

















NoSQL Data Model: Document Stores

Data Model

Document Database	Graph Databases
   	 
Wide Column Stores	Key-Value Databases
   	     

@cloud91 <http://www.anyanyany.com>

Document Stores

- Data model
 - **Documents**
 - Self-describing
 - **Hierarchical tree structure** (JSON, XML)
 - Scalar value, maps, lists, set, nested document
 - Identities by a uniq identified
 - Document are organized into collections
- Query pattern
 - Create, update or remove a document
 - Retrieve documents according to complex query conditions

Document Stores

- Suitable use cases
 - Event logging, content management systems, blogs, web analytic, e-commerce application, ...
 - for structured documents with similar schema
- When not to use
 - **Set operations**
 - involving multiple documents
 - Design of document structure is constantly changing
 - when the required level of granularity would outbalance the advantages of aggregates

Document Stores



Mongodb in practices

<https://gist.github.com/aponxi/4380516>

Mongodb Services

- Install on OSX
 - Run '**brew install mongodb**' via terminal
 - Create **/data/db** as database path
 - Enable read/write permission '**sudo chown -R `id -un` /data/db**'
 - **Run the Mongo daemon**, in one of your terminal windows run *mongod*. This should start the Mongo server.
 - **Run the Mongo shell**, with the Mongo daemon running in one terminal, type *mongo* in another terminal window. This will run the Mongo shell which is an application to access data in MongoDB.
 - To exit the Mongo shell run *quit()*
 - To stop the Mongo daemon hit *ctrl-c*

Connect with password

- To create username

- `db.createUser({ "user": "cjundang", "pwd": "1q2w3e4r", "roles": ["readWrite", "dbAdmin"] })`
- `show users`
- `sudo mongo -u cjundang -p 1q2w3e4r --authenticationDatabase testdb`

- Test with Login

- `mongo -u USERNAME -p PASSWORD --authenticationDatabase DATABASENAME`
- `mongo -u cjundang -p 1q2w3e4r --authenticationDatabase "testdb"`

Mongodb Command

Basic Commands		
To do this	Run this command	Example
Connect to local host on default port 27017	mongo	mongo
Connect to remote host on specified port	mongo --host <i><hostname or ip address></i> --port <i><port no></i>	mongo --host 10.121.65.23 --port 23020
Connect to a database	mongo <i><host>/<database></i>	mongo 10.121.65.58/mydb
Show current database	db	db
Select or switch database ^[1]	use <i><database name></i>	use mydb
Execute a JavaScript file	load(<i><filename></i>)	load (myscript.js)
Display help	help	help
Display help on DB methods	db.help()	db.help()
Display help on Collection	db.mycol.help()	db.mycol.help()

Mongodb Command

Show Commands		
Show all databases	show dbs	show dbs
Show all collections in current database	show collections	show collections
Show all users on current database	show users	show users
Show all roles on current database	show roles	show roles

CRUD - Insert

- Insert new document to collections
 - **db.collection.insert(<document>)**
- Example
 - `db.books.insert({"isbn": 9780060859749, "title": "After Alice: A Novel", "author": "Gregory Maguire", "category": "Fiction", "year": 2016})`
- Insert many document
 - **db.collection.insertMany([<document1>, <document2>, ...])**
- Example
 - `db.books.insertMany([{"isbn": 9780198321668, "title": "Romeo and Juliet", "author": "William Shakespeare", "category": "Tragedy", "year": 2008}, {"isbn": 9781505297409, "title": "Treasure Island", "author": "Robert Louis Stevenson", "category": "Fiction", "year": 2014}])`

CRUD - Update

- Update specific fields of a single document that match the query condition
 - **db.collection.update**(<query>, <update>)
- Example
 - `db.books.update({title : "Treasure Island"}, {$set : {category : "Adventure Fiction"}})`
- Remove certain fields of all documents that match the query condition
 - **db.collection.update**(<query>, <update>, {multi:true})
- Example
 - `db.books.update({category : "Fiction"}, {$unset : {category:""}}, {multi:true})`

