DSCI 551

Wensheng Wu

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 (relational database management system)

Collections

- Documents are stored in a collection

  rows become documents

- 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
  - Or "db.getCollectionNames()"

- ## show tables/show collections
  - List all tables/collections in the current db
  - Can also say "show collections"

Database

- Dropping a database
  - db.dropDatabase()

- Show current database
  - db

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"}) 新语法：不用insert，用inesrtOne
  - 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
  When you don't specify a primary key '_id',
  this function generates one for you.

- 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
  - 48 = 0100 1000
  - d = 1101 E = 1110 F = 1111
- 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"],
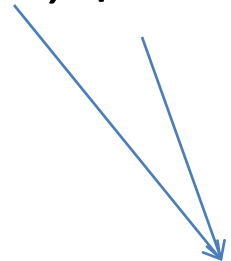   "scores": [25, 35]
    })

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]}])

- But note that insert([...]) takes multiple docs too

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, $eq, $ne, $in, $all
  - Comparison operators

- $or, $and, $not
  - Logical operators
  - $and/$or requires array [...] as value
  - $not requires either a regex /.../ or a document {...}; it can not be a top-level operator either

Value is an array

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}}] })
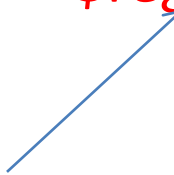 - Specify "or" condition

Query operator

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

- ERROR:
  - db.person.find({$not: {"name": "david"}})
  - $not ~~can~~ be a top-level operator
    cannot

$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

Pattern matching

- db.person.find({name: {$not: /john/i}})

- Note that this matches documents w/o "name" too, e.g.,
  - { "_id" : ObjectId("5db0f0ef51ae73d9a5803cd1"), "age" : 25, "gender" : "F" }

- Unless add condition: {name: {$exists: true}}

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}]})

Multiple conditions on same attribute

- Range query:
  - db.person.find({age: {$gt:25, $lt: 30}})

- db.person.find({age: {$ne: 20, $exists: true}})
  - db.person.find({age: {$ne: 20}}) results include persons without ages or with null ages

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)

  { "scores" : [3, 2, 3, 7, 8, 5], "name" : "johnson" }
  { "scores" : [5, 2, 4, 7, 9], "name" : "john smith" }

- db.person.find({scores: {$all: [2, 5]}})
  - Find persons whose scores contain both 2 and 5 (what if we change $all to $in?)

  ⇔db.person.find({$and: [{scores: 2}, {scores: 3}]})

  in就是只要有一个就选
  上，all是要全都包含才选

Condition on document elements of array

- db.person.find({"scores.midterm": "A"})
- db.person.find({"scores.midterm": "B", "scores.score": {$gt: 90}})
  - Note the score may not be the score for the midterm

{ "name" : "apple", "scores" : [{ "midterm" : "A", "score" : 93 }, { "final" : "B", "score" : 75 }] }

{ "name" : "tangerine", "scores" : [{ "final" : "A", "score" : 99 }, { "midterm" : "B", "score" : 88 }] }

{ "name" : "orange", "scores" : [{ "midterm" : "B", "score" : 85 }, { "final" : "A", "score" : 92 }] }

Condition on same element of array

- db.person.find({"scores": {$elemMatch: {"midterm": "B", "score": {$gt: 90}}}})
  - No match this time

{ "name" : "apple", "scores" : [{ "midterm" : "A", "score" : 93 }, { "final" : "B", "score" : 75 }] }
{ "name" : "tangerine", "scores" : [{ "final" : "A", "score" : 99 }, { "midterm" : "B", "score" : 88 }] }
{ "name" : "orange", "scores" : [{ "midterm" : "B", "score" : 85 }, { "final" : "A", "score" : 92 }] }

Sorting

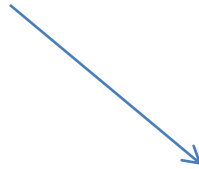This is needed

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

  Select *

  From person

  Order by age desc

- What about: db.person.find().sort({name:1, age:-1})

Skip & limit

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

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

Skip needs to follow find()

Distinct

- db.person.distinct("age")

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

- Note: MongoDB does not allow mixing distinct with find
  - E.g., db.find(...).distinct(...)

