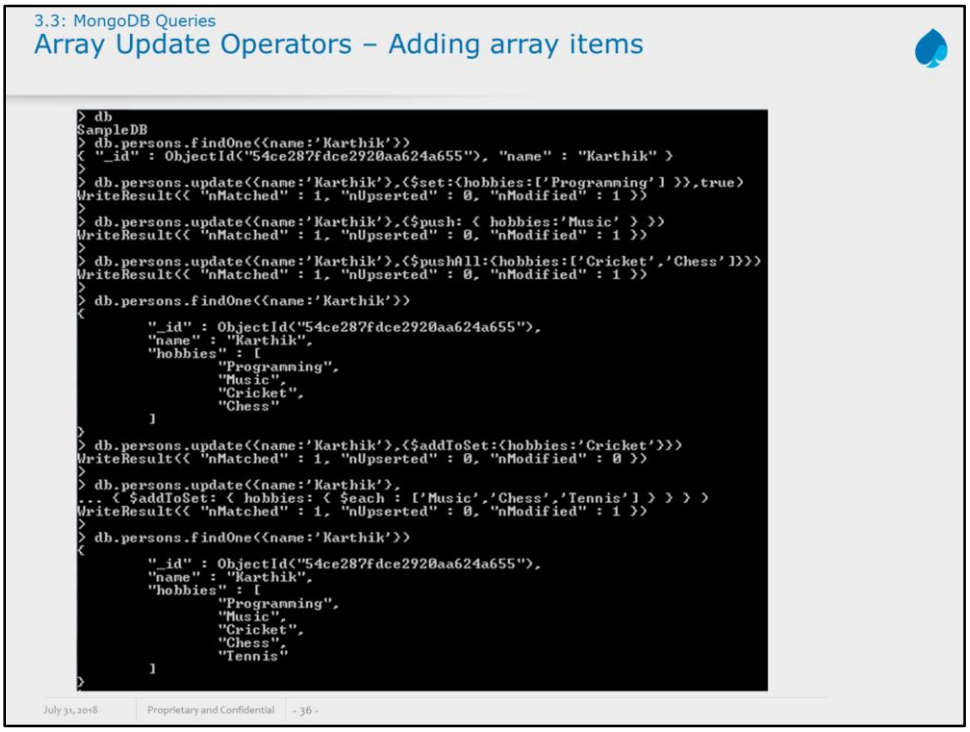
Instructor Notes:

Add instructor notes here.



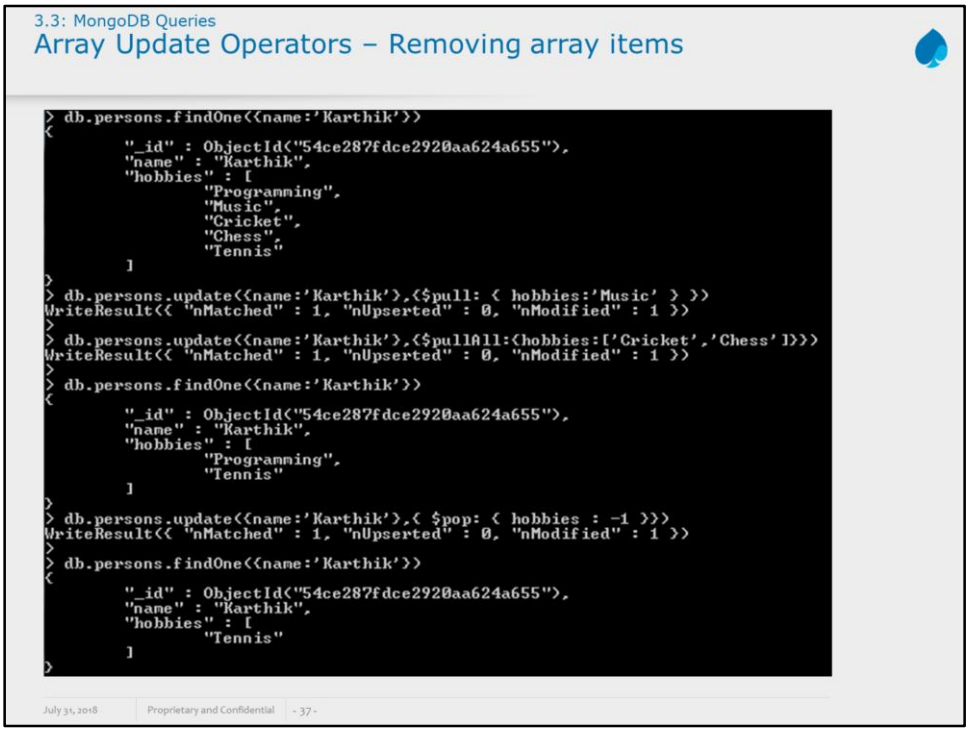
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

