BIG DATA: TECHNOLOGIES AND APPLICATIONS

C13 MongoDB

Il-Yeol Song, Ph.D.
College of Computing & Informatics
Drexel University
Philadelphia, PA 19104

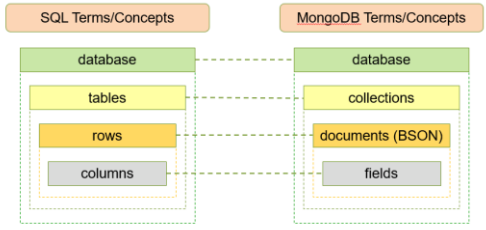
DREXEL UNIVERSITY College of Computing & Informatics

Contents

MongoDB Introduction

- MongoDB is from "Humongous"
- A document-based NoSQL database by MongoDB, Inc.
  - Documents are stored as **JSON** files, this makes it easier to read and manipulate using different programming languages.
  - Schemaless
- Released in 2009.
- Within MongoDB
  - **Data** is stored in **documents in JSON**.
  - **Documents** of a similar type are stored in **collections**.
  - Related **collections** are stored in a **database**.

RDBMS and MongoDB



JSON

- JavaScript Object Notation (JSON) – data interchange format used to represent data as a logical object.
- Every JSON document requires an object ID.
- **Object** is enclosed by a pair of **curly brackets** {**K**:**V**}
- Array is enclosed by []
- Example:

```
{_id: 101, title: "Database Systems", author: ["Coronel", "Morris"]}
```

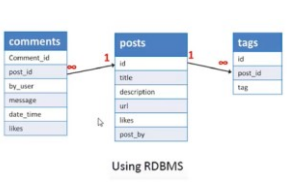
```
{
_id: 101,
title: "Database Systems",
author: ["Coronel", "Morris"]
}
```


NoSQL databases sacrifice redundancy to improve scalability

```
{
_id: 101,
title: "Database Systems",
author: ["Coronel", "Morris"],
publisher: {
        name: "Cengage",
        street: "500 Topbooks Avenue",
        city: "Boston",
        state: "MA"
        }
}
```

```
{
_id: 101,
title: "Database Systems",
author: [
        {
        name: "Coronel",
        email: "ccoronel@mtsu.edu",
        phone: "6155551212"
        },
        {
        name: "Morris",
        email: "smorris@mtsu.edu",
        office: "301 Codd Hall"
        }
        ],
publisher: {
    name: "Cengage",
    address: {
        street: "500 Topbooks Avenue",
        city: "Boston",
        state: "MA"
            }
        }
}
```

MongoDB represent Aggregated Objects



Using RDBMS

```
{
  _id:          POST_ID
  title:        TITLE_OF_POST,
  description:                POST_DESCRIPTION,
  by:           POST_BY,
  url:          URL_OF_POST,
  tags: [       TAG1, TAG2, TAG3],
  likes:        TOTAL_LIKES,
  comments: [
              {
                  user:'COMMENT_BY',
                  message: TEXT,
                  dateCreated: DATE_TIME,
                  like: LIKES
              },
              {
                  user:'COMMENT_BY',
                  message: TEXT,
                  dateCreated: DATE_TIME,
                  like: LIKES
              }
            ]
}
```

Using MongoDB

Syntactic Rules in MongoDB

- MongoDB is ***Case-Sensitive*** – Capitalization matters.
- Semi-colons are not required.
- **All string data** being saved should be in ***double quotes***.
- Commands are **space-independent**.
  - Example:-

```
> db.Employee.update(
...{ "Employeeid" : 2},
...{ $set: {"Employeeid" : "NewMartin"}});
```

The compiler ignores spaces

- Comments are indicated by //

```
//Retrieve comments by user
db.comments.find({'_userId' : someUserId})
```

- Data is displayed in the order of insertion order.
- The field names **cannot** start with the $ character.
- The field names **cannot** contain the . character.

