INF 551

Wensheng Wu

Installation on EC2

- Create a new yum repository file for MongoDB
  - cd /etc/yum.repos.d
  - Create a file called: mongodb-org-4.0.repo

- Add the following content to the file:
  - [mongodb-org-4.0]
  - name=MongoDB Repository
  - baseurl=https://repo.mongodb.org/yum/amazon/2013.03/mongodb-org/4.0/x86_64/
  - gpgcheck=1
  - enabled=1
  - gpgkey=https://www.mongodb.org/static/pgp/server-4.0.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"})
  - May omit "" in keys when doing insert
  - May also use single quotes (unlike JSON)

- 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 multiple documents at once

- db.person.insertMany([{"name": "kevin small", "age": 35, "scores":[5, 6, 3]}, {"name": "mary lou", "age": 25, "scores":[5,8,2]}])

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, $in, $all
  - 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

- db.person.find({name: {$not: {$eq: "john"}}})
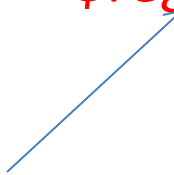 - May omit "" in keys when doing find

Same as:

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

$in

- db.person.find({age: {$in: [25, 35]}})
  - Find persons whose age are either 25 or 35

- $in matches any of the values in the array

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

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}]})

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)

- db.person.find({scores: {$all: [2, 5]}})
  - – Find persons whose scores contain both 2 and 5 (all elements in the given array)
  - – Compared to: db.person.find({scores: {$in: [2, 5]}})

Sorting

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

- Equivalent to:

  Select *

  From person

  Order by age desc

Skip & limit

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

- db.person.skip(1).limit(1)

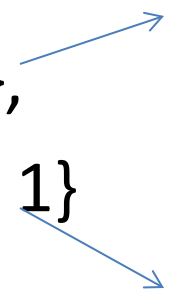# Distinct

- db.person.distinct("age")

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

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()

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'

30

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

```
age   : 55,   status   : c  }
> db.person.find({"age": 25}, {"name": 1, _id: 0})
{ "name" : "mary lou" }
```

Update/upsert 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

Query a embedded document

- Using dot notation to identify field in the embedded document

- db.person.find({"address.city": "LA"})
  - Return all documents whose city sub-field of address field = "LA"
  - Note "" is required here for key

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: sum

- 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: count

- 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.product.aggregate({$match: {store: 2}}, {$group: {_id: "$category", total: {$sum: "$qty"}}}, {$match: {total: {$gt: 10}}}, {$limit: 2}, {$sort: {total: 1}})

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

```
{ "_id" : "cell", "total" : 20 }
{ "_id" : "laptop", "total" : 70 }
```

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

Hash function…

- h("abc") = ?

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C		
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

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/