mongoDB®

INF 551

Wensheng Wu

Installation on EC2

- Create a new yum repository file for MongoDB
  - cd /etc/yum.repos.d
  - sudo nano mongodb-org-3.4.repo

- Add the following content to the file:
  - [mongodb-org-3.4]
  - name=MongoDB Repository
  - baseurl=https://repo.mongodb.org/yum/amazon/2013.03/mongodb-org/3.4/x86_64/
  - gpgcheck=1
  - enabled=1
  - gpgkey=https://www.mongodb.org/static/pgp/server-3.4.asc

Installation on EC2

- sudo yum -y install mongodb-org

- sudo service mongod start
  - Start the server

- sudo service mongod stop
  - Stop it

Document store

- MongoDB is a document database
- A document is similar to an JSON object
  - Consists of field-value pairs
  - Value may be another document, array, string, number, etc.
- Document = record/row in RDBMS

Collections

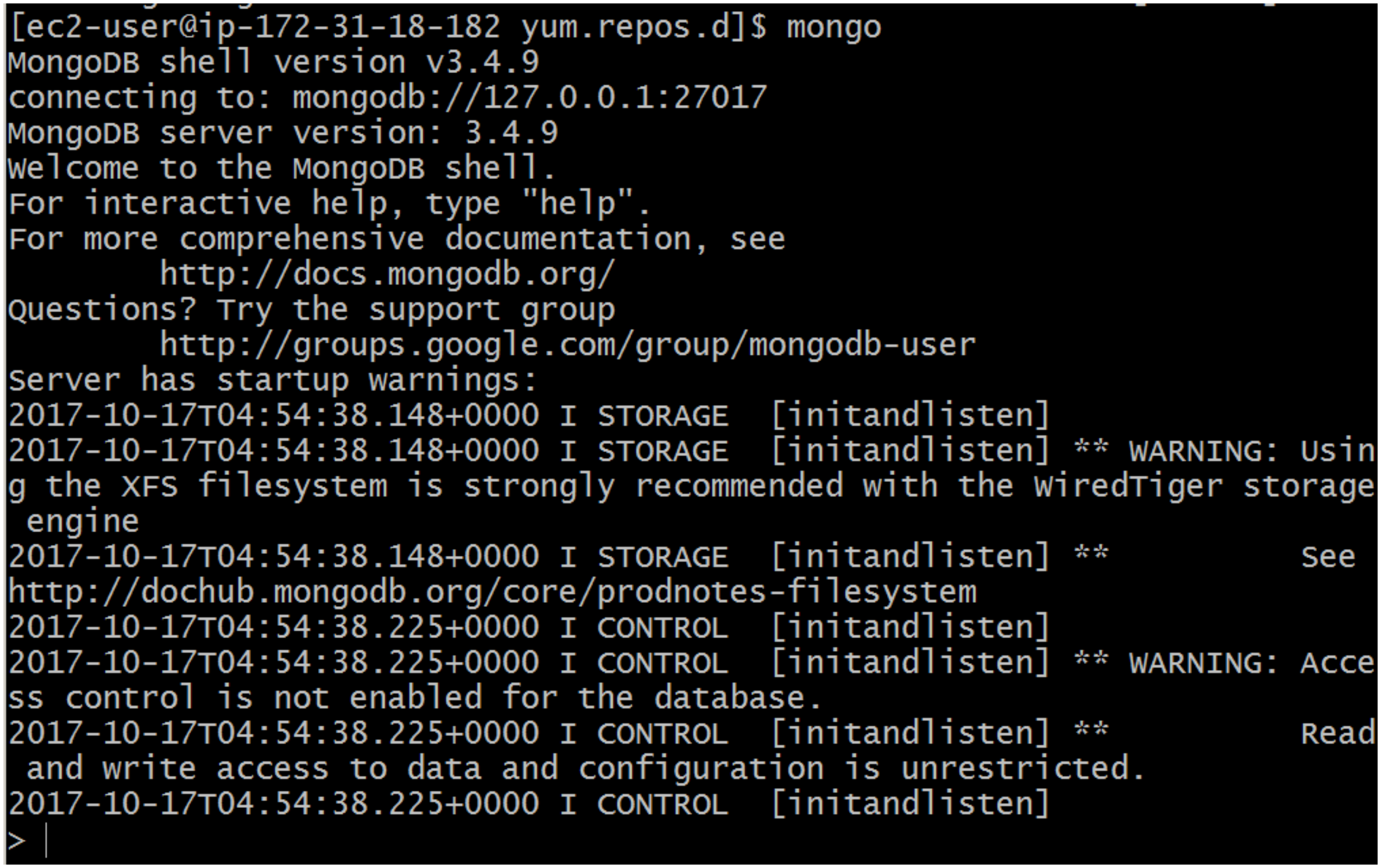
- Documents are stored in a collection
- Collection = table in RDBMS
- But documents may have different structures
  - In contrast, records in RDBMS have the same schema