Distinct

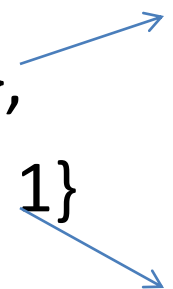- db.person.distinct("age").length
  - Return # of distinct ages

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()

Watch out for null values

- db.person.find({age: {$ne: 20}})
  - This will also return persons whose ages are NULL

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'

34

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" }
```

Renaming in projection

- db.person.find({}, {home: "$address"})

- Example:

  { "_id" : 5, "address" : { "city" : "LA", "state" : "CA" } }

  =>

  { "_id" : 5, "home" : { "city" : "LA", "state" : "CA" } }

Update/upsert documents

- db.person.update(

    { "age": { $gt: 25 } },

    { $set: { "status": "C" } },

    { multi: true }

)

Similar to:

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

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

Update one or all documents

Without $set, it will be an overwrite

Update many

- db.person.updateMany(…)
  - Equivalent to update() with multi set to true

upsert option

- db.student.update({age: 25}, {$set: {name: 'jonny'}}, {upsert: true})

- If no documents with age = 25, a new document will be inserted with random id
  - { "_id" : ObjectId("5e750d2cdea10ee6edfdfb24"), "age" : 25, "name" : "jonny" }

- db.student.update({_id: 2}, {$set: {name: 'jonny'}}, {upsert: true})
  - If no document with id = 2, a new doc with id = 2 will be inserted

More update examples

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

- Setting multiple attributes:
  - {$set: {"status":'C', "gender": "M"}}

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
  - Similar to "delete from person where age > 30"

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" : 3 }

  { "_id" : "cell", "total" : 2 }

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

Sum and count

- db.product.aggregate({"$group": {_id: "$category", sum: {$sum: "$qty"}, cnt:{$sum:1}}})
  - { "_id" : "laptop", "sum" : 80, "cnt" : 3 }
  - { "_id" : "cell", "sum" : 30, "cnt" : 2 }

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: {$gt: 10}}}, {$sort: {total: 1}}, {$limit: 2})

- $match -> $group -> $match -> $sort -> $limit
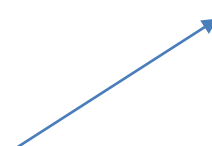
{ "_id" : "cell", "total" : 20 }
{ "_id" : "laptop", "total" : 70 }

Projection in aggregate

- db.product.aggregate({$group:{_id: null, max: {$max: "$qty"}}})
  - Getting global max
  - { "_id" : null, "max" : 45 }
  - select max(qty) from product;

- Remove _id from result:
  - db.product.aggregate({$group:{_id: null, max: {$max: "$qty"}}}, {$project: {_id: 0}})
  - { "max" : 45 }

$lookup for joining two collections

{ "_id" : 1, "name" : "david", "age" : 25, "gender" : "male", "address" : { "city" : "LA", "state" : "CA" }, "deptID" : 1 }

- db.person.aggregate({$lookup: {from: 'department', localField: 'deptID', foreignField: "_id", as: 'res'}}, {$match: {name: {$ne: null}}}, {$project: {name: 1, 'res.name': 1, _id: 0}})

  – person.deptID = department._id

- Department:

  { "_id" : 1, "name" : "CS" }
  { "_id" : 2, "name" : "ECE" }

```
db.person.aggregate(
    {$lookup: {
        from: 'department',
        localField: 'deptID',
        foreignField: "_id",
        as: 'res'
        }
    },
    {$match:
      {name: {$ne: null}}
    },
    {$project:
      {name: 1, 'res.name': 1, _id: 0}
    }
)
```

Unwind

- Form a new document for each element in an array

- Example:
  - db.person.aggregate({$match: {did: {$exists: true}}}, {$lookup: {from: "department", localField: "did", foreignField: "_id", as: "res"}}, {$project: {name: 1, deptName: "$res.name", _id: 0}}, {$unwind: "$deptName"})
  - { "name" : "bill", "deptName" : ["CS"] }

    => { "name" : "bill", "deptName" : "CS" }

Sales

