**Part 3: Queries design, indexing and partitioning/sharding**

Carry out the following tasks. For submission, all queries must be executed and with screen shots of their outputs. Save the mongodb queries/statements into a script file called **part3.js**. Submit a word document including answers to Task 1 to Task 6.

**Task 1.**

1. Joe Black would like to see the retrieval of documents from multiple collections. He would like to see all customer information including the phones, addresses, interests, and the shopping carts with the cart items details for the customer id = 2.
2. Write a query to retrieve the documents from the customers and carts collections.
3. Create indexes to give a better performance for the query.
4. ShoppingWorld has realised the following queries are frequently used many times every day. The query responded very slow. Optimise the following queries to give better performance by creating indexes (i.e. sort key) to the appropriate data fields:

1. Display customer with information such as customer ID, first name, last name or company name and email and the interests where the customer first name equals to “John” and customer last name equals to “Smith”.
2. Write the query.
3. Determine which data field(s) should be indexed so that the query may have better performance.
4. Write the create index statement to optimise the data retrieval.
5. Display all information of the shopping carts together with the shopping cart items where the delivery date is smaller than the system date and the shopping cart has not been delivered.
6. Write the query.
7. Determine which data field(s) should be indexed.
8. Write the create index statement to optimise the data retrieval.

4) Write a command to show the index created in the collection.

1. For the query in Part 2 Task 4 c:

Joe Balck recently found that the following query has a very high demand in retrieving shopping carts by date/time. You are required to determine and to create an additional index to improve the performance of the retrieval.

*“Display the shopping cart where the shopping cart was created on “2022-02-20 14:30:00” with information such as date time, delivery instruction, delivered or not, delivery date, rating from customer etc. together with the shopping cart item information.”*

Note: the output should not include the \_id of the shopping card nor the rating from customer information since that information is not often needed for day-to-day business operation.

1. Write the query first and determine which data fields should be indexed.
2. Write the create index statements to optimise the data retrieval.

Your submission for Task 1 must include the following:

* Add the create index statements into the **part3.js.**
* The screen shots with the output of execution of these statements.

**Task 2**. Time-To-Live for documents

Joe Black has a concern that the logging of the transactions may cause a lot of database storage. He would like to see the logging information will be purged permanently after a certain time. He would like you to illustrate the Time-To-Live feature in MongoDB. Implement a TTL feature with the following steps:

1. Given a sample document of the log file, write a JSON schema with validator for a new collection called “**ttlLogs**” to store the logging of transaction document with information such as object \_id, logDateTime, transactionType & logMessage. Define all data fields are required and with appropriate data types. (i.e. Note: you cannot create the index without the data field defined. Therefore, you must define the JSON schema which first.)

The following is a sample document of a transaction log (given):

{

“\_id” : objectId(“6277d4474d57c31b1ff95f10”),

“loggingDateTime” : “2022-05-01 00:00:00”,

“transactionType” : “insert”,

“logMessage” : “inserted document into customers collection”

}

1. In the collection, create an index on the loggingDateTime with expire after 180 seconds.
2. Write a insertOne() query to insert a transaction log into the ttlLogs collection. Run the insertOne() statement. Hints: the current system datetime is **new Date().** Submit a screen shot with the output showing the document has been inserted. Note: The screen shot must also include the current time of your workstation.
3. Run the find() query after 3 minutes from the completion of Part c. Submit a screen shot to show any document in the ttlLogs collection. Note: The screen shot must include the current time of your workstation.
4. Explain the TTL features with the context of record being deleted. (Approx. 30 words)

Your submission for Task 2 must include the following:

* Add the mongodb statements into the part3.js
* The screen shots with the output of execution of these statements.
* Add the answers of task part e into the word document.

**Task 3**. Multiple users and authentication with permissions

Joe Black would like to know more on the authentication in the MongoDB. He would like you to illustrate the multi-user access in MongoDB with the following features:

a. Configure the user authentication by creating a user called “userAdmin” with the password “Password1” in the “admin” database with the following roles:

userAdminAnyDatabase

readWriteAnyDatabase

dbAdminAnyDatabase

b. Connect as userAdmin. (\*\*Note: you may need to restart the mongod instance o activate the authentication). Create additional user “JimBrown” with password “Password1” to have the ”read” role for the shoppingCartDB.

c. Connect as JimBrown. Test the read role by querying the supplier collection. Display the details of supplier name “Garden Grower”. Show the output message. For submission, the screen shots must show the query executed with the output showing the permission granted or not.

d. Test JimBrown whether he is authenticated to update the supplier collection by changing a supplier contact phone no. to ’08-27788888’. Show the output message.