Primary key

- Every document has a unique _id field
  - That acts as a primary key

MongoDB shell

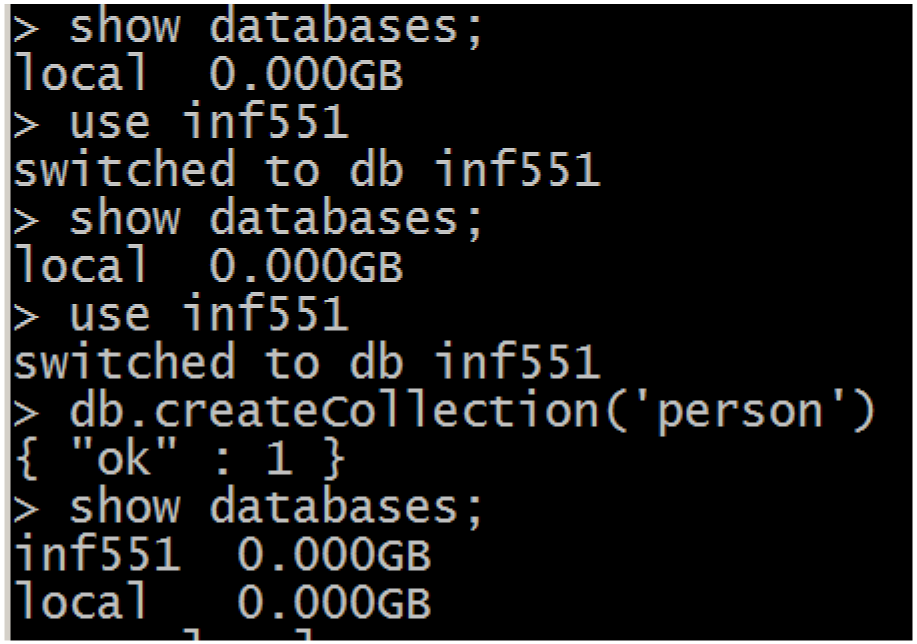
- mongo



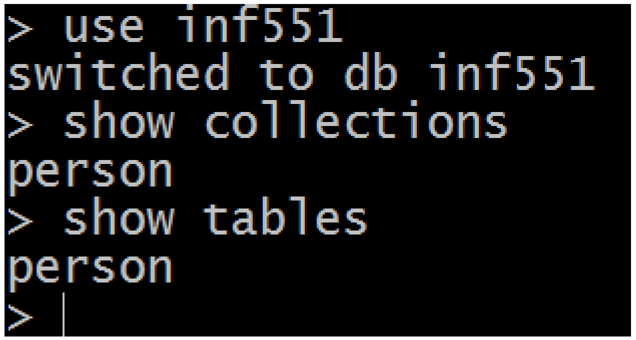
```
[ec2-user@ip-172-31-18-182 yum.repos.d]$ mongo
MongoDB shell version v3.4.9
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.9
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
Server has startup warnings:
2017-10-17T04:54:38.148+0000 I STORAGE  [initandlisten]
2017-10-17T04:54:38.148+0000 I STORAGE  [initandlisten] ** WARNING: Usin
g the XFS filesystem is strongly recommended with the WiredTiger storage
 engine
2017-10-17T04:54:38.148+0000 I STORAGE  [initandlisten] **          See
http://dochub.mongodb.org/core/prodnotes-filesystem
2017-10-17T04:54:38.225+0000 I CONTROL  [initandlisten]
2017-10-17T04:54:38.225+0000 I CONTROL  [initandlisten] ** WARNING: Acce
ss control is not enabled for the database.
2017-10-17T04:54:38.225+0000 I CONTROL  [initandlisten] **          Read
 and write access to data and configuration is unrestricted.
2017-10-17T04:54:38.225+0000 I CONTROL  [initandlisten]
>
```

