Graph & Modern Databases

COMP1835

1

1

MongoDB II

2

2

Objectives

- To introduce advanced document query
  - Projection
  - Limiting Records
  - Sorting
- To discuss indexing
- To discuss aggregation
- To introduce Replication and Sharding
- To configure User Accounts and Access Control
- To discuss MapReduce

3

3

Part 1

4

4

Projection

- In MongoDB, projection means selecting only the necessary data rather than selecting whole of the data of a document.
  - similar to selecting only a few columns from a table in RDBMS
- MongoDB's **`find()`** method accepts second optional parameter that is list of fields that you want to retrieve.
- Syntax: `db.COLLECTION_NAME.find({},{KEY:Value})`

The <Value> can be:
- 1 or true to include the field in the return docs.
- 0 or false to exclude the field.

- Example: consider this document

- Select only **`Title`** field:

```
>db.video_records. find ({}, {Title:1})
>{"_id" : ObjectId("5e01132dfd8d5670e6a8e26b"),
"Title" : "Skyfall"}
>
```

- Note: **`_id`** field is always displayed while executing find() method, if you don't want this field, then you need to set it as 0:

```
>db.video_records. find ({}, {Title:1, _id:0})
>{"Title" : "Skyfall"}
>
```

5

5

Limiting Records

- To limit the records in MongoDB, you need to use **`limit()`** method.
  - The method accepts one number type argument, which is the number of documents that you want to be displayed.
- Syntax: `>db.COLLECTION_NAME.find().limit(NUMBER)`
- Example:

```
>db.video_records. find ({}, {Title:1, _id:0}).limit(2)
```

- In addition, there is method **`skip()`** which also accepts number type argument and is used to skip the number of documents.
- Syntax: `db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)`
- For example, if there are more than 1 document in collection, *to display only the second document from collection*, use:

```
>db.video_records. find ({}, {Title:1, _id:0}).limit(1).skip(1)
```

- The default value in **skip()** method is 0.

6

6

Sorting

- To sort documents in MongoDB, you need to use sort() method.
  - The method accepts a document containing a list of fields along with their sorting order.
  - **1** is used for ascending order while **-1** is used for descending order.
- Syntax:

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

- Example, consider the collection **video_records:**

```
{ "_id" : ObjectId(5983548781331adf45ec5), "Title":"Skyfall"}
{ "_id" : ObjectId(5983548781331adf45ec6), "Title":"Casino Royal"}
{ "_id" : ObjectId(5983548781331adf45ec7), "Title":"Golden Eye"}
```

  - Display the documents sorted by title in the descending order:

```
>db.video_records.find({},{"Title":1,_id:0}).sort({"Title":-1})
{"Title":"Skyfall"}
{"Title":"Golden Eye"}
{"Title":"Casino Royal"}
```

If you don't specify the sorting preference, then `sort()` method will display the documents in ascending order.

7

7

Aggregation

- Aggregation operations process data records and return computed results
  - (Remember aggregate functions in SQL: count, min, max, avg, …?)
  - In MongoDB aggregation operations group values from multiple documents together, and perform a variety of operations on the grouped data to return a single result
  - MongoDB provides three ways to perform aggregation:
    - The aggregation pipeline
    - Single purpose aggregation methods
    - Map-reduce function
- For the aggregation pipeline in MongoDB, use **`aggregate()`** method.
- Syntax:

```
>db.COLLECTION_NAME.aggregate(pipeline, options)
```

  - The MongoDB aggregation pipeline consists of stages.
  - Each stage transforms the documents as they pass through the pipeline.
  - Pipeline stages do not need to produce one output document for every input document; e.g., some stages may generate new documents or filter out documents.

More on Aggregate pipeline stages:
https://docs.mongodb.com/manual/meta/aggregation-quick-reference/#stages

8

Aggregation Pipeline Stages

- There are many aggregation pipeline stages, the most common ones:
  - **$group**
    - groups all documents based on some keys (similar to **GROUP BY**)
  - **$match**
    - is used in filtering operation and it can reduce the number of documents (similar to **WHERE** or **HAVING** in SQL)
  - **$project**
    - is used to select certain fields from a collection (similar to **SELECT**)
  - **$sort**
    - is used to sort all the documents (similar to **ORDER BY**)
- Example: how many videos are in the collection for each Director?
- In SQL `select Director, count(*) from video_records group by Director.`
- In MongoDB:

```
> db.video_records.aggregate([{$group:{_id:"$Director",num_videos :{$sum : 1}}}])

{"id" : "Raoul Walsh", "num_videos": 1}
{"id" : "Sam Mendes", "num_videos": 2}
>
```