Create a Database

- **Database –** It is a container for collections. A MongoDB server can store multiple databases. Each database has its own group of files.
- **Create a database command** – "**use**"
  - Creates a new database if a database in that name doesn't exist. Once created, it switches to the created database.

use <database_name>

```
> use EmployeeDB
```

Name of the Database

Output - Database is created

```
switched to db EmployeeDB
```

- Databases can be displayed by **"show dbs "** command.

"show dbs" Command

- To retrieve the list of databases available on the server, type the command –**show dbs**
- After the installation of the software, MongoDB only includes an admin and a local database.

```
mongo
> show dbs
admin  0.000GB
local  0.000GB
>
```

  - **Admin Database** – Records data on database administration issues like users, roles, and privileges for the databases hosted on the server.
  - **Local Database** – Stores data about the server's start-up process and the server's role in sharing operations.
  - The Admin and Local database will not store any end-user data.
- **Note:-** use method **db.getName()** to display the database being

Create a Collection

- To create a collection, use createCollection() method:

  createCollection("collectionName")
- Use the **db** variable with the above method
- Example –

db.createCollection("newproducts")

  - Creates a collection named "newproducts" inside the previously defined demo database.

```
> use demo
switched to db demo
> db.createCollection("newproducts")
{ "ok" : 1 }
```

- Collections can be displayed by **"show collections"** command.

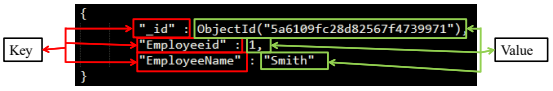
```
> show collections
newproducts
```


Dropping a Collection

- To drop a collection, we use the method – **drop()**
- This method is used to delete all the documents present in the collection as well as any indexes that have been created with that collection.
- Example:-
  - Removes the collection "newproducts" along with all its documents:
    - **db.newproducts.drop()**
  - Delete all the documents in the collection
    - **db.newproducts.remove({})**

Common Terms in MongoDB

- **Field** – A key-value pair in a document is denoted as a field. **All key names may optionally have quotes, while text data entered as value should be written in double quotes.**



- **_id**
  - A mandatory field for every document.
  - Serves as the ***primary key*** of the document.
  - Its value must be unique in the collection.
  - If you do not assign a value to this variable, MongoDB will automatically assign a value.

CRUD Operations

- Create
  - db.collection.insert(<document>)
  - db.collection.insertOne(<document>)
  - db.collection.insertMany([{d1}, {d2}.., {d3}])
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)

CRUD Operations

- Update
  - db.collection.update(<query>, <update>, <options>)
  - db.collection.updateOne(<query>, <update>, <options>)
  - db.collection.updateMany(<query>, <update>, <options>)
  - db.collection.replaceOne(<query>, <replacement>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)
  - db.collection.deleteOne(<query>, <options>)
  - db.collection.deleteMany(<query>, <options>)
  - db.collection.deleteMany({}) //delete all documents

CRUD example

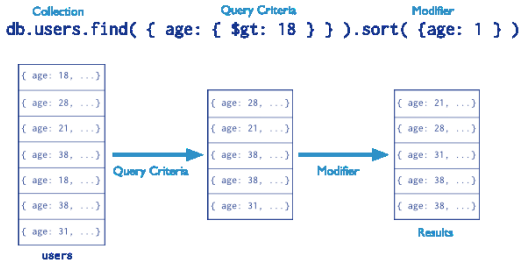
```
> db.user.insert({
      first: "John",
      last : "Doe",
      age: 39
})
```

```
> db.user.find ()
{
      "_id" : ObjectId("51..."),
      "first" : "John",
      "last" : "Doe",
      "age" : 39
}
```