Create a new database

- No need to explicitly create it, just use it
  - It will be automatically created once you add a collection (i.e., table) to it



```
> show databases;
local   0.000GB
> use inf551
switched to db inf551
> show databases;
local   0.000GB
> use inf551
switched to db inf551
> db.createCollection('person')
{ "ok" : 1 }
> show databases;
inf551  0.000GB
local   0.000GB
```



```
> use inf551
switched to db inf551
> show collections
person
> show tables
person
>
```

Databases

- use inf551
  - Switch to database "inf551"
- show databases
  - List all databases
- show tables/show collections
  - List all tables/collections in the current db
  - Can also say "show collections"

Database

- Dropping a database
  - db.dropDatabase();

Create/drop a collection

- db.createCollection('person')
  - db is a shell variable representing the current db

- db.person.drop()
  - Dropping a collection

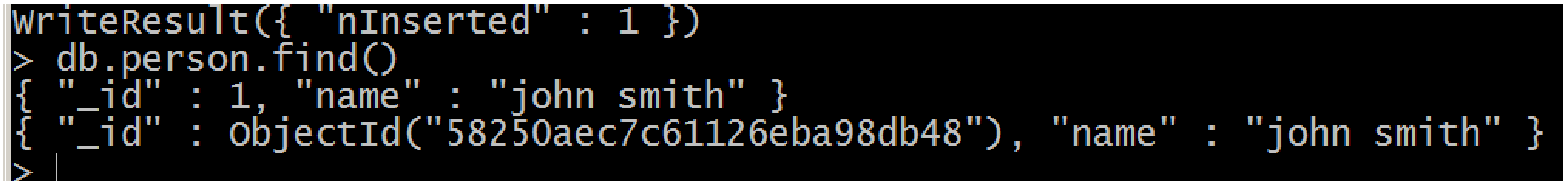
Adding documents

- db.person.insert({"_id": 1, "name": "john smith"})
- db.person.insert({"_id": 1, "name": "david smith"})
  - Error: duplicate key!

ObjectId()

- ObjectId() function creates an ID

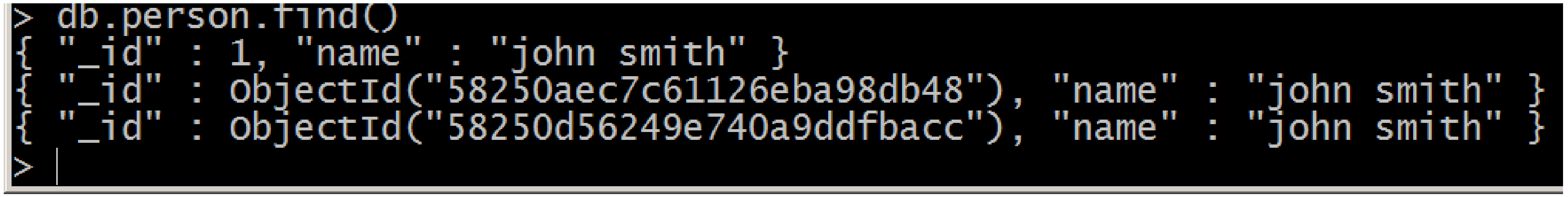
- db.person.insert({"_id": ObjectId(), "name": "john smith"})



```
WriteResult({ "nInserted" : 1 })
> db.person.find()
{ "_id" : 1, "name" : "john smith" }
{ "_id" : ObjectId("58250aec7c61126eba98db48"), "name" : "john smith" }
>
```

ObjectId()

- db.person.insert({"name": "john smith"})
  - Here no specification of "_id" field
  - Bu an id will be automatically created



```
> db.person.find()
{ "_id" : 1, "name" : "john smith" }
{ "_id" : ObjectId("58250aec7c61126eba98db48"), "name" : "john smith" }
{ "_id" : ObjectId("58250d56249e740a9ddfbacc"), "name" : "john smith" }
> |
```

ObjectId()

- A 12-byte hexademical value
  - E.g., 58250aec7c61126eba98db48

- Among 12 bytes:
  - 4-byte: the seconds since 1970/1/1
  - 3-byte: machine identifier
  - 2-byte: process id
  - 3-byte: a counter, starting with a random value

