BDS Assignment - MongoDB

Table of contents

Deliverable 1

Dataset

We are using a lung cancer patient dataset taken from Lung Cancer Dataset

It has 3000 records of patients which includes their demographic information, lifestyle factors, medical history, and clinical symptoms

Setting up MongoDB cluster

- Using https://cloud.mongodb.com/ with free tier
- Install MongoDB Shell for accessing the cluster Install mongosh - mongosh - MongoDB Docs
- Connect to the cluster from local terminal using

```
mongosh "mongodb+srv://cluster0.knq1n0l.mongodb.net/" --apiVersion 1 --username aditya
```

- Create the database for collection

```
use lung_cancer_db
```

- Load the JSON dataset into this db

```
const data = EJSON.parse(fs.readFileSync("C:\Users\aditya\Downloads\dataset.json"))
db.lung_cancer_db.insertMany(data)
```

```
Atlas atlas-13erxl-shard-0 [primary] lung_cancer_db> db.lung_cancer_db.countDocuments()
3000
Atlas atlas-13erxl-shard-0 [primary] lung_cancer_db> db.lung_cancer_db.findOne()
{
  _id: ObjectId('680bceefba963ccdc4b5f899'),
  GENDER: 'M',
  AGE: 65,
  SMOKING: 1,
  YELLOW_FINGERS: 1,
  ANXIETY: 1,
  PEER_PRESSURE: 2,
  CHRONIC_DISEASE: 2,
  FATIGUE: 1,
  ALLERGY: 2,
  WHEEZING: 2,
  ALCOHOL_CONSUMING: 2,
  COUGHING: 2,
  SHORTNESS_OF_BREATH: 2,
  SWALLOWING_DIFFICULTY: 2,
  CHEST_PAIN: 1,
  LUNG_CANCER: 'NO'
}
Atlas atlas-13erxl-shard-0 [primary] lung_cancer_db>
```

Exploring dataset

Analysis and MongoDB Queries

I have performed 5 types of analysis which involves AGE, GENDER, SMOKING

- **Total Number of Lung Cancer Diagnosed Cases**
  - Filter the document with `LUNG_CANCER=YES`
  - Group/aggregate this using `$group` and get the total of count using the `$sum` operator

```
db.lung_cancer_db.aggregate([
  { $match: { LUNG_CANCER: "YES" } },
  { $group: { _id: "Total Diagnosed Cases", count: { $sum: 1 } } }
])
```

- **Smokers with Lung Cancer**
  - Filter the document with `LUNG_CANCER=YES` and `SMOKING=1`
  - Group/aggregate this using `$group` and get the total of count using the `$sum` operator

```
db.lung_cancer_db.aggregate([
  { $match: { SMOKING: 1, LUNG_CANCER: "YES" } },
  { $group: { _id: "Smokers with Lung Cancer", count: { $sum: 1 } } }
])
```

- **Gender-wise Smoking and Lung Cancer Correlation**
  - Filter the document with `LUNG_CANCER=YES` and `SMOKING=1`
  - Aggregate this output filtered documents **based on** `GENDER` (`M` or `F`) and for each group, get the count using `$sum`
  - Use `$project` to format the output which creates field with name `gender` , keep the `count` field from previous stage by setting it to `1` and drop the `_id` field from output
  - With this analysis, we get the number of Males and Females who smoke and have a Lung cancer

```
db.lung_cancer_db.aggregate([
  { $match: { SMOKING: 1, LUNG_CANCER: "YES" } },
  { $group: { _id: "$GENDER", count: { $sum: 1 } } },
  { $project: { gender: "$_id", count: 1, _id: 0 } }
])
```

- **Average Age of Diagnosed Patients**
  - Filter the document with `LUNG_CANCER=YES`
  - Group the document and find an average age using `$avg` operator
  - Format the output using `$project`

```
db.lung_cancer_db.aggregate([
  { $match: { LUNG_CANCER: "YES" } },
  { $group: { _id: null, average_age: { $avg: "$AGE" } } },
  { $project: { _id: 0, description: "Average Age of Diagnosed Patients", aver
age_age: 1 } }
])
```