**Task 4.** Partitioning/sharding and sort key

Joe Black has heard about MongoDB can be scaled vertically and horizontally. He would like to see the illustration on the horizontal scaling with partitioning/sharding features in MongoDB.

From the past data of the shopping cart, it showed that most customers did not return for shopping (i.e. most customers have one shopping card only). But recently, many customers are return customers and frequently shopping with ShoppingWorld (i.e. most customers have many shopping carts). MongoDB provides the sharding (horizontal scaling) feature which is to distribute data across multiple machines. These features support the deployment of large data sets with higher throughout which may help the performance of the queries.

For showing the features of partitioning to Joe Black, use the MongoDB installed on your local computer to complete this task. The reason is MongoDB Atlas webservice requires to upgrade from the free subscription before you can create replica sets for partitioning data using the partition key.

In general, you are provided with the VMWare Windows Servers with the MongoDB installed to show these features instead. Once when you have started the VMWare server, you need to make sure that the Windows domain controller is able to communicate with the member server. (Note: Instruction are given in the class exercise.)

Run the insert queries which you have saved in the previous tasks **(Part 2 Task 3 a, b, c & d)** to insert the document in the collections for the database. Do not worry about the validator.

You are required to configure the MongoDB instances in the VM Windows servers (Server1 & Server2) as shown in the diagram below so that the MongoDB router would be able to route transactions into the two shards. Without using 9 separate physical computers, use the following port numbers to simulate the mongod instances.

A diagram of a server

Description automatically generated with low confidence

1. Setup a config server, two shards and a router
2. Given the VMWare Windows servers, MongoDB replica setup script and config files in the MongoDB Assignment (ShoppingWorld) - Part 3 Student Files, configure the VMWare servers so that they are communicating using the fixed IP address as shown in the diagram above. Submit screen shots with the two server IP Addresses. Note: you must use the given IP addresses above since servers in practice must have fixed IP addresses..
3. Setup the sharded cluster in the following procedures:

* In the VMWare servers, both MongoDB has been installed locally in Server1 and Server2.
* With the given Server 2 scripts files, copy the MyCluster folder into the C: drive of the VM Sever2.***Note: You must change all folder names used in the scripts, config file names used, IP addresses used to line up with the diagram above.***
* With the given Server 1 scripts files, copy the MyCluster folder into the C: driver of the VM Server1. ***Note: You must change all folder names used in the scripts, config file names used, IP addresses used to line up with the diagram above****.*
* In Server2, run Server2\_Script1 script in the MyCluster folder.
* In Server2, run Server2\_Script1 script in the MyCluster folder.
* It sets up the folder structures for data and log files for the config server and shard01 and shard02 server instances.
* It also configures the replica set for the config server and shard01 and shard02.
* It initiates the config server replica set with one primary and two secondary members
* In Server1, run the Server1\_Script1 script.
* It configures the mongo router (mongos)
* In Sever 2, run Server2\_Script2 script in the MyCluster folder.
* It initiates the shard01 and shard 02 replica set. Each with one primary and two secondary members.
* Run commands to connect to the router port no. (> mongo --port xxxxxxxx) and show the replica status. Capture screen shots of the replica sets status.
* In Server1, run the Server1\_Script2 & Server1\_Start scripts.
* It connects the mongos client and add the shards to the mongos router.
* Run mongos commands to connect to the router port no (> **mongos** --port xxxxxxxx) and show the sharding status.
* Enable the authentication of the mongoDB in the Server1 and Server2 by creating a super user.
* use admin
* create a super user named “myAdmin” with password “password”
* allocate the role “userAdminAnyDatabase” db: “admin” and also with the role of readWriteAnyDatabase
* Write commands to enable sharding for the databases. (\*\*note; need to connect to the port no of the router first > **mongo --port 27000**).
* Show the shard status. Capture screen shot showing the sharding of the shoppingCartDB is enabled.

For submission:

* Screen shot of the Server2 with the 9 mongod instances started.
* Screen shots of the Server 2 with one of the replicas set with 1 primary and at least one of the secondary members
* Screen shot of Server2 showing the sharding of the shoppingCartDB is enabled.