Embedded sub-document

- db.person.insert(

```
{
"name": "david johnson",
"address": {"street": "123 maple",
             "city": "LA",
             "zip": 91989},
"phone": ["323-123-0000", "626-124-0999"],
"scores": [25, 35]
})
```

Array

Insert some more documents

- db.person.insert({"name": "kevin small", "age": 35, "scores":[12, 20]})

- db.person.insert({"name": "mary lou", "age": 25})

Query

- db.person.find()
  - Return all documents in person

- db.person.find({"name": "kevin small"})
  - Return all documents with specified name

- db.person.find().pretty()
  - Pretty print the output

Query operators

- Introduced by $
- $lt, $gt, $lte, $gte, $ne
  - Comparison operators
- $or, $and, $not
  - Logical operators

Query operators

- db.person.find({"age": {$gt: 25}})

- db.person.find({"name": "kevin small", "age": {$gt: 25}})
  - Specify "and" condition

- db.person.find({ $or: [{"name": "kevin small"}, {"age": {$gt: 25}}] })
  - Specify "or" condition

Query operator

- What does each of these queries find?
  - db.person.find({$or: [{"name":/kevin/i}, {"age":25}]})
  - db.person.find({$or: [{"name":/kevin/i, "age":25}]})
  - db.person.find({$and: [{"name":/kevin/i}, {"age":25}]})

Query operator

- db.person.find({name: {$not: {$eq: "john"}}})

Same as:

- db.person.find({name: {$ne: "john"}})

Pattern matching

- db.person.find({"name":/Kevin/i})
  - This finds person whose name contains "kevin"
  - "i" means case-insensitive

$regex is a query operator

- Above is equivalent to:
  - db.person.find({"name":{$regex: /Kevin/, $options: 'i'}})

- In general, /pattern/ where pattern is a regular expression

Matching elements in array

- db.person.find({"scores": {$gt: 20}})
  - Note the "scores" field is an array and at least one value of the array should satisfy the specified condition (i.e., > 20)

Sorting

- db.person.find().sort({age:-1})
  - 1 for ascending; -1 descending

- Equivalent to:

  Select *

  From person

  Order by age desc

Limit

- db.person.find().limit(1)
  - Returns the first person

Distinct

- db.person.distinct("age")

- db.person.distinct("age", {age: {$gt: 20}})
  – distinct ages (for ages > 20)

Projection

- db.person.find(

  ```
  {"age": {$ne: 25} },
  {"name":1, "age": 1}
  )
  ```

Specify query condition

Specify projection
1: included in result; 0: do not

- This will return name and age (plus _id)
  – i.e., similar to 'select _id, name, age from users where age != 25'

Projection

- This does not work:
  - db.person.find(

```
      {"age": {$ne: 25} },
      {"name":1, "age": 0}
)
```

  - Can not mix 1 and 0 conditions (unless it is "_id")

Projection

- db.person.find(

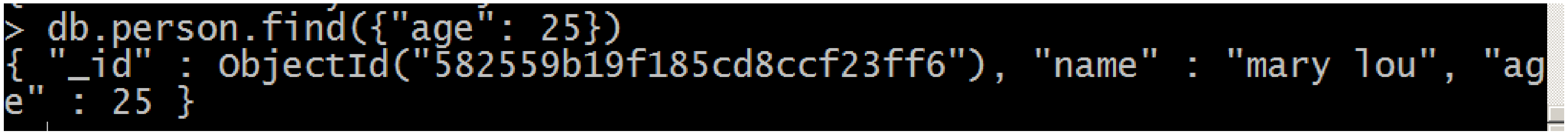
```
        {"age": {$ne: 25} },
        {"name":1, "age": 1, "_id": 0}
 )
```

- This does not return id, e.g.,

```
 { "name" : "john smith" }
 { "name" : "david johnson" }
 { "name" : "kevin small", "age" : 35 }
```

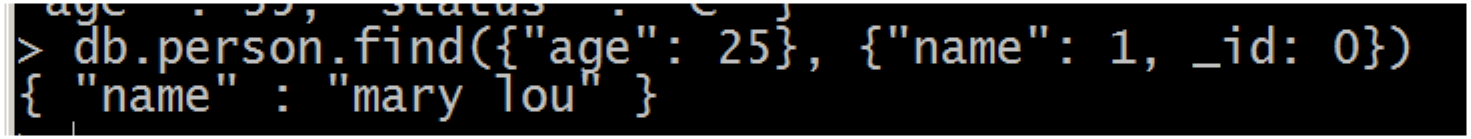
Example

- Without projection



```
> db.person.find({"age": 25})
{ "_id" : ObjectId("582559b19f185cd8ccf23ff6"), "name" : "mary lou", "ag
e" : 25 }
```