- Number of people who smoke and have lung cancer with age and gender based distribution
  - Filter the document with `LUNG_CANCER=YES` and `SMOKING=1`
  - We have 4 pipelines with Male and Female of age less than 41 and greater for each. For that we will use `$facet` aggregation
  - Filter this grouped data based on age and gender

```
db.lung_cancer_db.aggregate([
  {$match: { SMOKING: 1, LUNG_CANCER: "YES"}},
  {$facet:
    {
      "age_40_or_less_M": [{$match: {AGE: {$lte: 40}, GENDER: "M"}},
                           {$count: "count" }],
      "age_40_or_less_F": [{$match: {AGE: {$lte: 40}, GENDER: "F"}},
                           {$count: "count" }],
      "age_41_or_greater_M": [{ $match: {AGE: {$gt: 40}, GENDER: "M"}},
                           {$count: "count"}],
      "age_41_or_greater_F": [{ $match: {AGE: {$gte: 40}, GENDER: "F"}},
                           {$count: "count"}],
    }
  }
])
```

Deliverable 2: MongoDB Queries with their output snapshot

Query 1

```
Atlas atlas-13erxl-shard-0 [primary] lung_cancer_db> db.lung_cancer_db.aggregate([
...   { $match: { LUNG_CANCER: "YES" } },
...   { $group: { _id: "Total Diagnosed Cases", count: { $sum: 1 } } }
... ])
[ { _id: 'Total Diagnosed Cases', count: 1518 } ]
Atlas atlas-13erxl-shard-0 [primary] lung_cancer_db>
```

Total Number of Lung Cancer Diagnosed Cases

Query 2

```
Atlas atlas-13erxl-shard-0 [primary] lung_cancer_db> db.lung_cancer_db.aggregate([
...   { $match: { SMOKING: 1, LUNG_CANCER: "YES" } },
...   { $group: { _id: "Smokers with Lung Cancer", count: { $sum: 1 } } }
... ])
[ { _id: 'Smokers with Lung Cancer', count: 763 } ]
Atlas atlas-13erxl-shard-0 [primary] lung_cancer_db>
```

Smokers with Lung Cancer

Query 3

```
Atlas atlas-13erxl-shard-0 [primary] lung_cancer_db> db.lung_cancer_db.aggregate([
...   { $match: { SMOKING: 1, LUNG_CANCER: "YES" } },
...   { $group: { _id: "$GENDER", count: { $sum: 1 } } },
...   { $project: { gender: "$_id", count: 1, _id: 0 } }
... ])
[ { count: 376, gender: 'F' }, { count: 387, gender: 'M' } ]
Atlas atlas-13erxl-shard-0 [primary] lung_cancer_db>
```

Gender-wise Smoking and Lung Cancer Correlation

Query 4

```
Atlas atlas-13erxl-shard-0 [primary] lung_cancer_db> db.lung_cancer_db.aggregate([
...   { $match: { LUNG_CANCER: "YES" } },
...   { $group: { _id: null, average_age: { $avg: "$AGE" } } },
...   { $project: { _id: 0, description: "Average Age of Diagnosed Patients", average_age: 1 } }
... ])
[
  {
    average_age: 54.64953886693017,
    description: 'Average Age of Diagnosed Patients'
  }
]
Atlas atlas-13erxl-shard-0 [primary] lung_cancer_db>
```