9

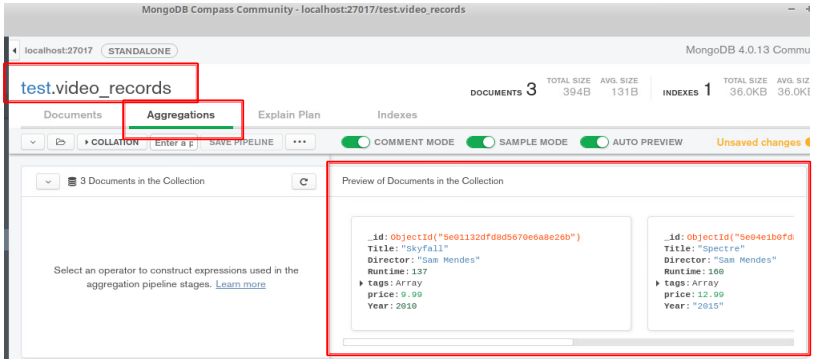
Aggregation Expressions

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.video_records.aggregate([{$group : {_id : "$Director", num_video : {$sum : "$price"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.video_records.aggregate([{$group : {_id : "$Director", avg_price : {$avg : "$price"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.video_records.aggregate([{$group : {_id : "$Director", min_price : {$min : "$price"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.video_records.aggregate([{$group : {_id : "$Director", max_price : {$max : "$price"}}}])
$push	Inserts the value to an array in the resulting document.	db.video_records.aggregate([{$group : {_id : "$Director", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.video_records.aggregate([{$group : {_id : "$Director", url : {$addToSet : "$url"}}}])
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.video_records.aggregate([{$group : {_id : "$Director", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.video_records.aggregate([{$group : {_id : "$Director", last_url : {$last : "$url"}}}])

10

10

Aggregation using MongoDB Compass



11

Quiz

1. I have 40 documents in ***my_collection***. I need to show only documents from 21st to 30th. How can I do it?

2. _____ are operations that process data records and return computed results.
   A. ReplicaAgg
   B. SumCalculation
   C. Aggregations
   D. None of the above

3. You can specify only one key in the `$group` stage.
   A. True
   B. False

12

12

Part 2

13

13

Indexing

- Indexes support the efficient resolution of queries.
  - Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement.
    - This scan is highly inefficient and require MongoDB to process a large volume of data.
  - Indexes are special data structures, that store a small portion of the data set in an easy-to-traverse form (map).
    - The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.
- To create an index you need to use **`ensureIndex()`** method
- Syntax: `>db.COLLECTION_NAME.ensureIndex(keys, options)`
- Example: `>db.video_records.ensureIndex({"Title":1})`
- You can create index on multiple fields:

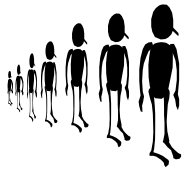
```
>db.video_records.ensureIndex({"Title":1, "Director":1})
```

  - Using optional parameter **`options`** you can give a name to the index(using parameter **`name:`** or create a unique index, by specifying **`unique:true`**

14

14

Replication



- Replication
  - is the process of synchronizing data across multiple servers.
  - provides redundancy and increases data availability with multiple copies of data on different database servers.
  - protects a database from the loss of a single server.
  - also allows you to recover from hardware failure and service interruptions.
- With additional copies of the data, you can dedicate one to disaster recovery, reporting, or backup.