- With projection



```
> db.person.find({"age": 25}, {"name": 1, _id: 0})
{ "name" : "mary lou" }
```

Update documents

- db.person.update(

      { "age": { $gt: 25 } },
      { $set: { "status": "C" } },
      { multi: true }
)

Existing documents may not have status field; if not, insert it instead

Update one or all documents

Similar to:

Update users set status = 'C' where age > 25

Another example

- db.person.update({}, {$set: {"status":'C'}}, {multi:true})
  - Note the empty query {}

- Add "status" field to all documents

Remove fields

- db.person.update({}, {$unset: {"status": ""}}, {multi: true})

Can put any value here

- Remove the "status" field from all documents

Remove documents

- db.person.remove({})
  - Remove all documents/records of person
- db.person.remove({ "age": {$gt: 30} })
  - Remove documents which satisfy a condition

Remove a collection/table

- db.person.drop()
  - This will remove the person collection/table

Count()

- db.person.count()
  - Return # of documents in the person collection
- db.person.count({age: {$gt: 25}})
  - What does this do?
- db.person.find({age: {$gt: 25}}).count()

Query a embedded document

- Using dot notation to identify field in theembedded document

- db.person.find({"address.city": "LA"})
  - Return all documents whose city sub-field of address field = "LA"

Example for aggregation

- db.product.insert({category: "cell", store:1, qty: 10})
- db.product.insert({category: "cell", store:2, qty: 20})
- db.product.insert({category: "laptop", store:1, qty: 10})
- db.product.insert({category: "laptop", store:2, qty: 30})
- db.product.insert({category: "laptop", store:2, qty: 40})

Aggregation

- db.product.aggregate([{$group: {_id: "$category", total:{$sum:"$qty"}}}])
  - { "_id" : "laptop", "total" : 80 }
  - { "_id" : "cell", "total" : 30 }

- Similar to: "select category, sum(qty) from product group by category"

Aggregation

- db.product.aggregate([{"$group": {_id: "$category", total:{$sum:1}}}])
  ```
  { "_id" : "laptop", "total" : 2 }
  { "_id" : "cell", "total" : 2 }
  ```

- Similar to: "select category, count(*) from product group by category"

Aggregation with "having …"

- db.product.aggregate([{$group: {_id: "$category", total:{$sum:"$qty"}}}, {$match: {total: {$gt: 50}}}])
  - { "_id" : "laptop", "total" : 80 }

- In SQL:

  Select category, sum(qty) total

  from product

  group by category

  having total > 50

Aggregation on more than one field

- db.product.aggregate([{$group: {_id: {cat: "$category", st: "$store"}, total:{$sum:"$qty"}}}])

```
{ "_id" : { "cat" : "laptop", "st" : 1 }, "total" : 10 }
{ "_id" : { "cat" : "laptop", "st" : 2 }, "total" : 70 }
{ "_id" : { "cat" : "cell", "st" : 2 }, "total" : 20 }
{ "_id" : { "cat" : "cell", "st" : 1 }, "total" : 10 }
```

Aggregation

- Other operators
  - $avg
  - $min
  - $max

Aggregation pipeline

- db.person.aggregate([{$match: {age: {$gt: 25}}}, {$group:{_id: "$gender", val: {$min: "$weight"}} }, {$match: {val: {$gt: 120}}}, {$limit: 2}, {$sort: {val: -1}}])

- $match -> $group -> $match -> $limit -> $sort

Sharding in MongoDB

- Distribute documents/records in a large collection/table over multiple machines
- User can specify a sharding key
  - i.e., a field in a document
- Support sharding by key range or hashing

Sample data set

- Restaurants data
  - https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json

Import sample dataset

- mongoimport --db inf551 --collection restaurants --file primer-dataset.json
  – No need to pre-create inf551 and restaurants if they do not exist yet
- More details:
  – https://docs.mongodb.com/getting-started/shell/import-data/

Resources

- Install MongoDB Community Edition on Amazon Linux
  - https://docs.mongodb.com/manual/tutorial/install-mongodb-on-amazon/