



WQD7007 Big Data Management

NoSQL and MongoDB

NoSQL

- NoSQL → Not Only SQL
 - being **non-relational**, **distributed**, and **open source**
- Characteristics:
 - schema-free
 - easy replication support
 - simple API
 - Big data
 - Commodity
 - Flexible data model

MongoDB

- A document-based (e.g. JSON or BSON) database.
- Characteristics:
 - Expressive query languages and secondary indexes
 - Strong consistency
 - Enterprise Management and Integration
 - Flexibility
 - Scalability and performance
 - Always on, global deployments

JSON and BSON

- Java Script Object Notation (JSON) is the lightweight data-interchange format to exchange data.
- Binary JSON → BSON: JSON that is serialized as a binary document
- JSON build on two structures:
 - A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
 - An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

Example of JSON:

```
{
  "Rail Booking": {
    "reservation": {
      "ref_no": 1234567,
      "time_stamp": "2016-06-24T14:26:59.125",
      "confirmed": true
    },
    "train": {
      "date": "07/04/2016",
      "time": "09:30",
      "from": "New York",
      "to": "Chicago",
      "seat": "57B"
    },
    "passenger": {
      "name": "John Smith"
    },
    "price": 1234.25,
    "comments": ["Lunch & dinner incl.", "\"Have a nice day!\""]
  }
}
```

Why JSON and not XML?

- XML is not well suited to data-interchange. It carries a lot of baggage, and it doesn't match the data model of most programming languages.

XML

```
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

JSON

```
{ "empinfo" :
  {
    "employees" : [
      {
        "name" : "James Kirk",
        "age" : 40,
      },
      {
        "name" : "Jean-Luc Picard",
        "age" : 45,
      },
      {
        "name" : "Wesley Crusher",
        "age" : 27,
      }
    ]
  }
}
```

MongoDB VS RDBMS

MongoDB	RDBMS
Database	Database
Collection	Table
Document	Tuple/Row
Field	Column
Embedded Documents	Table Join

From RDBMS to MongoDB

Relational

Person:

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rom

Car:

Car_ID	Model	Year	Value	Pers_ID
101	Bentley	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

no relation



MongoDB Document

```
{
  first_name: 'Paul',
  surname: 'Miller'
  city: 'London',
  location: [45.123,47.232],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, ... },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, ... }
  ]
}
```


Benefits of using JSON (document model)

- Document maps nicely to programming languages data type
- Embedded documents and arrays reduces the need for joins

MongoDB vs HBase

MongoDB	HBase
In MongoDB data is stored in JSON format. We have to insert/update/fetch the document as a whole	In HBase we can work on partial data.
MongoDB relies on the OS to buffer its data and maps the whole data to memory with mmap.	HBase use the log-structured merge tree, which probably is more efficient for heavy load and big data.
MongoDB supports secondary index and the index should be keep in memory.	HBase requires a third party module to support secondary index.
MongoDB has good feature of searching a document data using find/findOne by its inner attribute. The attribute can be indexed to search faster.	HBase do not have this feature.
Examples: Product/catalog data, geospatial data, Aadhar biometric database, ebay search suggestion	Examples: storing genome sequences, sports match histories, time-series data, log files

Installation

- Download MongoDB in <https://www.mongodb.org/downloads#production>
 - If you're using Mac, you can use Homebrew
 - For Windows:
 - This lab assumes that you install it to C:\Program Files\MongoDB. Add the bin directory to your System Environment Variables
 - If you're using any other operating systems, please refer to this <https://docs.mongodb.org/manual/installation/> for more info
- During installation, you'll be required to install it to a PATH. Remember the path that you installed MongoDB to.

Running MongoDB Server

- Before you proceed, you will need to create a directory which stores your MongoDB data.
 - It is always /data/db by default. (For Windows, it's C://data/db/.) Create that data directory.
 - If you can't create the directory in Mac or Linux, use “sudo” privileges
- Finally, head to your terminal and type `mongod --smallfiles --noprealloc` to start your MongoDB Server.
 - If it states “no such command is recognized”, that means you have not added the bin directory into your PATH
 - If you use “sudo” privileges to create path, please use “sudo” privileges for this command too

Run MongoDB

- open a new terminal (or a tab), and type mongo.
- Basic commands:
 - **show dbs**: Show the names of all databases
 - **show collections**: Show the names of all collections in the database
 - **use dbname**: Change to the database named dbname
 - **db**: Returns the name of the current database that you are connected to

Run MongoDB