3.3: MongoDB Queries

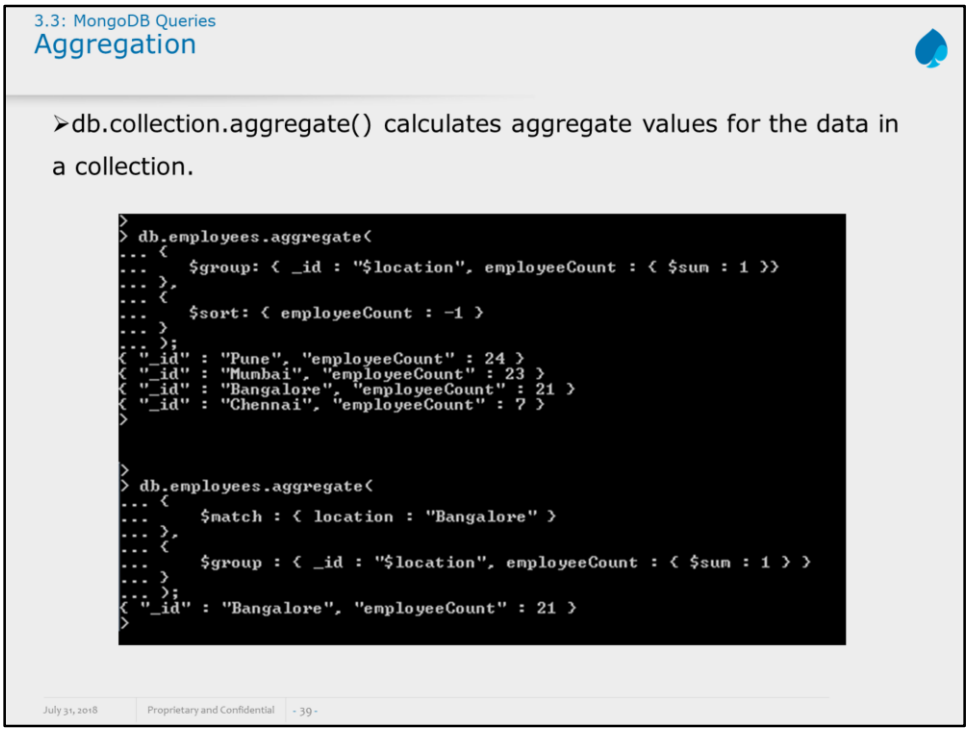
Renaming and Deleting fields and documents

- 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
```

Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

Instructor Notes:

Add instructor notes here.

3.3: MongoDB Queries

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
}
```