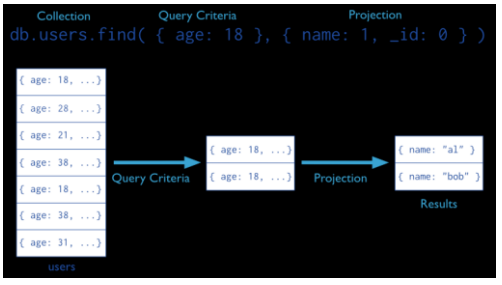
```
> db.user.update(
      {"_id" :
ObjectId("51...")},
      { $set: {
            age: 40,
            salary: 7000}
      }
)
```

```
> db.user.remove({
      "first": /^J/
})
```


Query Statements



Projections



CREATE Operation: Inserting documents

- Use the method:
  - **db.<<collectionName>>.insert({document})**
  - **db.<<collectionName>>.insertOne({document})**
  - **db.<<collectionName>>.insertMany([{document1}, {document2}])**
- Example:-

```
db.users.insertOne(          <-- collection
   {
      name: "sue",           <-- field: value
      age: 26,               <-- field: value
      status: "pending"      <-- field: value
   }                                  } document
)
```

```
>>> db.emp.insert({id: 1, name: "Song"})
WriteResult({ "nInserted" : 1 })
>>>
```


Inserting Multiple documents

- Use the method – **insertMany([{d1}, {d2}.., {dn}])**
- Example:-

```
db.inventory.insertMany([
   { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },
   { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
   { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm" } }
])
```


SQL and MongoDB: CREATE/INSERT

```
CREATE TABLE users (
    id MEDIUMINT NOT NULL
        AUTO_INCREMENT,
    user_id Varchar(30),
    age Number,
    status char(1),
    PRIMARY KEY (id)
)
```

Implicitly created on first `insert` operation. The primary key `_id` is automatically added if `_id` field is not specified.

```
db.users.insert( {
    user_id: "abc123",
    age: 55,
    status: "A"
  } )
```

However, you can also explicitly create a collection:

```
db.createCollection("users")
```

```
db.people.insertOne(
   { user_id: "bcd001", age: 45, status: "A" }
)
```


READ with find() method

- Syntax - **find({<query>}, {<projection>})**
  - Both objects are optional
  - ***If only one object parameter is written, MongoDB assumes it belongs to the query object parameter***
  - **{<query>} is the same as WHERE clause**
  - **{<projection>} is the same as SELECT clause**
  - **find({<WHERE>}, {<SELECT>})**
  - When the query object is not needed but projection object is needed, an empty object must be used as a query object
- **Retrieve the display field of every document**

  SELECT display FROM patron;

  db.patron.find({}, {display:1, _id:0})

READ with find() method

- Syntax - **find({<query>}, {<projection>})**

```
db.users.find(                    <-- collection
   { age: { $gt: 18 } },          <-- query criteria
   { name: 1, address: 1 }        <-- projection
).limit(5)                        <-- cursor modifier
```

- **The value with each key in the projection object** is either the value 0 or 1.
  - **'1'** –the key:value pair should be included in the results.
  - **'0'** –the key:value pair should be omitted in the results.
- **Retrieve the *display* field and suppress _id field**

  SELECT display FROM patron;

  db.patron.find({}, {display:1, _id:0})

READ with find() method: Example

```
SELECT *
FROM
   [users]
WHERE
   name = "Johan A"
```

```
db.users.find
  (
    {
       name: "Johan A"
    }
  )
```

READ Operation: Document Retrieval

- To retrieve and display the documents present in the collection, use the method – **find()**
- Example:-

```
> db.products.find()
```

  - Output:-

```
{ "_id" : ObjectId("598e01613ae3ad8abf1b8300"), "name" : "standard desk chair",
"price" : 150, "brand" : "CheapCo", "type" : "chair" }
```