1. ShoppingWorld realised that customers always enquire their “shopping cart” information by providing their customer id when making enquiries. Recommend a shard key for the carts collection that is suitable for sharding. Explain why you choose that field(s) as the shard key. (approx. 20 words)
2. In Server 1, run commands to do the following:

* Create a hashed index for the field to be act as the partition key (i.e. sort key) of the collection.
* Define a shard collection with the partition key.
* Capture a screen shot to show the shard collection distribution.
* Enable the balancer for the collections.

For submission, give the commands being used and provide one screen shot showing no distribution (i.e. 0% ) of records between the two shards yet.

1. In Server1, use the MongoDB Compass in your **windows host machine** to connect to the router.

( e.g. mongodb://router ip: router port).

Use a for loop to insert 500 new documents into the carts collection where the cartID and customerID as the following:

i.e. cartID as cart1, cart2, cart3, …….cart500

customerID as customer1, customer2, customer3 …. , customer500

The document structure can be simplified ads the following:

{

cartID: “cartID” + i,

cartdateTime: new Date(),

customerID: “customerID” + i,

cart\_Items: [ { itemID: 2,

qtyOrdered: 2}]}

1. Since there are 500 records inserted through the router, run a command to show the distribution of the carts documents are partitioned between the two shards. Submit a screen shot of the command used and the output with the % distributed between the two shards.

**Task 5.** Joe Black is also concerned about whether the MongoDB has any utility tools to measure the performance of the database. You are required to analyse the throughput of the transactions with an instance of the connection. Note: this task can be completed in MongoDB Atlas or in local MongoDB.

1. Run a system command to start the MongoTop for connecting to the mongoDB with your administration credential. For simplicity, specify 1 second as the amount of time as the instance for reading and writing by the mongod. Provide the screen shot of mongotop for the statistical reading without any activity in the database yet.
2. Use another connection to your database with the Mongo Atlas. Write a loop to insert one 100,000 documents into a collection called “**perftrans**” in the shoppingCartDB with the following details.

The inserted document should have the following information:

{ “transactionID”: “writing-“ + i,

“transactionName: “bulk write”,

“sysDateTime”: new Date(),

“tranaction amount”: Math.random() \* i}

1. Run the mongotop command, measure the output activity that happened in the “**perftrans**” collection in the database. Submit a screen shot showing the amount of time that the mongod spent in performing the write operations on the collection.
2. Use a mongo shell command to show the amount of time that needs to write the 100,000 document. Note: the unit of the reading is in micro second and you may need to stop the mango service to reset the statistics.

For submission, you must provide the mongo command being used and the screen shot to show the no. of record inserted and the duration of time used.

1. With the above information, calculate how long (in milli second) does it take to insert 4000 documents into the database. For submission, you must show your calculation.
2. Use the same mongo command to measure the read statistic. Use one findMany() command statement to find the following the following records.

transactionID: “writing-80000” or transactionID: “writing-20000”

For submission, show a screen shot with the amount of time that the query will take to locate the two records in the database.

1. Set the profile level for the mongoDB to record all queries that takes longer than two seconds to run. For submission, show the command used.

**Task 6**. Monitor progress of plan and need changes in the customer information requirements.

Joe Black recognised that the customers collection with the phone type and phone number is not practical that they are the required field. He changed his mind that he would like the customers validator to be changed for the phone type and phone number no longer be required.

He also like to have the user password be encrypted when sending request from the client to the server in the user authentication process. All changes must be tested. He would like to see this feature be prototyped by xx/xx/xxxx (e.g. 31/12/2022).

You are required to record these two changes of requirements in the **Development Plan document** under the section of Changes suggested from clients.

|  |  |  |  |
| --- | --- | --- | --- |
| **Date requested the changes** | **Changes to be made** | **Name of the client representative** | **Actions to be taken** |
|  |  |  |  |
|  |  |  |  |

For submission, you must submit the updated ITWorks Database **Development Plan document** to LEARN.

**Part 3 Submission:**

* + Submit a world document includes answers of Task 1 to Task 6. Label each task and sub tasks clearly with the answers, commands and screen shots as required. The screen shots must cover the script and the output of the execution.
  + Submit the script file **part3.js** which include all mongo queries/statements used in each task. Use // or /\* … \*/ as comments to label the task number of your answer.
  + Submit ITWorks Database **Development Plan document**.