15

15

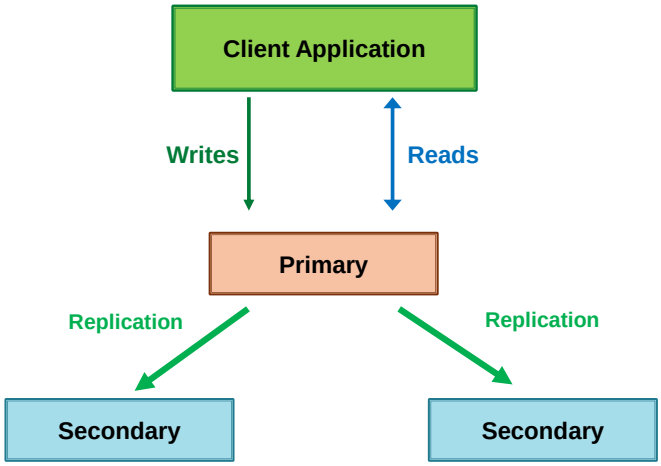
Replication in MongoDB

- MongoDB achieves replication by the use of replica set.
- A replica set is a group of mongod instances (nodes) that host the same data set (two or more nodes, generally minimum 3 nodes are required).
- In a replica, one node is primary node that receives all write operations and remaining nodes are secondary.
  - Replica set can have only one primary node.
  - All other instances, such as secondaries, apply operations from the primary so that they have the same data set.
- All data replicates from primary to secondary node.
  - At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.
  - After the recovery of failed node, it again join the replica set and works as a secondary node.

16

16

Replica Sets



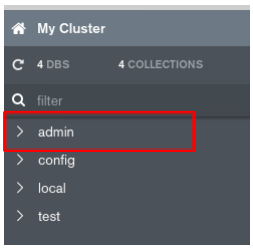
Client application always interact with the primary node and the primary node then replicates the data to the secondary nodes.

17

17

Admin Database

- When MongoDB is installed, an **admin** database is installed automatically.
  - The **admin** database is a special database that provides additional functionality above normal databases.
- For example, certain user account roles grant a user rights to multiple databases.
  - These roles can be created only in **admin** database.
- When you want to create a superuser that has rights to all databases, you must add that user to the **admin** database.
- When checking for credentials, MongoDb checks the user accounts in the specified database and in the **admin** database



18

18

User Accounts

- The important part of the database administration is creating user accounts that can administrate users and databases.
- As of MongoDB 3.0, with the localhost exception, you can only create users on the admin database.
  - Once you create the first user, you must authenticate as that user to add subsequent users.
- For routine user creation, you must possess the following permissions:
  - To create a new user in a database, you must have the **`createUser`** action on that database resource.
  - To grant roles to a user, you must have the **`grantRole`** action on the role's database.
  - The **`userAdmin`** and **`userAdminAnyDatabase`** built-in roles provide **`createUser`** and **`grantRole`** actions on their respective resources.

19

19

Creating User Accounts

- User accounts are added by using **`db.createUser()`** method or **`createUser`**
  - **`db.createUser()`** method or **`createUser`** accept a document object that enables you to specify the username, roles and password that apply to that user
- Example:

```
use admin
db.createUser(
   {
     user: "videoUser",
     pwd:  "welcome1",
     roles: [{ role: "readWrite", db: "test" } ]
   }
)
```

- When using the mongo shell, use the **`db.auth()`** method to authenticate:

```
db.auth("videoUser", "welcome1")
```

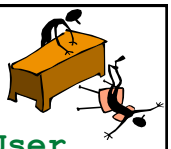
Or starting from version 4.2:

```
db.auth("videoUser", passwordPrompt())
```

20

20

Drop User



- To remove a user from the database run **dropUser** command on that database.
  - You must have the **dropUser** action on a database to drop a user from that database.
- Syntax:

```
use your_database
db.dropUser( userName)
```

- Example:

```
use admin
db.dropUser( "videoUser")
```

- Before dropping a user who has the **userAdminAnyDatabase** role, ensure you have at least another user with user administration privileges.
- To see all user use **show users** command
- To see all roles use **show roles** command

21

21

Quiz

1. How do you add a unique index on the *name* and *number* fields in a collection *my_collection*?

2. A _____ set is a group of MongoDB instances that host the same data set.
   A. copy
   B. sorted
   C. multi
   D. replica

3. Is this statement correct? In MongoDB a user can be assigned to only one role.
   A. True
   B. False

22

22

Part 3

23

23

MapReduce

- The **MapReduce** paradigm is the core of the distributed programming model in many applications to solve big data problems across different industries in the real world.
- **MapReduce** operations are designed for performing computations over large datasets.
  - There are many challenging problems such as log analytics, data analysis, recommendation engines, fraud detection, and user behaviour analysis, among others, for which MapReduce is used as a solution.
- Every **MapReduce** operation is split into two basic steps:
  - First, a **map** step performs some series of filtering and/or sorting operation, winnowing the original dataset down into subset.
  - Then, a **reduce** step performs some kind of operation on that subset.
  - For example, finding all tennis players in APT history with the first name Dave (the map step) and then finding the cumulative prize money average for all of those Daves (the reduce step).

