

## Task 2: NoSQL Model Implementation and Queries

### 1. NoSQL Model Design

#### 1.1 NoSQL Model Representation

The database is designed using MongoDB, a document-oriented NoSQL database. The schema consists of a collection named persons, where each document represents an individual with attributes such as:

- Name
- DOB (date of birth)
- Favorites :
  - Favorite Book
  - Favorite Drink
  - Favorite Activity
- Address Info:
  - Street
  - City
  - Country
  - Zip Code
- Neighbours' Info
  - Neighbour1 Name
  - Neighbour1 Email
  - Neighbour2 Name
  - Neighbour2 Email

Here is a diagrammatic representation of sample data model structure generated using MongoDB Compass Schema Tools.

```

1  [{
2    "_id": {
3      "$oid": "67eb306d0f0bfe5dbd1b393d"
4    },
5    "name": "Person 1",
6    "email": "person1@email.com",
7    "dob": {
8      "$date": "1995-03-15T00:00:00.000Z"
9    },
10   "address": {
11     "street": "12 Maple St",
12     "city": "London",
13     "country": "England",
14     "zip_code": "E1 6AN"
15   },
16   "favourites": {
17     "book": "A New Beginning",
18     "drink": "Lemonade",
19     "activity": "Outdoor Running"
20   },
21   "neighbours": [
22     {
23       "name": "Neighbor A",
24       "email": "neighborA@email.com"
25     },
26     {
27       "name": "Neighbor B",
28       "email": "neighborB@email.com"
29     }
30   ]
31 },

```

#### 1.2 Implementation of the Database

The MongoDB database was implemented using Python and the **pymongo** library. Three Python scripts were created:

- **setup.py**: Establishes the MongoDB connection and creates the database.
- **insert\_data.py**: Inserts sample data into the database.
- **queries.py**: Executes NoSQL queries and retrieves required task results.

### 1.3 Code for Database Creation (setup.py)

In this script, we have used pymongo library to start connection on local MongoDB server.

```
❏ setup.py > ...
1  from pymongo import MongoClient
2
3  # Connect to MongoDB
4  client = MongoClient("mongodb://localhost:27017/")
5
6  # Create the database and collection
7  db = client["task2_nosql"]
8  collection = db["persons"]
9
10 print("MongoDB connection successful!")
```

---

### 1.4 Code for Populating the Database (insert.py)

The dataset was inserted into MongoDB using the following Python script:

```
❏ insert_data.py > ...
1  import pandas as pd
2  from setup import collection
3
4  file_path = "Assessment2425DataCaseScenario.xlsx" # Change to the correct file path
5  df = pd.read_excel(file_path)
6
7  # Convert Excel data to a format suitable for MongoDB
8  persons_data = []
9  for _, row in df.iterrows():
10     person = {
11         "name": row["Name"],
12         "email": row["Email"],
13         "dob": row["DOB"],
14         "address": {
15             "street": row["Street"],
16             "city": row["City"],
17             "country": row["Country"],
18             "zip_code": str(row["Zip Code"]) # Convert zip code to string to prevent issues
19         },
20         "favourites": {
21             "book": row["Favorite Book"],
22             "drink": row["Favorite Drink"],
23             "activity": row["Favorite Activity"]
24         },
25         "neighbours": [
26             {"name": row["Neighbour 1"], "email": row["Neighbour1Email"]} if pd.notna(row["Neighbour 1"]) else None,
27             {"name": row["Neighbour 2"], "email": row["Neighbour2Email"]} if pd.notna(row["Neighbour 2"]) else None
28         ]
29     }
30
31     person["neighbours"] = [n for n in person["neighbours"] if n is not None]
32
33     persons_data.append(person)
34
35 collection.insert_many(persons_data)
36 print("Excel data inserted into MongoDB successfully!")
```

## 1.5 Evidence of Data Population

To verify the successful population of the database, we retrieved all records using:

```
1 from setup import collection
2
3 for person in collection.find():
4     print(person)
5
```