READ Operation: pretty() method

- To improve the readability of the retrieved document, use the method – **pretty()**
- Example:-

```
> db.products.find().pretty()
```

```
{
        "_id" : ObjectId("598e01613ae3ad8abf1b8300"),
        "name" : "standard desk chair",
        "price" : 150,
        "brand" : "CheapCo",
        "type" : "chair"
}
```


Example: find()

- Find display and type of "Robert Carter"

SELECT display, type FROM patron
WHERE display = "Robert Carter";

db.patron.find({display: "Robert Carter"}, {display:1, type:1})

- Find display of "Faculty" patron without object ID

SELECT display FROM patron WHERE type = "faculty";

db.patron.find({type: "faculty"}, {display:1, _id:0})

SQL and MongoDB: SELECT

```
SELECT * FROM users WHERE age>33
db.users.find({age:{$gt:33}})

SELECT * FROM users WHERE age!=33
db.users.find({age:{$ne:33}})

SELECT * FROM users WHERE a=1 and b='q'
db.users.find({a:1,b:'q'})

SELECT * FROM users WHERE a=1 or b=2
db.users.find( { $or : [ { a : 1 } , { b : 2 } ] } )

SELECT * FROM foo WHERE  name='bob' and (a=1 or b=2 )
db.foo.find( { name : "bob" , $or : [ { a : 1 } , { b : 2 } ] } )

SELECT * FROM users WHERE age>33 AND age<=40
db.users.find({'age':{$gt:33,$lte:40}})
```

- **MongoDB operators start with $.**
- Other Query operators:
- $ne, $gt, $gte $lt, $lte,
- $and, $or, $not
- $exists, and $regex.

Comparison Operators

Name	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$in	Matches any of the values specified in an array.
$nin	Matches none of the values specified in an array.

https://docs.mongodb.com/v3.0/reference/operator/query/

Logical Operators

Name	Description
$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.
$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
$not	Inverts the effect of a query expression and returns documents that do *not* match the query expression.
$nor	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.

https://docs.mongodb.com/v3.0/reference/operator/query/

Queries Using Inequalities

Example – *retrieves the display name and age from documents for patrons that are age 30 or less and sorts them in descending order of age.*

SELECT display, age FROM patron WHERE age <= 30
ORDER BY age DESC;

- **db.patron.find({age: {$lte:30} }, {display:1, age:1}).sort({age: −1})**

```
db.patron.find({age: {$lte:30} }, {display:1, age:1}).sort({age: -1});
{ "_id" : ObjectId("598e0649b4615ba6815141ce"), "display" : "Jimmie Love", "age" : 29 }
{ "_id" : ObjectId("598e0649b4615ba6815141dd"), "display" : "Desiree Harrington", "age" : 28 }
{ "_id" : ObjectId("598e0649b4615ba6815141d6"), "display" : "Keith Cooley", "age" : 27 }
{ "_id" : ObjectId("598e0649b4615ba6815141d1"), "display" : "Holly Anthony", "age" : 25 }
{ "_id" : ObjectId("598e0649b4615ba6815141d4"), "display" : "Iva Ramos", "age" : 24 }
{ "_id" : ObjectId("598e0649b4615ba6815141ea"), "display" : "Betsy Malone", "age" : 24 }
{ "_id" : ObjectId("598e0649b4615ba6815141d5"), "display" : "Rena Mathis", "age" : 23 }
{ "_id" : ObjectId("598e0649b4615ba6815141e2"), "display" : "Zach Kelly", "age" : 23 }
{ "_id" : ObjectId("598e0649b4615ba6815141e5"), "display" : "Wilfred Fuller", "age" : 23 }
{ "_id" : ObjectId("598e0649b4615ba6815141e6"), "display" : "Jeff Owens", "age" : 23 }
{ "_id" : ObjectId("598e0649b4615ba6815141e7"), "display" : "Homer Goodman", "age" : 23 }
{ "_id" : ObjectId("598e0649b4615ba6815141eb"), "display" : "Tony Miles", "age" : 23 }
```