```

Wais-MacBook-Air:~ wailamhoo$ mongo
MongoDB shell version v3.6.4
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.4
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:
2018-05-15T13:49:07.234+0800 I CONTROL [initandlisten]
2018-05-15T13:49:07.234+0800 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-05-15T13:49:07.234+0800 I CONTROL [initandlisten] **          Read and write access to data and configuration is unrestricted.
2018-05-15T13:49:07.234+0800 I CONTROL [initandlisten] ** WARNING: You are running this process as the root user, which is not recommended.
2018-05-15T13:49:07.235+0800 I CONTROL [initandlisten]
2018-05-15T13:49:07.235+0800 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.
2018-05-15T13:49:07.235+0800 I CONTROL [initandlisten] **          Remote systems will be unable to connect to this server.
2018-05-15T13:49:07.235+0800 I CONTROL [initandlisten] **          Start the server with --bind_ip <address> to specify which IP
2018-05-15T13:49:07.235+0800 I CONTROL [initandlisten] **          addresses it should serve responses from, or with --bind_ip_all to
2018-05-15T13:49:07.235+0800 I CONTROL [initandlisten] **          bind to all interfaces. If this behavior is desired, start the
2018-05-15T13:49:07.235+0800 I CONTROL [initandlisten] **          server with --bind_ip 127.0.0.1 to disable this warning.
2018-05-15T13:49:07.235+0800 I CONTROL [initandlisten]
2018-05-15T13:49:07.235+0800 I CONTROL [initandlisten]
2018-05-15T13:49:07.235+0800 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
> show collections
> use admin
switched to db admin
> use wlhoo
switched to db wlhoo

```

Run MongoDB

- There will not be any commands to **create a database**. If the database does not exist, say when you do **use random1234**, Mongo will **create a database called random1234** for you. So, be careful in your syntax!
- Mongo by default connects you to the **test database** upon opening a connection.

Create database and Insert Data

- We'll now be using a new database called **firstdb** , and we'll be inserting some entries into the collection named **users**.
 - **use firstdb**
 - **db.users.insert({ name: 'Enter Your Name Here', age: 25 })**
 - **db.users.find().pretty()**

```
[> use firstdb
switched to db firstdb
[> db.users.insert( { name: 'Enter Your Name Here', age: 25 } )
WriteResult({ "nInserted" : 1 })
[> db.users.find().pretty()
{
  "_id" : ObjectId("5afa78457d617295fea6c1a7"),
  "name" : "Enter Your Name Here",
  "age" : 25
}
```


Import JSON file

- Assume you download the JSON data, stored it in Desktop and wish to import to MongoDB:
- **mongoimport -d school -c students < ~/Desktop/student.json**

```
Wais-MacBook-Air:~ wailamhoo$ mongoimport -d school -c students < ~/Desktop/students.json
2018-05-15T14:11:53.352+0800    connected to: localhost
2018-05-15T14:11:53.422+0800    imported 200 documents
Wais-MacBook-Air:~ wailamhoo$
```

- Enter mongoDB by typing **mongo**. Then, switch to *school* database by typing **use school**

Inserting data

Try insert these documents!

Id	Name	Scores
200	Drake	[]
201	Logic	[{"type": "homework", "score": 89}, {"type": "quiz", "score": 56}]
202	Kraimir	[{"type": "homework", "score": 11}]

Finding documents

Example: Display the details of the student named “Brain Lachapelle”

```
[> db.students.find({"name":"Brain Lachapelle"}).pretty()
{
  "_id" : 86,
  "name" : "Brain Lachapelle",
  "scores" : [
    {
      "type" : "exam",
      "score" : 2.013473187690951
    },
    {
      "type" : "quiz",
      "score" : 45.01802394825918
    },
    {
      "type" : "homework",
      "score" : 63.74658824265818
    },
    {
      "type" : "homework",
      "score" : 88.04712649447521
    }
  ]
}
```

Finding documents

1. Obtain the number of students who achieved a score greater than 95 of any type.
2. Display the student who achieved the highest score on any type in the whole school
3. Display the `_id` and name of the first 10 students (arranged in alphabetical order by name) who achieved a score between 10 and 35 of any type
4. What is the number of students whose names are between C and E lexically? Also, display only the name of 10 of these students, in descending order

Updating and removing documents

- Update from one value to another
 - **`db.users.update({"username": "admin123"}, {$set: {"password": "secret"}})`**
- Removes "profile ID" attribute
 - **`db.users.update({"username": "admin123"}, {$unset: {"profile_id": 1}})`**
- Increments "login_count" value by 1
 - **`db.users.update({"username": "admin123"}, {$inc: {"login_count": 1}})`**
- Insert a new permission in "permission" array
 - **`db.users.update({"username": "admin123"}, {$push: {"permissions": "add_blog"}})`**

Updating and removing documents

- Removes the first element from “permission” array:
 - **db.users.update({"username":"admin123"}, {\$pop: {"permissions": -1}})**
- Removes the specified element from “permission” array
 - **db.users.update({"username":"admin123"}, {\$pull: {"permissions": "edit_bg"}})**
- Insert the specified element to “permission” array
 - **db.users.update({"username":"admin123"}, {\$addToSet: {"permissions": "edit_blog"}})**
- Remove user
 - **db.users.remove({"username":"admin123"})**