Average Age of Diagnosed Patients

Query 5

```
Atlas atlas-13erxl-shard-0 [primary] lung_cancer_db> db.lung_cancer_db.aggregate([
...   {$match: { SMOKING: 1, LUNG_CANCER: "YES"}},
...   {$facet:
...     {
...       "age_40_or_less_M": [{$match: {AGE: {$lte: 40}, GENDER: "M"}},
...                            {$count: "count" }],
...       "age_40_or_less_F": [{$match: {AGE: {$lte: 40}, GENDER: "F"}},
...                            {$count: "count" }],
...       "age_41_or_greater_M": [{ $match: {AGE: {$gt: 40}, GENDER: "M"}},
...                             {$count: "count"}],
...       "age_41_or_greater_F": [{ $match: {AGE: {$gte: 40}, GENDER: "F"}},
...                             {$count: "count"}],
...     }
...   }
... ])
[
  {
    age_40_or_less_M: [ { count: 85 } ],
    age_40_or_less_F: [ { count: 91 } ],
    age_41_or_greater_M: [ { count: 302 } ],
    age_41_or_greater_F: [ { count: 293 } ]
  }
]
Atlas atlas-13erxl-shard-0 [primary] lung_cancer_db>
```

Number of people who smoke and have lung cancer with age and gender based distribution

Deliverable 3: Programming language code to connect to MongoDB Atlas

- We can connect to MongoDB using Python which needs a library PyMongo 4.12.0 documentation
- To connect with running MongoDB, we need to create a client using `MongoClient()`
- This class takes in the input of connection string which is usually in format of

  `mongodb+srv://{username}:{password}@{cluster}/ retryWrites=true&w=majority&appName={app_name}`
- Passing this connection string to `MongoClient` will initialize a connection with cluster and we can create new or use existing database and collection
- The code is available at https://github.com/aditya-bits-cc/bds-assignment-2/blob/main/python_mongdb_connection.py

```
import os
from pymongo import MongoClient
from dotenv import load_dotenv

# Load environment variables from .env file
load_dotenv()

# Read variables
username = os.getenv("MONGO_USERNAME")
password = os.getenv("MONGO_PASSWORD")
cluster = os.getenv("MONGO_CLUSTER")

# MongoDB connection string
MONGO_URI = f"mongodb+srv://{username}:{password}@{cluster}/?retry
Writes=true&w=majority&appName=Cluster0"

# Connect to MongoDB
client = MongoClient(MONGO_URI)
print(client)
```

Output

Deliverable 4: CRUD operations using Python

Code

The code is available at https://github.com/aditya-bits-cc/bds-assignment-2/blob/main/mongo_crud.py