MongoDB connection successful!

```
{'_id': ObjectId('67eb306d0f0bfe5dbd1b393d'), 'name': 'Person 1', 'email': 'person1@email.com', 'dob': datetime.datetime(1995, 3, 15, 0, 0), 'address': {'street': '12 Maple St', 'city': 'London', 'country': 'England', 'zip_code': 'E1 6AN'}, 'favourites': {'book': 'A New Beginning', 'drink': 'Lemonade', 'activity': 'Outdoor Running'}, 'neighbours': [{'name': 'Neighbor A', 'email': 'neighborA@email.com'}, {'name': 'Neighbor B', 'email': 'neighborB@email.com'}]}
{'_id': ObjectId('67eb306d0f0bfe5dbd1b393e'), 'name': 'Person 2', 'email': 'person2@email.com', 'dob': datetime.datetime(1993, 6, 22, 0, 0), 'address': {'street': '45 Oak Ave', 'city': 'Manchester', 'country': 'England', 'zip_code': 'M1 2WD'}, 'favourites': {'book': 'The Road to Success', 'drink': 'Coffee', 'activity': 'Hiking'}, 'neighbours': [{'name': 'Neighbor C', 'email': 'neighborC@email.com'}, {'name': 'Neighbor D', 'email': 'neighborD@email.com'}]}
{'_id': ObjectId('67eb306d0f0bfe5dbd1b393f'), 'name': 'Person 3', 'email': 'person3@email.com', 'dob': datetime.datetime(1991, 9, 10, 0, 0), 'address': {'street': '89 Pine Rd', 'city': 'Birmingham', 'country': 'England', 'zip_code': 'B1 1AB'}, 'favourites': {'book': 'Endless Possibilities', 'drink': 'Smoothie', 'activity': 'Swimming'}, 'neighbours': [{'name': 'Neighbor E', 'email': 'neighborE@email.com'}, {'name': 'Neighbor F', 'email': 'neighborF@email.com'}]}
{'_id': ObjectId('67eb306d0f0bfe5dbd1b3940'), 'name': 'Person 4', 'email': 'person4@email.com', 'dob': datetime.datetime(1998, 12, 5, 0, 0), 'address': {'street': '23 Birch St', 'city': 'Edinburgh', 'country': 'Scotland', 'zip_code': 'EH1 1YZ'}, 'favourites': {'book': 'Journey of Life', 'drink': 'Iced Tea', 'activity': 'Traveling'}, 'neighbours': [{'name': 'Neighbor G', 'email': 'neighborG@email.com'}, {'name': 'Neighbor H', 'email': 'neighborH@email.com'}]}
{'_id': ObjectId('67eb306d0f0bfe5dbd1b3941'), 'name': 'Person 5', 'email': 'person5@email.com', 'dob': datetime.datetime(1983, 11, 30, 0, 0), 'address': {'street': '67 Cedar Ln', 'city': 'Bristol', 'country': 'England', 'zip_code': 'BS1 3XE'}, 'favourites': {'book': 'The Adventure Continues', 'drink': 'Green Tea', 'activity': 'Gardening'}, 'neighbours': [{'name': 'Neighbor I', 'email': 'neighborI@email.com'}, {'name': 'Neighbor J', 'email': 'neighborJ@email.com'}]}
{'_id': ObjectId('67eb306d0f0bfe5dbd1b3942'), 'name': 'Person 6', 'email': 'person6@email.com', 'dob': datetime.datetime(1989, 7, 18, 0, 0), 'address': {'street': '56 Elm St', 'city': 'Liverpool', 'country': 'England', 'zip_code': 'L1 1AA'}, 'favourites': {'book': 'Finding Inner Peace', 'drink': 'Coconut Water', 'activity': 'Reading'}, 'neighbours': [{'name': 'Neighbor K', 'email': 'neighborK@email.com'}, {'name': 'Neighbor L', 'email': 'neighborL@email.com'}]}
{'_id': ObjectId('67eb306d0f0bfe5dbd1b3943'), 'name': 'Person 7', 'email': 'person7@email.com', 'dob': datetime.datetime(1996, 4, 25, 0, 0), 'address': {'street': '12 Maple St', 'city': 'Glasgow', 'country': 'Scotland', 'zip_code': 'G1 2TF'}, 'favourites': {'book': 'Exploring New Horizons', 'drink': 'Fruit Juice', 'activity': 'Cycling'}, 'neighbours': [{'name': 'Neighbor M', 'email': 'neighborM@email.com'}, {'name': 'Neighbor N', 'email': 'neighborN@email.com'}]}
{'_id': ObjectId('67eb306d0f0bfe5dbd1b3944'), 'name': 'Person 8', 'email': 'person8@email.com', 'dob': datetime.datetime(1990, 1, 9, 0, 0), 'address': {'street': '89 Oak Dr', 'city': 'Leeds', 'country': 'England', 'zip_code': 'LS1 3AB'}, 'favourites': {'book': 'The Great Journey', 'drink': 'Water', 'activity': 'Hiking'}, 'neighbours': [{'name': 'Neighbor O', 'email': 'neighborO@email.com'}, {'name': 'Neighbor P', 'email': 'neighborP@email.com'}]}
{'_id': ObjectId('67eb306d0f0bfe5dbd1b3945'), 'name': 'Person 9', 'email': 'person9@email.com', 'dob': datetime.datetime(1993, 8, 17, 0, 0), 'address': {'street': '123 Pine Rd', 'city': 'Newcastle', 'country': 'England', 'zip_code': 'NE1 2AB'}, 'favourites': {'book': 'The Power of Change', 'drink': 'Hot Chocolate', 'activity': 'Skiing'}, 'neighbours': [{'name': 'Neighbor Q', 'email': 'neighborQ@email.com'}]}
```