CRUD - Delete

- Delete a single document that match the query condition
 - **db.collection.remove(<query>, {justOne:true})**
- Example
 - `db.books.remove({title : "Treasure Island"}, {justOne:true})`
- Delete all documents matching a query condition
 - **db.collection.remove(<query>)**
- Example
 - `db.books.remove({ "category" : "Fiction" })`

CRUD - Query

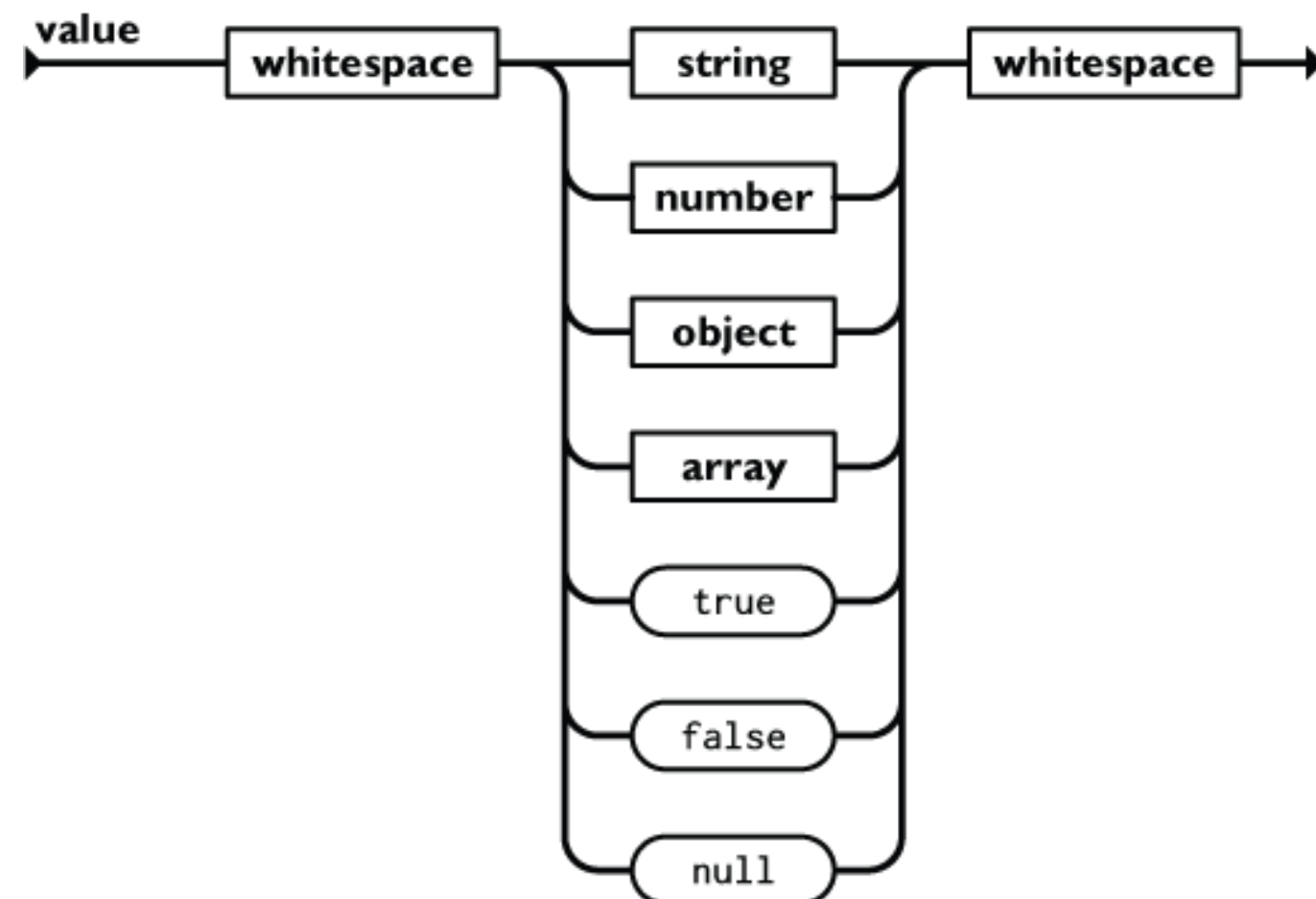
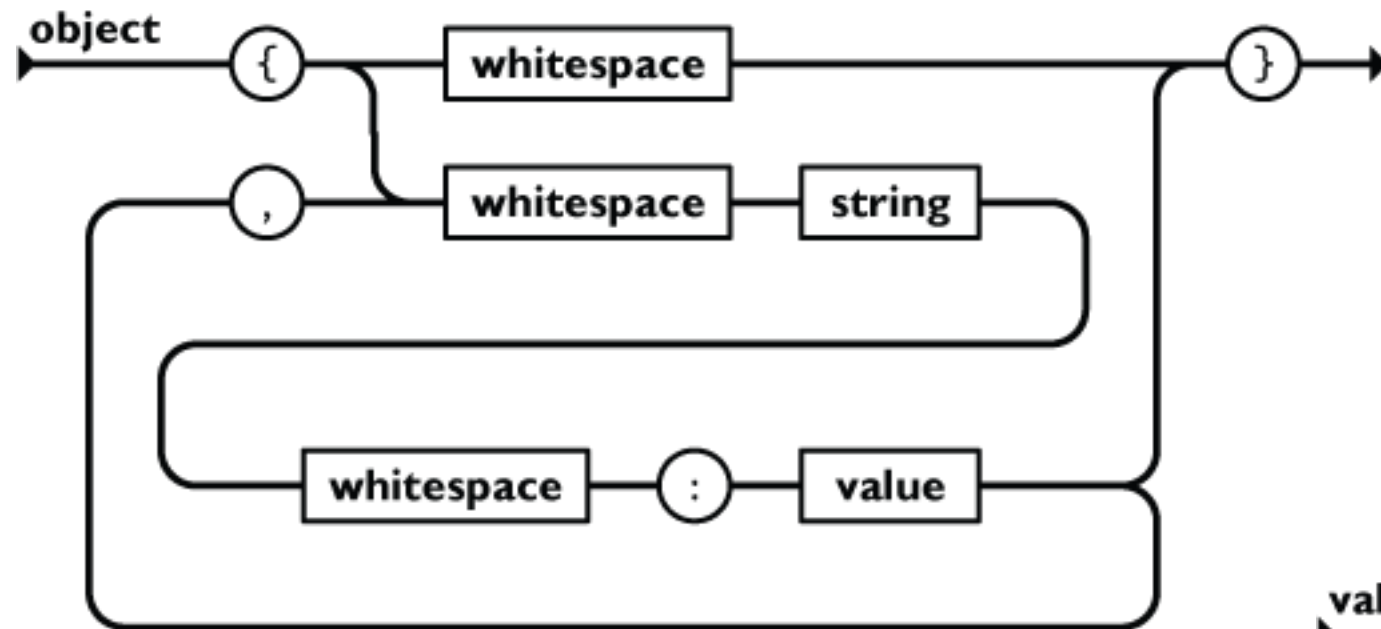
- Filter documents by field value condition
 - **db.collection.find**(<query>)
- Example
 - `db.books.remove({title : "Treasure Island"}, {justOne:true})`