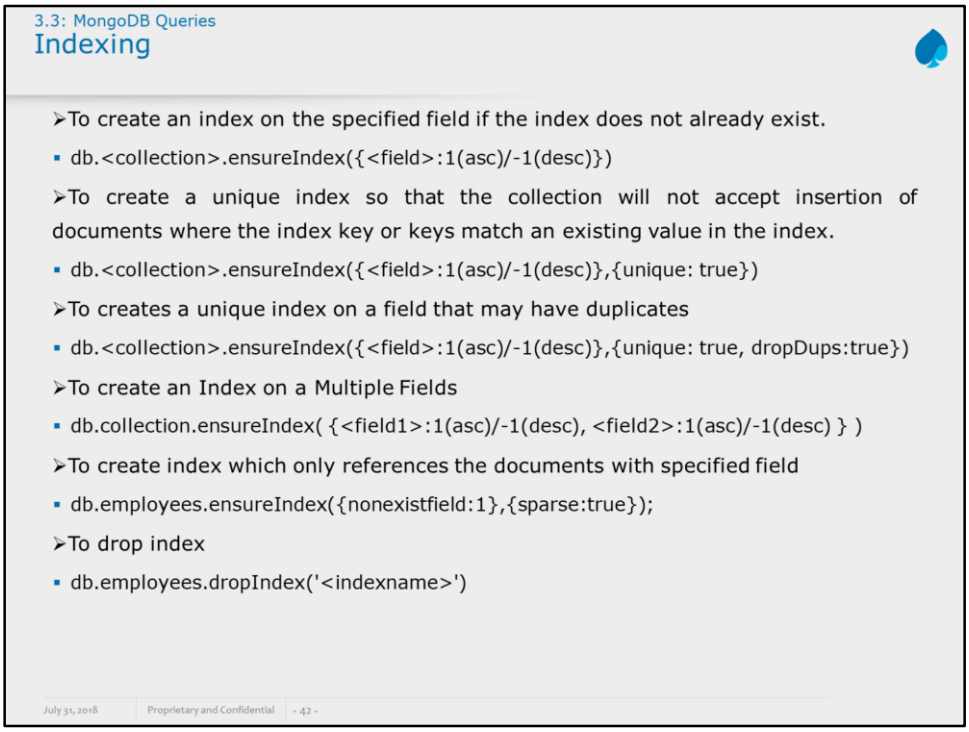
Searching with Non-indexed Field

Searching with Indexed Field

July 31, 2018 Proprietary and Confidential - 41 -

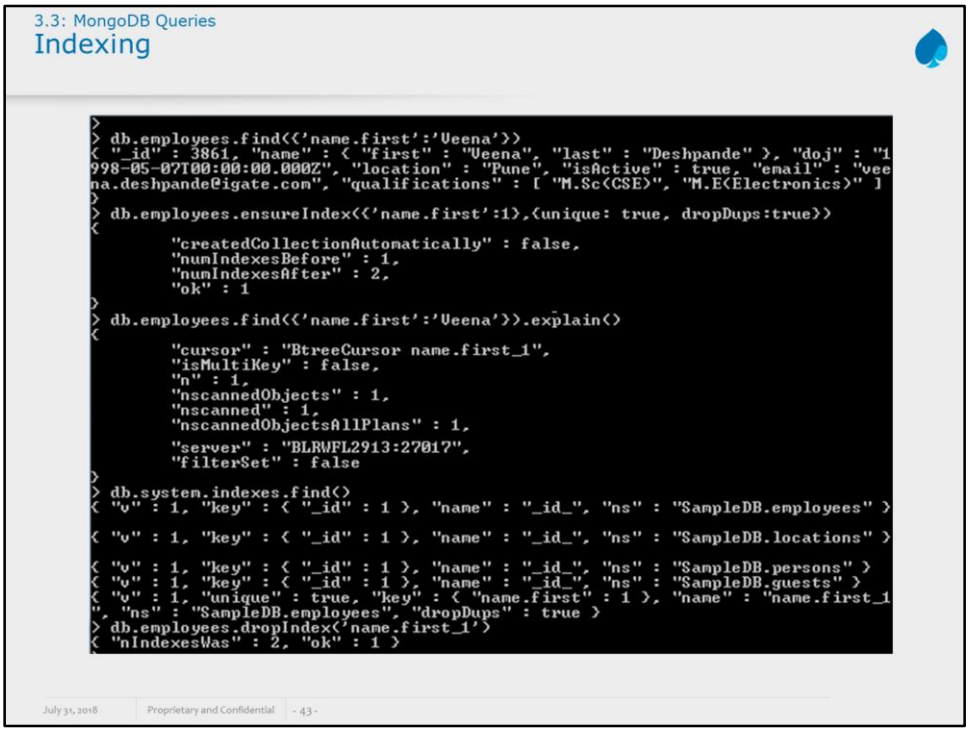
Instructor Notes:

Add instructor notes here.



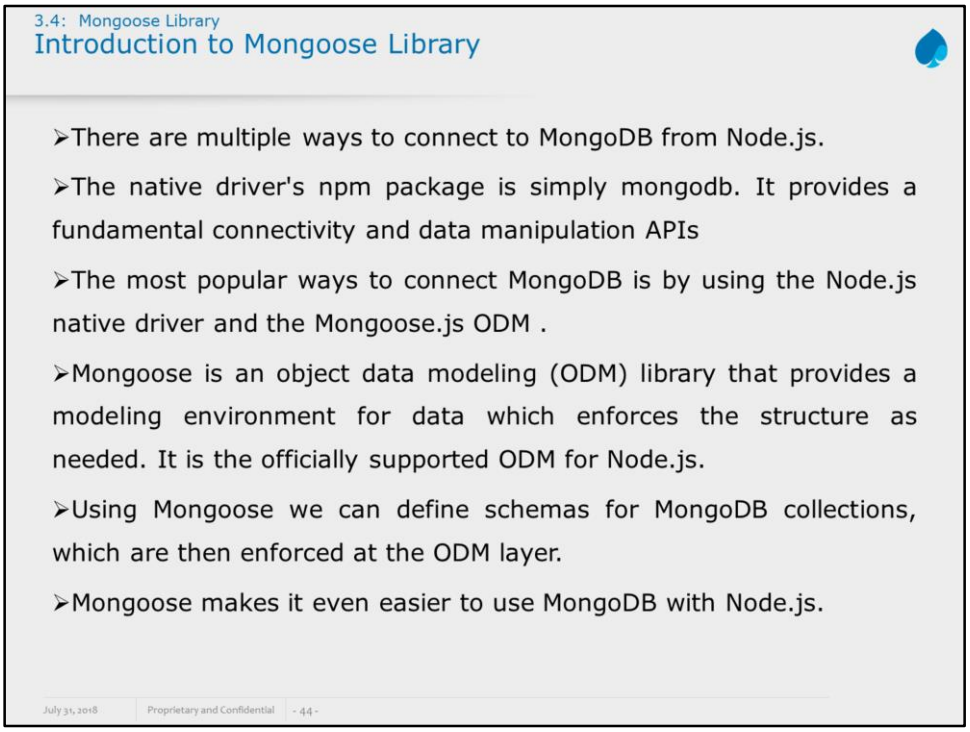
Instructor Notes:

Add instructor notes here.