## 2. NoSQL Queries Implementation

### 2.1 Display Person's Name and Their Age in Years

To calculate and display the age of each person:

```
from setup import collection
from datetime import datetime

# Function to calculate age from DOB
def calculate_age(dob):
    if isinstance(dob, str):
        dob = datetime.strptime(dob, "%Y-%m-%d")

    birth_year = dob.year # Extract year directly
    current_year = datetime.now().year
    return current_year - birth_year

print("\n--- Persons and Their Age ---")
for person in collection.find({}, {"name": 1, "dob": 1}):
    age = calculate_age(person["dob"])
    print(f"Name: {person['name']], Age: {age}")
```

MongoDB connection successful!

```
--- Persons and Their Age ---
Name: Person 1, Age: 30
Name: Person 2, Age: 32
Name: Person 3, Age: 34
Name: Person 4, Age: 27
Name: Person 5, Age: 42
Name: Person 6, Age: 36
Name: Person 7, Age: 29
Name: Person 8, Age: 35
Name: Person 9, Age: 32
Name: Person 10, Age: 28
Name: Person 11, Age: 33
Name: Person 12, Age: 39
Name: Person 13, Age: 34
Name: Person 14, Age: 38
Name: Person 15, Age: 41
Name: Person 16, Age: 35
Name: Person 17, Age: 30
Name: Person 18, Age: 31
Name: Person 19, Age: 33
Name: Person 20, Age: 37
Name: Person 1, Age: 30
Name: Person 2, Age: 32
Name: Person 3, Age: 34
Name: Person 4, Age: 27
Name: Person 5, Age: 42
Name: Person 6, Age: 36
Name: Person 7, Age: 29
Name: Person 8, Age: 35
Name: Person 9, Age: 32
Name: Person 10, Age: 28
Name: Person 11, Age: 33
Name: Person 12, Age: 39
Name: Person 13, Age: 34
Name: Person 14, Age: 38
Name: Person 15, Age: 41
Name: Person 15, Age: 41
Name: Person 16, Age: 35
Name: Person 17, Age: 30
Name: Person 18, Age: 31
Name: Person 19, Age: 33
Name: Person 20, Age: 37
```