SQL vs MongoDB Mapping

SQL Term	Mongo DB
database	database
Table	Collection
Row	Document
Column	Field
Primary Key	Primary Key set to <code>_id</code>
aggregation (e.g. group by)	aggregation pipeline
SELECT INTO NEW_TABLE	<u>\$out</u>
MERGE INTO TABLE	<u>\$merge</u>
transactions	<u>transactions</u>

<https://docs.mongodb.com/manual/reference/sql-comparison/>

JSON - JavaScript Object Notation



<https://www.json.org>

Connect to DB

```
[cjundang@cjundang:~$ mongo testdb
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017/testdb
MongoDB server version: 4.0.12
WARNING: shell and server versions do not match
Server has startup warnings:
2019-08-17T10:05:00.365+0000 I STORAGE [initandlisten]
2019-08-17T10:05:00.365+0000 I STORAGE [initandlisten] ** WARNING:
d with the WiredTiger storage engine
2019-08-17T10:05:00.365+0000 I STORAGE [initandlisten] **
lesystem
2019-08-17T10:05:01.968+0000 I CONTROL [initandlisten]
2019-08-17T10:05:01.968+0000 I CONTROL [initandlisten] ** WARNING:
2019-08-17T10:05:01.968+0000 I CONTROL [initandlisten] **
is unrestricted.
2019-08-17T10:05:01.968+0000 I CONTROL [initandlisten]
[> db
testdb
[>
[>
```