Instructor Notes:

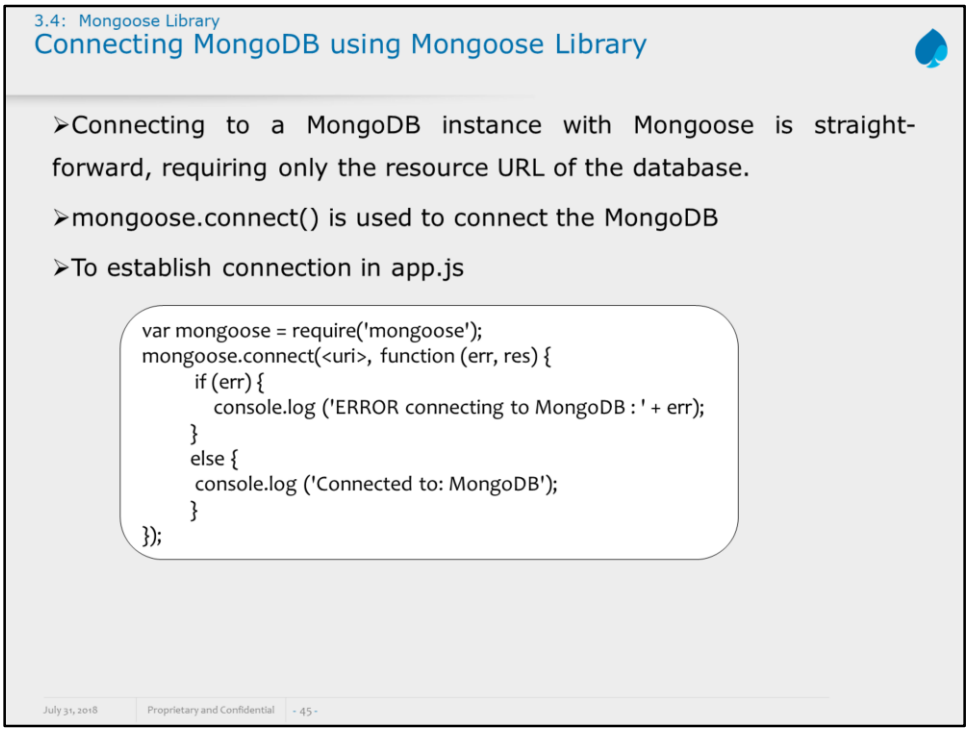
Add instructor notes here.



Mongoose has a thriving open source community and includes advanced schema-based features such as asyncvalidation, casting, object life-cycle management, pseudo-joins, and rich query builder support

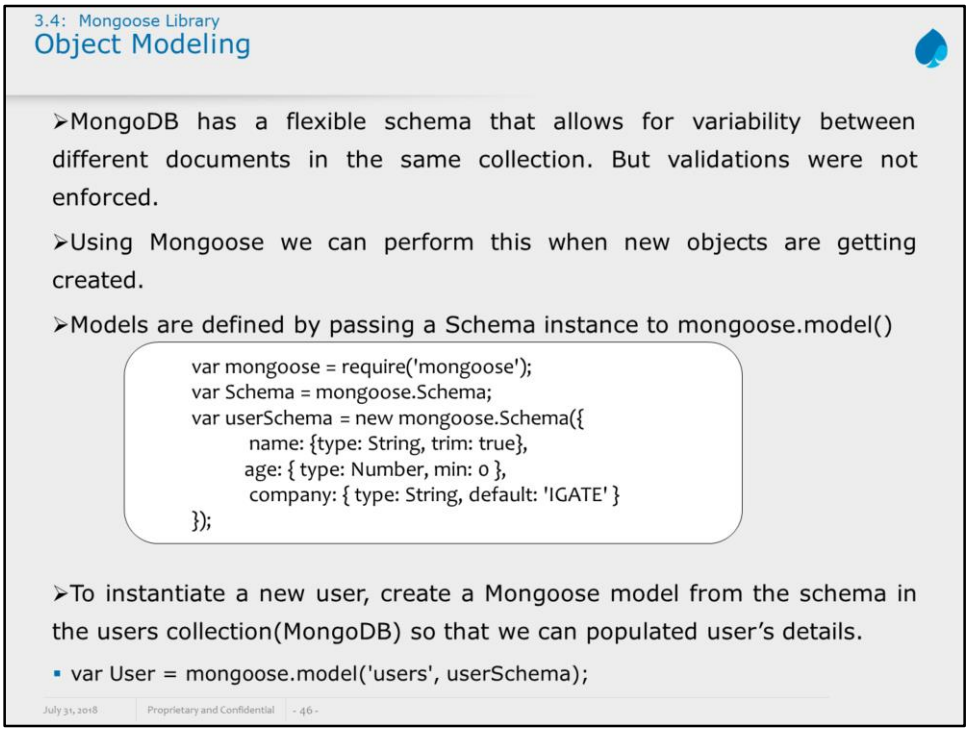
Instructor Notes:

Add instructor notes here.



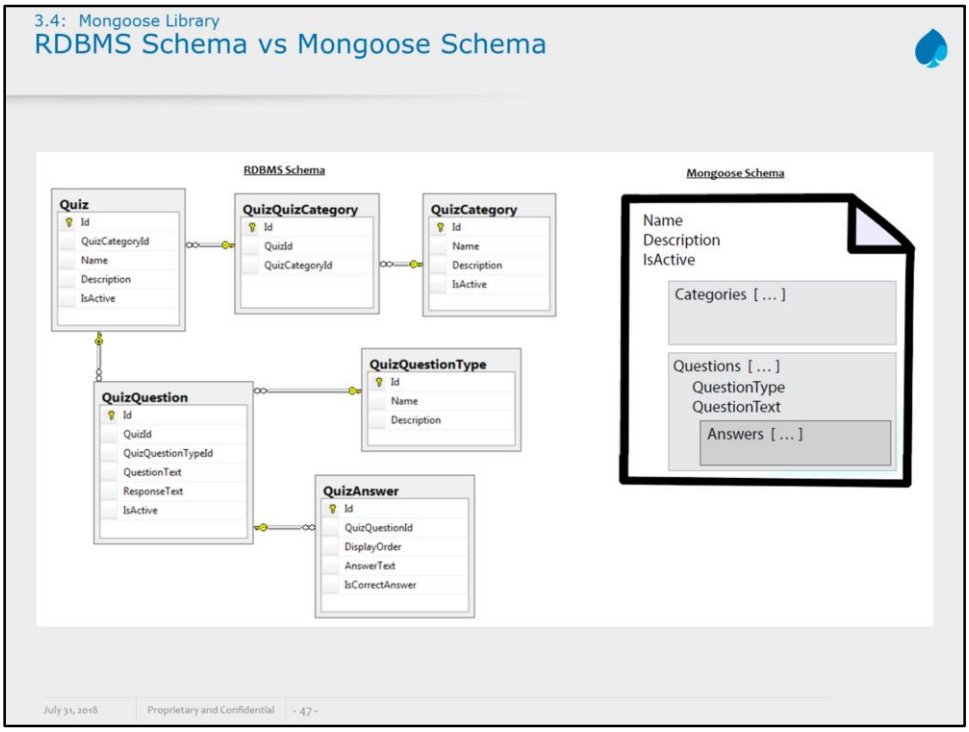
Instructor Notes:

Add instructor notes here.



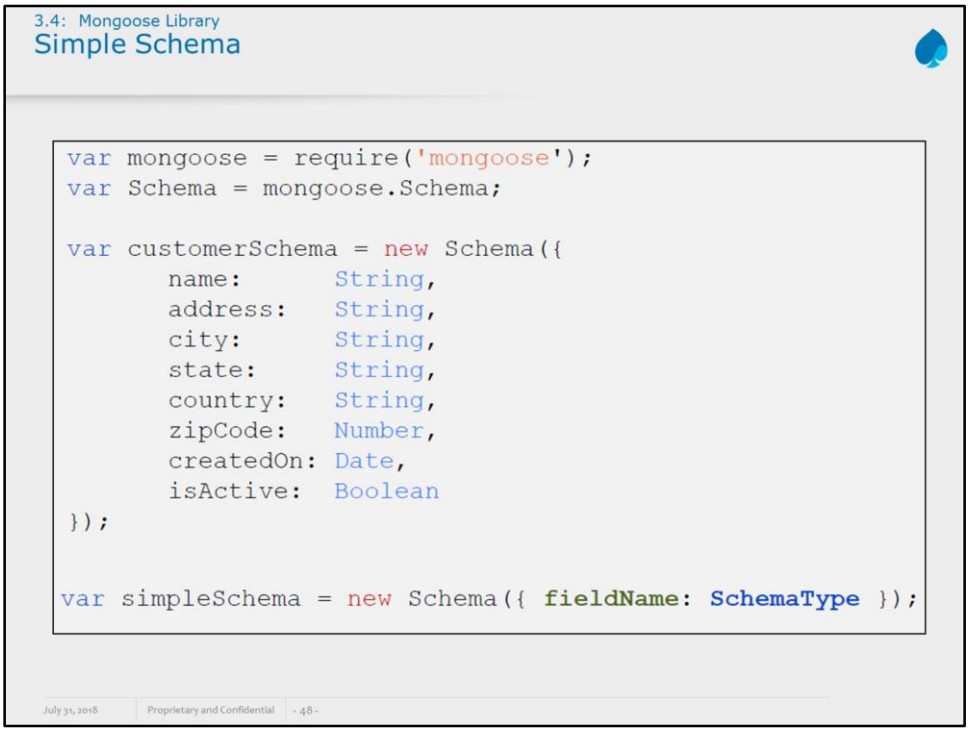
Instructor Notes:

Add instructor notes here.



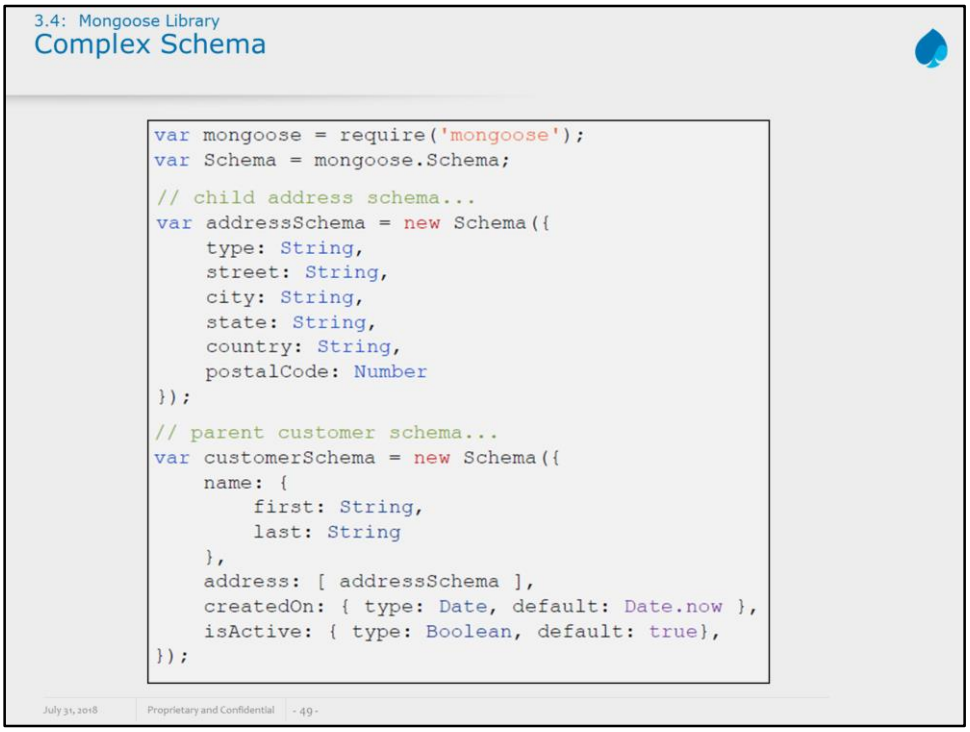
Instructor Notes:

Add instructor notes here.



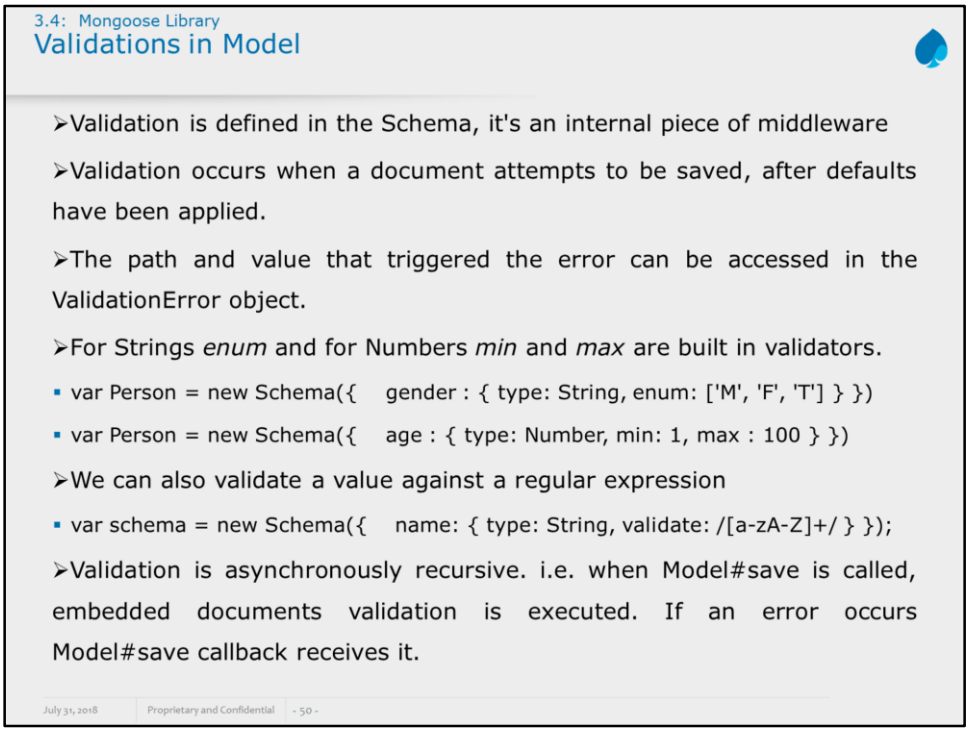
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Simple validation