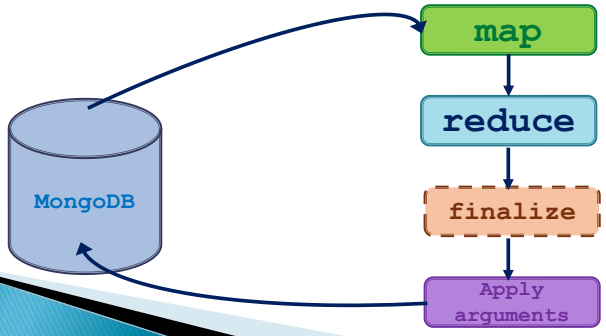
24

24

MapReduce in MongoDB

- MongoDB provides the **`mapReduce()`** method to perform map reduce operations on data before returning the results to the client.
- Syntax:

```
mapReduce (map, reduce, finalize, arguments)
```



25

25

Map Function

- The **`map`** parameter in **`MapReduce`** is a function that is applied on each object in the dataset and emits a key and value.
- The value is added to an array based on the key name to be used during the reduce phase.

```
function(){
<do stuff to calculate key and value>

emit (key, value);
}
```

26

26

Reduce

- The **reduce** parameter is a function that is applied after the map function is run on all objects with the **reduce** function.
- The **reduce** function must accept a **key** as the first parameter and array of **values** (that match that key) as the second
- The **reduce** function must then use the array of values to calculate a single value for the key and return that result
- Syntax:

```
function(key, values) {
<do stuff to on values to calculate a single result>
return result;
}
```

27

27

Finalize and Arguments

- Also, MongoDB provides an optional third step called **finalize()**, which is executed only once per mapped value after the reducers are run.
  - This allows you to perform any final calculations or clean-up you may need.

```
function(key, reducedValue) {
<do stuff to the reduced values to modify the object>
   return modifiedObject;
}
```

- The **arguments** parameter of the **mapReduce()** method is an object that defines the options to use when retrieving documents to be applied to the map function:
  - **out** specifies the location of the map-reduce query result (collection)
  - **query** specifies the optional selection criteria for selecting documents (document)
  - **sort** specifies the optional sort criteria (document)
  - **limit** specifies the optional maximum number of documents to be returned (number)

28

28

MapReduce() method

- Syntax:

```
db.collection_name.mapReduce
(
   function() {emit(this.key, this.value); },

   function(key, values)
   {
     return Array.sum(values);
   },

   function(key, reducedValue)
   {
     return modifiedObject;
   }

  {
    out: outputCollection ,
    query: {value: {$gt:6}
  }
)
```

map function

reduce function

finalize function

arguments

29

29

MapReduce() Example

- Collection *orders* contains thousands of documents like this:

```
{
     _id: 4,
     cust_id: "abc234",
     ord_date: new Date("Feb 12, 2020"),
     status: 'B',
     price: 20,
     items: [ { sku: "M25", qty: 1, price: 2.5 },
              { sku: "N678", qty: 1, price: 2.5 },
              { sku: "D221", qty: 1, price: 5 },
              { sku: "X345", qty: 1, price: 10 }]
}
```

- Perform mapReduce and return the total price per customer:

```
> db.orders.mapReduce
(
   function()
   {
      emit(this.cust_id, this.price);
    },

   function(keyCustId, valuesPrices)
   {
     return Array.sum(valuesPrices);
   },

  { out: "map_reduce_result" }
 )
> db.map_reduce_result.find()
```

map function

reduce function

arguments

30

30

Quiz

1. By default, MongoDB writes and reads data from both primary and secondary replica sets.
   A. True
   B. False

2. Map-reduce uses custom _____ functions to perform the map and reduce operations.
   A. Java
   B. Javascript
   C. BSON
   D. None of the above

3. The output of the reduce function may pass through a _____ function to further condense or process the results of the aggregation.
   A. finalize
   B. filter
   C. procedure
   D. none of the above

31

31

Essentials

- ✓ Introduced advanced document query
  - ✓ Projection
  - ✓ Limiting Records
  - ✓ Sorting
- ✓ Discussed indexing
- ✓ Discussed aggregation
- ✓ Introduced Replication and Sharding
- ✓ Configured User Accounts and Access Control
- ✓ Discussed MapReduce

32

32