Insert

```
[> db.cars.insert({name: "Audi", price: 52642})  
WriteResult({ "nInserted" : 1 })
```

name	price
Audi	52642
Mercedes	57127
Skoda	9000
Volvo	29000
Bentley	350000
Citroen	21000
Hummer	41400
Volkswagen	21600

Query

select * from cars

```
[> db.cars.find()
{ "_id" : ObjectId("5d5817e2f153840eeaff29a0"), "name" : "Audi", "price" : 52642 }
{ "_id" : ObjectId("5d5817e7f153840eeaff29a1"), "name" : "Mercedes", "price" : 57127 }
{ "_id" : ObjectId("5d5817ecf153840eeaff29a2"), "name" : "Skoda", "price" : 9000 }
{ "_id" : ObjectId("5d5817f0f153840eeaff29a3"), "name" : "Volvo", "price" : 29000 }
{ "_id" : ObjectId("5d5817f4f153840eeaff29a4"), "name" : "Bentley", "price" : 350000 }
{ "_id" : ObjectId("5d5817faf153840eeaff29a5"), "name" : "Citroen", "price" : 21000 }
{ "_id" : ObjectId("5d581801f153840eeaff29a6"), "name" : "Hummer", "price" : 41400 }
{ "_id" : ObjectId("5d581807f153840eeaff29a7"), "name" : "Volkswagen", "price" : 21600 }
[>
```

select * from cars where name = "Audi"

```
[> db.cars.find({name: 'Audi'})
{ "_id" : ObjectId("5d5817e2f153840eeaff29a0"), "name" : "Audi", "price" : 52642 }
[>
```

select * from cars where name in ("Audi","Skoda")

```
[> db.cars.find({name: { "$in": ["Audi", "Skoda"]} })
{ "_id" : ObjectId("5d5817e2f153840eeaff29a0"), "name" : "Audi", "price" : 52642 }
{ "_id" : ObjectId("5d5817ecf153840eeaff29a2"), "name" : "Skoda", "price" : 9000 }
[>
```

Query

select * from cars where name = "Audi" and price > 5000

```
[> db.cars.find({name:"Audi", price : {$gt: 50000 } })
{ "_id" : ObjectId("5d5817e2f153840eeaff29a0"), "name" : "Audi", "price" : 52642 }
[>
```

select * from cars where name = "Audi" or price > 5000

```
[> db.cars.find({$or:[{name:"Audi"}, {price : {$gt: 50000 } }] })
{ "_id" : ObjectId("5d5817e2f153840eeaff29a0"), "name" : "Audi", "price" : 52642 }
{ "_id" : ObjectId("5d5817e7f153840eeaff29a1"), "name" : "Mercedes", "price" : 57127 }
{ "_id" : ObjectId("5d5817f4f153840eeaff29a4"), "name" : "Bentley", "price" : 350000 }
[>
```

select * from cars where name like "A*"

```
[> db.cars.find({name : /^A/})
{ "_id" : ObjectId("5d5817e2f153840eeaff29a0"), "name" : "Audi", "price" : 52642 }
[>
```

Query

select * from cars where name = "Audi" and price > 5000

```
[> db.cars.find({name:"Audi", price : {$gt: 50000 } })
{ "_id" : ObjectId("5d5817e2f153840eeaff29a0"), "name" : "Audi", "price" : 52642 }
[>
```

select * from cars where name = "Audi" or price > 5000

```
[> db.cars.find({$or:[{name:"Audi"}, {price : {$gt: 50000 } }] })
{ "_id" : ObjectId("5d5817e2f153840eeaff29a0"), "name" : "Audi", "price" : 52642 }
{ "_id" : ObjectId("5d5817e7f153840eeaff29a1"), "name" : "Mercedes", "price" : 57127 }
{ "_id" : ObjectId("5d5817f4f153840eeaff29a4"), "name" : "Bentley", "price" : 350000 }
[>
```

select * from cars where name like "A*"

```
[> db.cars.find({name : /^A/})
{ "_id" : ObjectId("5d5817e2f153840eeaff29a0"), "name" : "Audi", "price" : 52642 }
[>
```

select name from cars where name like "A*"

```
[> db.cars.find({name : /^A/}, {name:1})
{ "_id" : ObjectId("5d5817e2f153840eeaff29a0"), "name" : "Audi" }
[>
```