Simple validation is declared by passing a function to validate and an error type to your SchemaType

```
function validator (v) {
  return v.length > 5;
};

new Schema({
   name: { type: String, validate: [validator, 'my error type'] }
})
```

Asynchronous validation

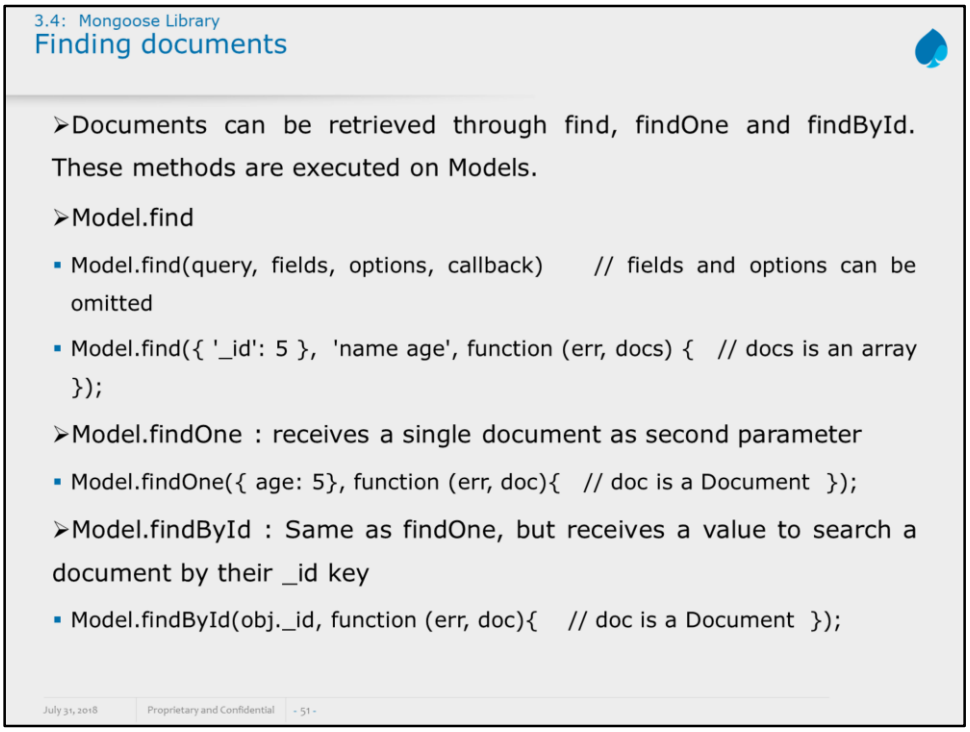
If you define a validator function with two parameters, like:

```
schema.path('name').validate(function (v, fn) {
  // my logic
}, 'my error type');
```

Then the function fn has to be called with true or false, depending on whether the validator passed. This allows for calling other models and querying data asynchronously from your validator.

Instructor Notes:

Add instructor notes here.



Model.count : Counts the number of documents matching conditions.
Model.count(conditions, callback);