Here, we will connect with our MongoDB cluster and perform CRUD operations

The `.env` file will contain the DB URL, Username and Password

```
import os
from pymongo import MongoClient
from dotenv import load_dotenv

# Load environment variables from .env file
load_dotenv()

# Read variables
username = os.getenv("MONGO_USERNAME")
password = os.getenv("MONGO_PASSWORD")
cluster = os.getenv("MONGO_CLUSTER")

# MongoDB connection string
MONGO_URI = f"mongodb+srv://{username}:{password}@{cluster}/?retry
Writes=true&w=majority&appName=Cluster0"

# Connect to MongoDB
client = MongoClient(MONGO_URI)
db = client['lung_cancer']
collection = db['lung_cancer']

# --------------------
# C - CREATE Operation
# --------------------
new_patient = [
   {"GENDER": "F", "AGE": 60, "SMOKING": 1, "YELLOW_FINGERS": 0, "ANX
IETY": 1, "PEER_PRESSURE": 2, "CHRONIC_DISEASE": 1, "FATIGUE": 1, "ALL
ERGY": 1,
    "WHEEZING": 1, "ALCOHOL_CONSUMING": 1, "COUGHING": 1, "SHORT
NESS_OF_BREATH": 1, "SWALLOWING_DIFFICULTY": 0, "CHEST_PAIN": 1,
"LUNG_CANCER": "YES"},
   {"GENDER": "M", "AGE": 60, "SMOKING": 2, "YELLOW_FINGERS": 1, "AN
XIETY": 1, "PEER_PRESSURE": 1, "CHRONIC_DISEASE": 2, "FATIGUE": 1, "AL
LERGY": 2,
    "WHEEZING": 1, "ALCOHOL_CONSUMING": 1, "COUGHING": 2, "SHORT
NESS_OF_BREATH": 1, "SWALLOWING_DIFFICULTY": 2, "CHEST_PAIN": 2,
```

```
"LUNG_CANCER": "YES"}
    ]

insert_result = collection.insert_many(new_patient)
print(f"Inserted document: {insert_result}")

# -----------------
# R - READ Operation
# -----------------
print("\nReading the db to find the Patients Diagnosed with Lung Cancer wi
th few parameters:")
for doc in collection.find({ "LUNG_CANCER": "YES" }, { "_id": 0, "GENDER":
1, "AGE": 1, "ANXIETY": 1, "PEER_PRESSURE": 1 }):
    print(doc)

# --------------------
# U - UPDATE Operation
# --------------------
print("\nUpdating F 60 with ANXIETY to 0 and PEER_PRESSURE TO 1:")
update_result = collection.update_one(
    { "AGE": 60, "GENDER": "F" },
    { "$set": { "ANXIETY": 0, "PEER_PRESSURE": 1 } }
)
print(f"\nUpdated {update_result.modified_count} document(s)")
for doc in collection.find({ "LUNG_CANCER": "YES" }, { "_id": 0, "GENDER":
1, "AGE": 1, "ANXIETY": 1, "PEER_PRESSURE": 1 }):
    print(doc)

# --------------------
# D - DELETE Operation
# --------------------
print("\n Delete the record of Age 60 and Gender F")
delete_result = collection.delete_one({ "AGE": 60, "GENDER": "F" })
print(f"Deleted {delete_result.deleted_count} document(s), remaining recor
ds in collection:")
for document in collection.find():  # Finds all documents in the collection
    print(document)
```

Output Screenshot

```
(venv) PS D:\Mtech\BDS\assignment\Assignment 2> python .\mongo_crud.py
Inserted document: InsertManyResult([ObjectId('680cfe5de32ec26f28540528'), ObjectId('680cfe5de32ec
26f28540529')], acknowledged=True)

Reading the db to find the Patients Diagnosed with Lung Cancer with few parameters:
{'GENDER': 'F', 'AGE': 60, 'ANXIETY': 1, 'PEER_PRESSURE': 2}
{'GENDER': 'M', 'AGE': 60, 'ANXIETY': 1, 'PEER_PRESSURE': 1}

Updating F 60 with ANXIETY to 0 and PEER_PRESSURE TO 1
Updated 1 document(s)
{'GENDER': 'F', 'AGE': 60, 'ANXIETY': 0, 'PEER_PRESSURE': 1}
{'GENDER': 'M', 'AGE': 60, 'ANXIETY': 1, 'PEER_PRESSURE': 1}

Delete the record of Age 60 and Gender F
Deleted 1 document(s), remaining records in collection:
{'_id': ObjectId('680cfe5de32ec26f28540529'), 'GENDER': 'M', 'AGE': 60, 'SMOKING': 2, 'YELLOW_FING
ERS': 1, 'ANXIETY': 1, 'PEER_PRESSURE': 1, 'CHRONIC_DISEASE': 2, 'FATIGUE': 1, 'ALLERGY': 2, 'WHEE
ZING': 1, 'ALCOHOL_CONSUMING': 1, 'COUGHING': 2, 'SHORTNESS_OF_BREATH': 1, 'SWALLOWING_DIFFICULTY'
: 2, 'CHEST_PAIN': 2, 'LUNG_CANCER': 'YES'}
(venv) PS D:\Mtech\BDS\assignment\Assignment 2>
```