```
db.sales.insertMany([
{
    _id: 0,
    items: [
        { item_id: 43, quantity: 2, price: 10 },
        { item_id: 2, quantity: 1, price: 240 }
    ]
},
{
    _id: 1,
    items: [
        { item_id: 23, quantity: 3, price: 110 },
        { item_id: 103, quantity: 4, price: 5 },
        { item_id: 38, quantity: 1, price: 300 }
    ]
},
{
    _id: 2,
    items: [
        { item_id: 4, quantity: 1, price: 23 }
    ]
}])
```

$filter

- Selects a subset of an array to return based on the specified condition

```
{
  _id: 0,
  items: [ { item_id: 43, quantity: 2, price: 10 },   { item_id: 2, quantity: 1, price: 240 } ]
}
{
  _id: 1,
  items: [    { item_id: 23, quantity: 3, price: 110 },  { item_id: 103, quantity: 4, price: 5 },
              { item_id: 38, quantity: 1, price: 300 }   ]
}
{
  _id: 2,
  items: [        { item_id: 4, quantity: 1, price: 23 }     ]
}
```

Example

- Get only items whose price > 100

```
db.sales.aggregate(
  {
    $project: {
      myitems: {
        $filter: {
          input: "$items",
          as: "item",
          cond: { $gte: [ "$$item.price", 100 ] }
        }
      }
    }
  },
  {$match: {myitems: {$ne: []}}}
)
```

result:

```
[
  { _id: 0, myitems: [ { item_id: 2, quantity: 1, price: 240 } ] },
  {
    _id: 1,
    myitems: [
      { item_id: 23, quantity: 3, price: 110 },
      { item_id: 38, quantity: 1, price: 300 }
    ]
  }
]
```

Sharding in MongoDB

- Distribute documents/records in a large collection/table over multiple machines

- User can specify a sharding/partitioning key
  - i.e., a field in a document

- Support sharding by key range or hashing

- (Hadoop mapreduce: partitioner)

Hash function…

- h("john") = (106+111+104+110) % 2 = 1

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 dsci551 --collection restaurants --file primer-dataset.json
  - No need to pre-create dsci551 and restaurants if they do not exist yet

- More details:
  - https://docs.mongodb.com/manual/reference/program/mongoimport/

Writing Javascript in Mongo Shell

- Mongo shell supports Java-scripting

  Note cursor is a generator
  Similar to a generator in Python

- var cursor = db.person.find();
  while (cursor.hasNext()) {
      printjson(cursor.next());
      }

- db.person.find().forEach(function (p) {printjson (p);});

Writing Javascript in Mongo Shell

- var persons = db.person.find().toArray();

- persons[0]
  - { _id: ObjectId("652eea58ff6e4a0a9fe6f807"), name: 'john', age: 27 }

- var myage = 25

- db.person.find({'age': myage})
  - [{ _id: 6, age: 25 }]

Other useful methods

- db.person.findOne()
  - Find a single document
- db.person.deleteOne()
  - Remove a single document
- db.person.explain().find({age: {$gt: 20}})
  - Explain query execution plan

Index

- Useful for searching and sorting results
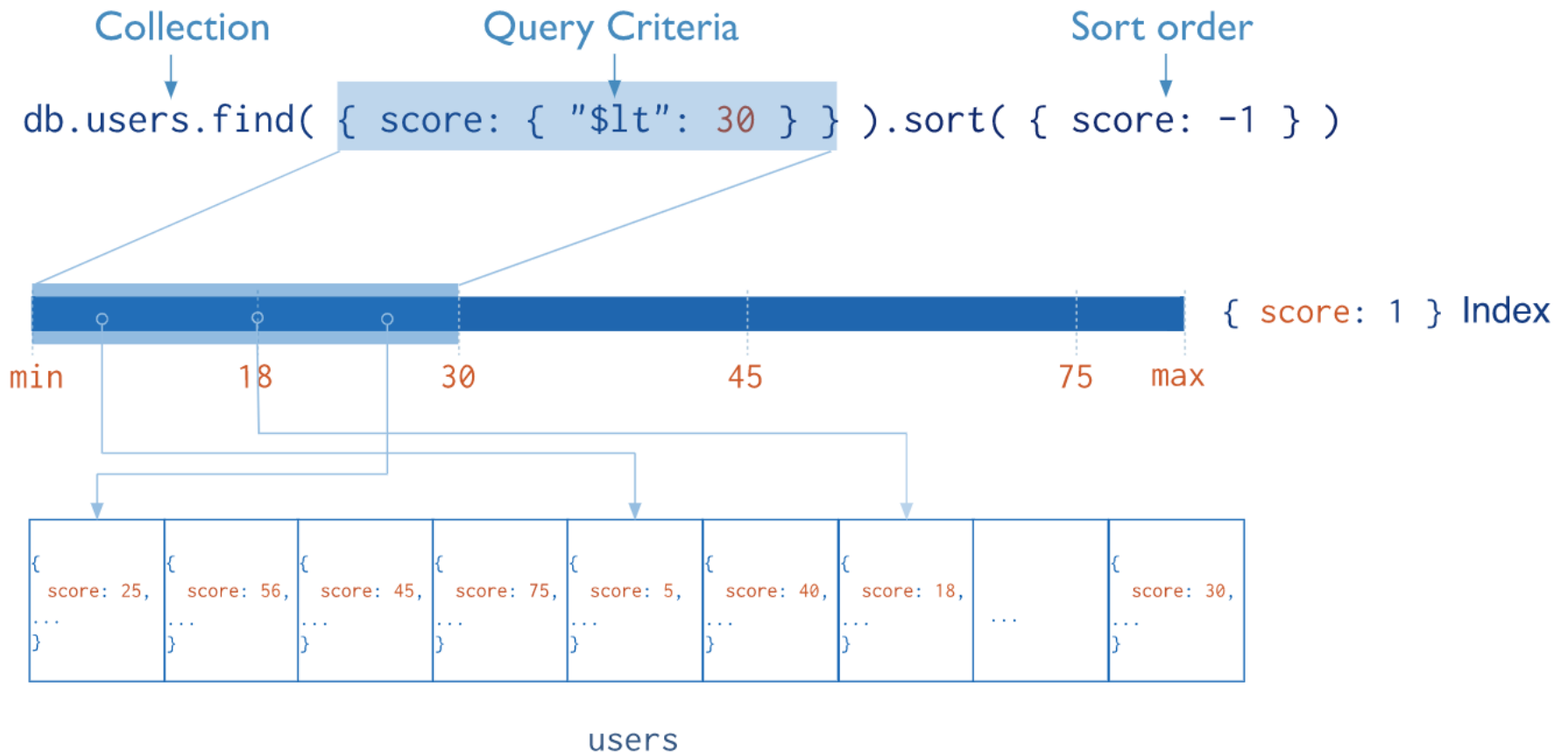
Creating an index

- db.users.createIndex({score: 1})
  - This creates an index on score
  - The index entries are sorted by score, ascending

- db.users.getIndexes()
  - This retrieves info of available indexes on users

Unique index

- db.users.createIndex({ssn: 1}, {unique: true})
  - This creates a unique index in the ascending order of ssn

- Similar to MySQL, unique field can take null value

Using index in query

- db.users.explain().find({score: {$gt: 30}})


```
> db.users.explain().find({score: {$gt: 30}})
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "inf551.users",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "score" : {
                                "$gt" : 30
                        }
                },
                "winningPlan" : {
                        "stage" : "FETCH",
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "keyPattern" : {
                                        "score" : 1
                                },
                                "indexName" : "score_1",
                                "isMultiKey" : false,
                                "multiKeyPaths" : {
                                        "score" : [ ]
                                },
                                "isUnique" : false,
                                "isSparse" : false
```

Dropping an index

- db.users.dropIndex("ssn_1")
  - ssn_1 is index name (shown in getIndexes())

Resources

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

- MongoDB references
  - https://docs.mongodb.com/master/reference/
  - $filter (https://docs.mongodb.com/master/reference/operator/aggregation/filter/#exp._S_filte)