## 2.2 Group Persons by Favourite Drink and Return Average Age

```
from setup import collection
from datetime import datetime

print("\n--- Average Age by Favourite Drink ---")
pipeline = [
    {
        "$group": {
            "_id": "$favourites.drink",
            "average_age": {"$avg": {"$subtract": [datetime.now().year,
{"$toInt": {"$substr": ["$dob", 0, 4]}]}}}
        }
    }
]
results = collection.aggregate(pipeline)
for res in results:
    print(f"Drink: {res['_id']], Average Age: {res['average_age']}")
```

```
--- Average Age by Favourite Drink ---
Drink: Coffee, Average Age: 32.0
Drink: Water, Average Age: 36.0
Drink: Hot Chocolate, Average Age: 32.0
Drink: Sparkling Water, Average Age: 33.0
Drink: Iced Tea, Average Age: 27.0
Drink: Fruit Smoothie, Average Age: 28.0
Drink: Herbal Tea, Average Age: 36.0
Drink: Smoothie, Average Age: 34.0
Drink: Iced Coffee, Average Age: 41.0
Drink: Green Tea, Average Age: 36.0
Drink: Coconut Water, Average Age: 33.5
Drink: Fruit Juice, Average Age: 32.0
Drink: Lemonade, Average Age: 34.0
```

## 2.3 Display Average Age of People Who Like Hiking

```
from setup import collection
from datetime import datetime

print("\n--- Average Age of People Who Like Hiking ---")
pipeline = [
    {"$match": {"favourites.activity": "Hiking"}},
    {
        "$group": {
            "_id": None,
            "average_age": {"$avg": {"$subtract": [datetime.now().year,
{"$toInt": {"$substr": ["$dob", 0, 4]}]}}}
        }
    }
]
result = list(collection.aggregate(pipeline))
print(f"Average age of people who like Hiking: {result[0]['average_age']}")
```

MongoDB connection successful!

```
--- Average Age of People Who Like Hiking ---
Average age of people who like Hiking: 33.8
```

## 2.4 Display the Total Number of People per City in Ascending Order

```
from setup import collection
from datetime import datetime

print("\n--- Total Number of People per City ---")
pipeline = [
    {"$group": {"_id": "$address.city", "total_people": {"$sum": 1}}},
    {"$sort": {"total_people": 1}}
]
results = collection.aggregate(pipeline)
for res in results:
    print(f"City: {res['_id']], Total People: {res['total_people']}")
```

```
--- Total Number of People per City ---
City: Oxford, Total People: 2
City: Birmingham, Total People: 2
City: Leeds, Total People: 2
City: Nottingham, Total People: 2
City: Leicester, Total People: 2
City: London, Total People: 2
City: Cambridge, Total People: 2
City: Southampton, Total People: 2
City: Norwich, Total People: 2
City: Manchester, Total People: 2
City: Newcastle, Total People: 2
City: Glasgow, Total People: 2
City: Liverpool, Total People: 2
City: Sheffield, Total People: 2
City: Bristol, Total People: 2
City: Edinburgh, Total People: 4
City: Cardiff, Total People: 6
```

## 2.5 Display Name of Person(s) Whose Neighbour is 'Neighbour C'

```
from setup import collection
from datetime import datetime
```

```
print("\n--- People Whose Neighbour is 'Neighbour C' ---")
results = collection.find({"neighbours.name": "Neighbor C"},
                           {"name": 1, "_id": 0})
for person in results:
    print(person["name"])
```

MongoDB connection successful!

--- People Whose Neighbour is 'Neighbour C'  
Person 2

---

## 3. Conclusion

This documentation represents the implementation of a NoSQL database using MongoDB, including model design, population, and execution of required queries. The results validate the successful retrieval and processing of data per the given task requirements. The database structure effectively handles unstructured data while maintaining query efficiency.

### Key Outcomes:

- Successfully implemented a document-based NoSQL model.
- Inserted and retrieved sample data efficiently.
- Executed queries to analyze and manipulate data as required.
- Demonstrated NoSQL's flexibility in handling unstructured datasets.

MongoDB's schema-less nature provides scalability advantages, making it a suitable choice for evolving datasets. Further optimization could involve indexing frequently queried fields and implementing security best practices for data protection.

---

You could take a look at the full implementation code from this link:

[https://gitlab.uwe.ac.uk/y2-youssef/advanced-db/-/tree/main/Task%202?ref\\_type=heads](https://gitlab.uwe.ac.uk/y2-youssef/advanced-db/-/tree/main/Task%202?ref_type=heads)