Query

```
> var c = db.cars.find({$or:[{name:"Audi"}, {price : {$gt: 50000 }}] })
> while (c.hasNext()){ print (tojson(c.next())) }
{
  "_id" : ObjectId("5d5817e2f153840eeaff29a0"),
  "name" : "Audi",
  "price" : 52642
}
{
  "_id" : ObjectId("5d5817e7f153840eeaff29a1"),
  "name" : "Mercedes",
  "price" : 57127
}
{
  "_id" : ObjectId("5d5817f4f153840eeaff29a4"),
  "name" : "Bentley",
  "price" : 350000
}
```


Query

```
[> var c = db.cars.find({$or:[{name:"Audi"}, {price : {$gt: 50000 }}] })
[> c.forEach(printjson)
{
  "_id" : ObjectId("5d5817e2f153840eeaff29a0"),
  "name" : "Audi",
  "price" : 52642
}
{
  "_id" : ObjectId("5d5817e7f153840eeaff29a1"),
  "name" : "Mercedes",
  "price" : 57127
}
{
  "_id" : ObjectId("5d5817f4f153840eeaff29a4"),
  "name" : "Bentley",
  "price" : 350000
}
```

Update

- `db.collection.updateOne(<filter>, <update>, <options>)`
- `db.collection.updateMany(<filter>, <update>, <options>)`
- `db.collection.replaceOne(<filter>, <update>, <options>)`

```
[> db.cars.find()
{ "_id" : ObjectId("5d5817e2f153840eeaff29a0"), "name" : "Audi", "price" : 52642 }
{ "_id" : ObjectId("5d5817e7f153840eeaff29a1"), "name" : "Mercedes", "price" : 57127 }
{ "_id" : ObjectId("5d5817ecf153840eeaff29a2"), "name" : "Skoda", "price" : 9000 }
{ "_id" : ObjectId("5d5817f0f153840eeaff29a3"), "name" : "Volvo", "price" : 29000 }
{ "_id" : ObjectId("5d5817f4f153840eeaff29a4"), "name" : "Bentley", "price" : 350000 }
{ "_id" : ObjectId("5d5817faf153840eeaff29a5"), "name" : "Citroen", "price" : 21000 }
{ "_id" : ObjectId("5d581801f153840eeaff29a6"), "name" : "Hummer", "price" : 41400 }
{ "_id" : ObjectId("5d581807f153840eeaff29a7"), "name" : "Volkswagen", "price" : 21600 }
[>
[> db.cars.update({name:'Hummer'}, {$set:{price:100}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
[>
[> db.cars.find({name:'Hummer'})
{ "_id" : ObjectId("5d581801f153840eeaff29a6"), "name" : "Hummer", "price" : 100 }
[>
```


Delete

```
[> db.cars.remove({name:'Hummer'})
WriteResult({ "nRemoved" : 1 })
[> db.cars.find()
{ "_id" : ObjectId("5d5817e2f153840eeaff29a0"), "name" : "Audi", "price" : 52642 }
{ "_id" : ObjectId("5d5817e7f153840eeaff29a1"), "name" : "Mercedes", "price" : 57127 }
{ "_id" : ObjectId("5d5817ecf153840eeaff29a2"), "name" : "Skoda", "price" : 9000 }
{ "_id" : ObjectId("5d5817f0f153840eeaff29a3"), "name" : "Volvo", "price" : 29000 }
{ "_id" : ObjectId("5d5817f4f153840eeaff29a4"), "name" : "Bentley", "price" : 350000 }
{ "_id" : ObjectId("5d5817faf153840eeaff29a5"), "name" : "Citroen", "price" : 21000 }
{ "_id" : ObjectId("5d581807f153840eeaff29a7"), "name" : "Volkswagen", "price" : 21600 }
[
```