Combining criteria with *implicit logical and*

- Example of **implicit logical and** – *finds the patrons who are under 21 years of age **and** have checked out a book published in 2017*
  - **SELECT * FROM patron**
    WHERE age < 21 AND checkout.pubyear = 2017;
  - **db.patron.find({age:{$lt:21}, "checkouts.pubyear":2017}).pretty()**
- A comma between two conditions is treated as an **implicit logical and.**

Combining criteria with both $and and $or

- The **$and** and **$or** functions can be mixed within the same query object.
- Example – *retrieves the _id, display name, and age for patrons that either have the last name "barry" and are faculty, or have the last name "hays" and are under 30 years old.*

- SELECT display, age, type FROM patron WHERE
  (lname ="barry" AND type = "faculty") OR
  (lname = "hays AND age > 30);

  - **db.patron.find({$or: [**
    {$and: [{lname: "barry"}, {type: "faculty"}]},
    {$and: [{lname: "hays"}, {age: {$lt: 30}}]}
    ]},
    {display: 1, age: 1, type: 1}).pretty()

SQL and MongoDB: Aggregation: count()

```
SELECT COUNT(*)
FROM users
```

```
db.users.count()
```

```
db.users.find().count()
```

```
SELECT COUNT(user_id)
FROM users
```

```
db.users.count( { user_id: { $exists: true } } )
```

```
db.users.find( { user_id: { $exists: true } } ).count()
```

```
SELECT COUNT(*)
FROM users
WHERE age > 30
```

```
db.users.count( { age: { $gt: 30 } } )
```

```
db.users.find( { age: { $gt: 30 } } ).count()
```


SQL and MongoDB: Aggregation: distinct()

```
SELECT DISTINCT(status)
FROM users
```

```
db.users.distinct( "status" )
```


SQL and MongoDB: ORDER BY

```
SELECT *
FROM users
WHERE status = "A"
ORDER BY user_id ASC
```

```
db.users.find( { status: "A" } ).sort( { user_id: 1 } )
```

In sort() method:
(-1) stands for descending order
(1)stands for ascending order

```
SELECT *
FROM users
WHERE status = "A"
ORDER BY user_id DESC
```

```
db.users.find( { status: "A" } ).sort( { user_id: -1 } )
```

UPDATE Operation

- Update operations modify existing documents in a collection
  - db.collection.updateOne()
  - db.collection.updateMany()
  - db.collection.replaceOne()

```
db.users.updateMany(                    <-- collection
   { age: { $lt: 18 } },                <-- update filter
   { $set: { status: "reject" } }       <-- update action
)
```


SQL and MongoDB: UPDATE

```
UPDATE people
SET status = "C"
WHERE age > 25
```

```
db.people.updateMany(
   { age: { $gt: 25 } },
   { $set: { status: "C" } }
)
```

```
UPDATE users
SET age = age + 3
WHERE status = "A"
```

```
db.users.update(
   { status: "A" } ,
   { $inc: { age: 3 } },
   { multi: true }
)
```

```
db.users.updateMany(
   { status: "A" } ,
   { $inc: { age: 3 } }
)
```


DELETE Operation

- Delete operations remove documents from a single collection:

```
db.collection.remove( <query>, <justOne> )
db.collection.deleteOne( <query>, <options> )
db.collection.deleteMany( <query>, <options> )
db.collection.deleteMany({}) //delete all documents
```

- Example:

```
db.users.deleteMany(                    <-- collection
   { status: "reject" }                 <-- delete filter
)
```


SQL and MongoDB: DELETE

```
db.collection.remove( <query>, <justOne> )
db.collection.deleteOne( <query>, <options> )
db.collection.deleteMany( <query>, <options> )
db.collection.deleteMany({}) //delete all documents
```

- Remove the first document from the inventory collection where the status field equals "D":
  db.inventory.deleteOne({ status: "D" })

```
DELETE FROM users
WHERE status = "D"
```

```
db.users.remove( { status: "D" } )
```