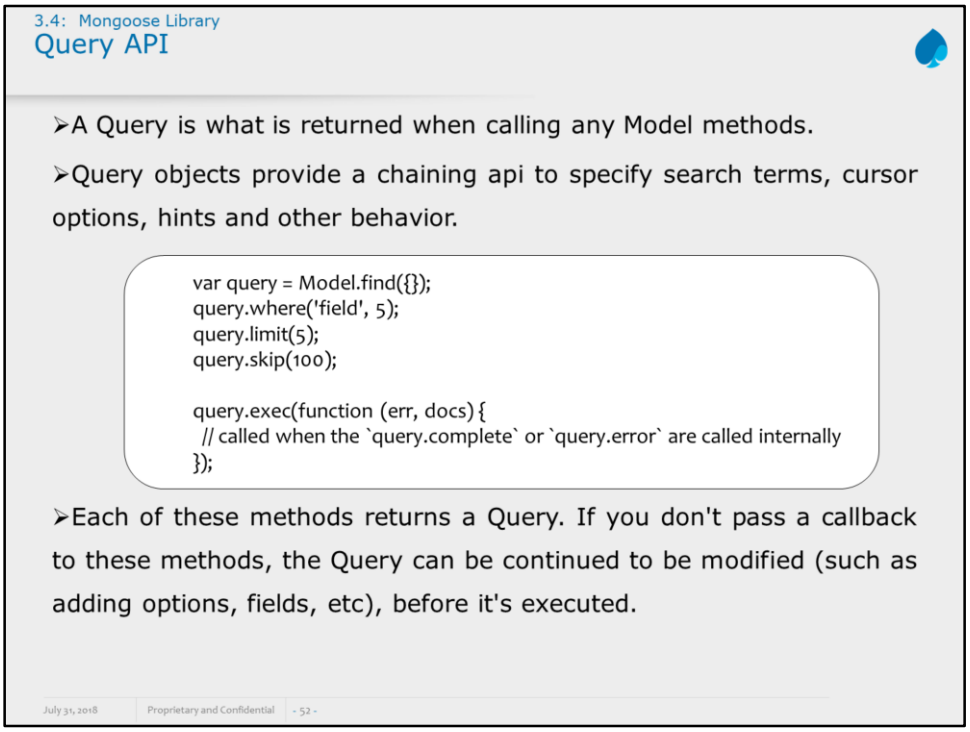
Model.remove : Removes documents matching conditions.
Model.remove(conditions, callback);

Model.distinct : Finds distinct values of field for documents matching conditions.
Model.distinct(field, conditions, callback);

Model.$where : Querying Mongodb using a JavaScript expression
Model.$where('this.firstname === this.lastname').exec(callback)

Instructor Notes:

Add instructor notes here.



For Code Snippets refer : Mongoose ODM-Queries.pdf

Instructor Notes:

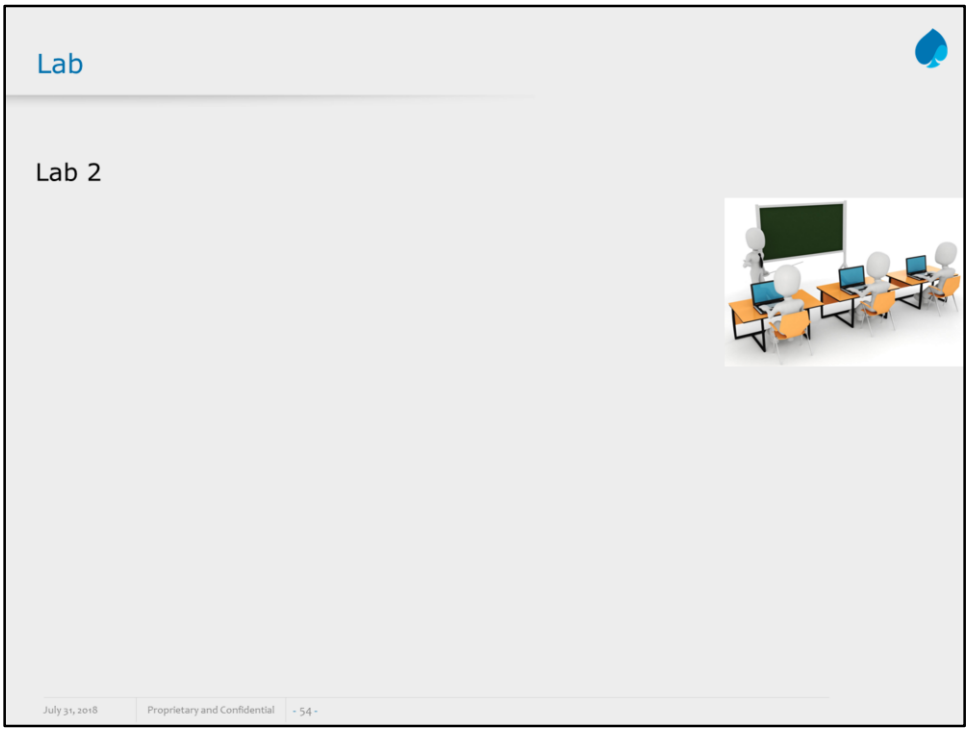
Demo

nodewithmongo

July 31, 2018 | Proprietary and Confidential | - 53 -

Instructor Notes:

Add instructor notes here.



Add the notes here.