Connect MongoDB with PHP

<https://www.php.net/manual/en/set.mongodb.php>

Connect & Query

```
$mng = new MongoDB\Driver\Manager("mongodb://localhost:27017");  
$query = new MongoDB\Driver\Query([]);  
$rows = $mng->executeQuery("testdb.cars", $query);  
echo "<pre>";  
foreach ($rows as $row) {  
    echo "$row->name : $row->price\n";  
}  
echo "</pre>";
```

Insert

```
$mng = new MongoDB\Driver\Manager("mongodb://localhost:27017");  
$bulk = new MongoDB\Driver\BulkWrite;  
$doc = [  
    '_id' => new MongoDB\BSON\ObjectId,  
    'name' => 'Toyota',  
    'price' => 26700  
];  
$bulk->insert($doc);  
$mng->executeBulkWrite('testdb.cars', $bulk);
```

Query with filters

```
$mng = new MongoDB\Driver\Manager("mongodb://localhost:27017");  
$filter = [ 'name' => 'Volkswagen' ];  
$query = new MongoDB\Driver\Query($filter);  
$res = $mng->executeQuery("testdb.cars", $query);  
$car = current($res->toArray());  
echo "<pre>";  
if (!empty($car)) {  
    echo $car->name, ": ", $car->price, PHP_EOL;  
} else {  
    echo "No match found\n";  
}  
echo "</pre>";
```

Filters

select * from cars where name = "Audi"

```
$filter = [  
  'name' => 'Audi'  
];
```

select * from cars where name in ("Audi","Skoda")

```
$filter = [  
  'name' => [  
    '$in' => [ 'Audi', 'Skoda' ]  
  ]  
];
```

Filters

select * from cars where name = "Audi" and price > 5000

```
$filter = [  
  'name' => "Audi",  
  'price' => ['$gt' => 5000 ]  
];
```

select * from cars where name = "Audi" or price > 5000

```
$filter = [  
  '$or'=>[  
    ['name' => "Audi"],  
    ['price' => ['$gt' => 50000 ]],  
  ],  
];
```

Update

```
$mng = new MongoDB\Driver\Manager("mongodb://localhost:27017");  
$bulk = new MongoDB\Driver\BulkWrite;  
$bulk->update(  
    ['name' => "Toyota"],  
    ['$set' => ['price' => "9"]],  
    ['multi' => false, 'upsert' => false]  
);  
$result = $mng->executeBulkWrite('testdb.cars', $bulk);  
var_dump($result->getMatchedCount());
```


Delete

```
$manager = new MongoDB\Driver\Manager( 'mongodb://localhost:27017' );  
  
$bulk = new MongoDB\Driver\BulkWrite;  
$filter = [ 'name' => "Toyota" ];  
$option = [ 'limit' => 0 ];  
$bulk->delete( $filter, $option );  
$result = $manager->executeBulkWrite( 'testdb.cars', $bulk );  
print_r( $result );
```

กิจกรรม (1)

- ให้สร้าง class สำหรับการ CRUD collection ของ mongodb ตาม interface ที่กำหนดให้ต่อไปนี้ พร้อมทั้งสร้าง driven program

```
<?php
```

```
class MyMongo{  
    private $manager;  
    private $collection;  
    function __construct($collection);  
    public function getAll();  
    public function getWithFilters($filter);  
    public function add($document);  
    public function update($filter, $update, $option);  
    public function delete($filter, $option);  
}
```

กิจกรรม (2)

- เรียกใช้งาน php class ผ่าน slim framework

กิจกรรม (3)

- สร้างหน้า page สำหรับการเรียกใช้งาน slim framework เพื่อจัดการฐานข้อมูลต่อไปนี้

Colums	คำอธิบาย	ตัวอย่าง
std_id	รหัสนักศึกษา	58211029
f_name	ชื่อ	Chanankorn
l_name	นามสกุล	Jandaeng
gpa	เกรดเฉลี่ย	3.75