- Remove all documents from the inventory collection where the status field equals "A":

```
db.inventory.deleteMany({ status : "A" })
```


DELETE ALL Operation

- To delete all the documents from a collection, pass an empty filter document {} to deleteMany():
- The following example deletes all documents from the inventory collection:
  db.inventory.deleteMany({})
- Using remove({})

```
> db.users.remove({})
WriteResult({ "nRemoved" : 2 })
```


Pattern Matching

- MongoDB uses **PCRE** (Perl Compatible Regular Expression) as regular expression language
- Define the search criteria using the **$regex** operator.
- Or, use **"/pattern/"** in place of the $regex operator. This is known as a delimiter. We specify the pattern we are looking for in-between the delimiters.

Using regex Expression

The following regex query searches for all the posts containing string **tutorialspoint** in it:

```
>db.posts.find({post_text:{$regex:"tutorialspoint"}})
```

The same query can also be written as:

```
>db.posts.find({post_text:/tutorialspoint/})
```

Pattern Matching with wild characters

Wild Characters:

'^' character denotes that the string **starts with** a specific character, while '$' denotes that the string **ends with** a specific character.

- Find posts which has a tag that begins with only "tut"
  SELECT * FROM posts WHERE tags LIKE "tut%";
  db.posts.find({tags: {$regex: "**^tut**"}})

- Find posts which ends with "MongoDB" in the post_text
SELECT * FROM posts WHERE post_text LIKE "%MongoDB";
db.posts.find({post_text: {regex: "/MongoDB$/"}})

SQL and MongoDB: String Manipulation

```
SELECT *
FROM users
WHERE user_id like "%bc%"
```

```
db.users.find(
   { user_id: /bc/ }
)
```

db.users.find({ user_id: { $regex: /bc/ } })

```
SELECT *
FROM users
WHERE user_id like "bc%"
```

```
db.users.find(
   { user_id: /^bc/ }
)
```

db.users.find({ user_id: { $regex: /^bc/ } })

Pattern Matching with Case-Sensitivity

Using regex Expression with Case Insensitive

To make the search case insensitive, we use the **$options** parameter with value **$i**. The following command will look for strings having the word **tutorialspoint**, irrespective of smaller or capital case:

```
>db.posts.find({post_text:{$regex:"tutorialspoint",$options:"$i"}})
```

One of the results returned from this query is the following document which contains the word **tutorialspoint** in different cases:

```
{
   "_id" : ObjectId("53493d37d852429c10000004"),
   "post_text" : "hey! this is my post on TutorialsPoint",
   "tags" : [ "tutorialspoint" ]
}
```


Pattern Matching with Wild characters

Wild Characters:

- Find posts which has a tag that begins with either "t" or "T"
  SELECT * FROM posts WHERE tags LIKE "t%"
  OR tags LIKE "T%" ;
- **MongoDB:**
  db.posts.find({tags: {$regex: /**^[tT]**/}})
  db.posts.find({tags: /^t/i})

- Find posts which ends with "MongoDB" in the post_text
SELECT * FROM posts WHERE post_text LIKE "%MongoDB";
db.posts.find({post_text: {$regex: /MongoDB$/}})

MongoDB Reference

- MongoDB site: Webinars, free online courses, News, etc.
  http://www.10gen.com/
  http://www.mongodb.org/
  - Documentation, Downloads

- Various other Database commands can be found on the following website –
  https://docs.mongodb.com/manual/reference/command/

MongoDB Shell

The following Mongo Shell provides a cloud based MongoDB engine. You can create database, insert/update records, queries, etc.

- The below shell resets all data stored once closed.

https://docs.mongodb.com/manual/tutorial/query-embedded-documents/

- Copy from Notepad and paste into the shell
- Be careful about quotes
- Up arrow display the previous command