MongoDB CRUD operations using Python

Deliverable 5: Consistency Configurations of Read/Write on MongoDB

- Write Concern:
  - Defines when MongoDB can consider when write operation is considered as success
  - If the value is `w=1`
    - Write operation is acknowledged only by primary replica set member
    - Fast write operation but may not be safe if primary fails
  - If value is `w=majority`
    - Write operation is acknowledged only after write operation is committed to majority of members (e.g. 1 Primary and 2 secondary means majority votes is by 2 members)
    - The acknowledgement might be slower than `w=1`
- Read Concern
  - Defines when database should read the data
  - If value is `local`
    - Read whatever the current node has even if some data is not yet committed
    - Faster operation as it doesn't wait for acknowledgement by all member nodes
  - If value is `majority`
    - Read the data which is acknowledged by majority only
    - Could be slower but safer and better consistent

Python code to implement and check read/write consistency

- In this code, we will create two different collections with different write concerns. For sake of replicating real world case, we will use the dataset with 1000 records

- We are using timer function to calculate approximate time taken for write and read operations for each type of write concern and read level

The code is available at https://github.com/aditya-bits-cc/bds-assignment-2/blob/main/mongo_consistency_check.py

```
import os
import time
from datetime import datetime
from dotenv import load_dotenv
from pymongo import MongoClient, WriteConcern
from pymongo.read_concern import ReadConcern

# Load environment variables from .env file
load_dotenv()

# Read variables
username = os.getenv("MONGO_USERNAME")
password = os.getenv("MONGO_PASSWORD")
cluster = os.getenv("MONGO_CLUSTER")

# MongoDB connection string
MONGO_URI = f"mongodb+srv://{username}:{password}@{cluster}/?retry
Writes=true&w=majority&appName=Cluster0"

# Connect to MongoDB
client = MongoClient(MONGO_URI)

# Simulating a dataset insert operation
large_dataset = [{"name": f"User {i}", "age": i % 1000} for i in range(1000)]

def timer(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        print(f"Operation `{func.__name__}` completed in {end_time - start_tim
e:.4f} seconds")
        return result
```

```
    return wrapper

@timer
def write_data(w_val=1):
    print(f"\nInserting large dataset with Write Concern {w_val}...", end=" ")
    collection.with_options(write_concern=WriteConcern(w=w_val)).insert_m
any(large_dataset)
    return None

@timer
def read_data(read_level="local"):
    print(f"Reading with Read Concern '{read_level}'...", end=" ")
    collection.with_options(read_concern=ReadConcern(level=read_level)).fi
nd_one({"name": "User 50"})
    return None


# DB and collection for w=majority
db = client.get_database("test_db1")
collection = db.get_collection("test_db1")
collection.delete_many({})

# Insert a large dataset with Write Concern w="majority" (Slower but more
consistent)
write_data(w_val="majority")
read_data(read_level="majority")
read_data(read_level="local")

# Different collection for w=1
db = client.get_database("test_db2")
collection = db.get_collection("test_db2")
collection.delete_many({}) # Clear the db for testing if already populated

# Insert a large dataset with Write Concern w=1 (Faster but less consistent)
write_data(w_val=1)
read_data(read_level="majority")
read_data(read_level="local")
```

Output Screenshot

```
(venv) PS D:\Mtech\BDS\assignment\Assignment 2> python .\mongo_consistency_check.py

Inserting large dataset with Write Concern majority... Operation `write_data` completed in 3.7905 seconds
Reading with Read Concern 'majority'... Operation `read_data` completed in 0.0101 seconds
Reading with Read Concern 'local'... Operation `read_data` completed in 0.0066 seconds

Inserting large dataset with Write Concern 1... Operation `write_data` completed in 2.8575 seconds
Reading with Read Concern 'majority'... Operation `read_data` completed in 1.0250 seconds
Reading with Read Concern 'local'... Operation `read_data` completed in 0.0077 seconds
(venv) PS D:\Mtech\BDS\assignment\Assignment 2>
```

- We can see that write operation using `majority` concern takes little bit more time than `1` as there is slight delay of committing all the data to majority of the members
- In Both cases, we can see that the read operation using `majority` takes slightly more time than `local`

Metrics Screenshot