

Task 2 - Scenario 1 Non-Relational Database Design

A. Scenario 1- Business Problem and Solution Design

A1. Business Problem

Managing patient medical records efficiently is a critical challenge for healthcare providers. Traditional relational databases, while effective in many scenarios, often struggle to handle the scalability needs and dynamic nature of semi-structured data, such as patient information, treatment histories, and data from fitness trackers. As healthcare organizations continue to grow, there is a pressing need for a more adaptable database solution capable of handling large volumes of unstructured and semi-structured data without compromising performance or flexibility.

A2. Justification for NoSQL Database Solution

Because the given data is a JSON file, This bodes well with a NoSQL environment. MongoDB , allows data to be stored in JSON-like documents, which is ideal for this exact business proposition. For example, patient records can vary in structure—different patients may have different medical histories or use various fitness trackers with unique metrics. MongoDB's ability to handle these diverse data types without a predefined schema makes it an ideal solution.

A3. NoSQL Database Type

MongoDB is a document-oriented NoSQL database, and this makes it an ideal solution for the healthcare data problem. It stores data in collections of documents, which can each contain a wide range of fields. This flexibility is perfect for storing patient information and data from various fitness trackers. As healthcare data can vary widely, this document model makes it easier to manage and store both structured and unstructured information in a way that aligns with the needs of the industry.

A4. Use of Business Data in the Solution

The business data will be organized in two main collections within MongoDB:

Patients Collection: This collection will store patient-specific information, such as personal details (name, date of birth), medical conditions, allergies, and medication histories.

Fitness Trackers Collection: This collection will hold information about fitness trackers, including device specifications (model, brand) and performance metrics (e.g., usage data, ratings).

This structure allows healthcare providers to quickly retrieve relevant patient and tracker information during consultations and track trends in patient health and fitness device usage. By

storing this data in a flexible and scalable database, it becomes easier to analyze how treatments are affecting patient health and the effectiveness of various fitness trackers.

B. Scalability

MongoDB offers some good features to help keep things running smoothly as the dataset grows:

- **Horizontal Scaling** : MongoDB lets you split data across multiple servers, so when the number of patient records increases, the load is spread out. This helps keep the system performing well even as the data keeps growing.
- **Indexing**: To speed up queries, we'll set up indexes on fields that are frequently searched, like medical conditions, last appointment dates, and tracker models. This should help pull up data faster and make everything run more efficiently.
- **Replication**: With MongoDB's replication, the database can duplicate data across different servers. This ensures that if a server goes down, the data is still accessible, minimizing downtime and keeping things up and running.
- **Dynamic Schema Evolution**: MongoDB doesn't require a strict schema, so it's easy to add new fields or collections without having to redesign the whole structure. If we need to add new tracker metrics or medical conditions later, it can be done smoothly without disrupting things.

C. Privacy and Security

Since healthcare data is really sensitive, here are a few steps to keep it safe:

- **Data Encryption**: All patient data will be encrypted, both when it's stored and when it's being transmitted over the network. This ensures the data stays protected from unauthorized access.
- **Access Control**: We'll set up role-based access control (RBAC) so only authorized people can access sensitive patient information. This helps limit the risk of data breaches and makes sure users only see what they need for their role.
- **Audit Logs**: MongoDB has built-in auditing, which will keep track of who's accessing or changing data. This way, we have a record of everything that's happening, which adds a layer of accountability and security.
- **Regulatory Compliance**: The database will follow healthcare standards like HIPAA to make sure everything stays compliant. This ensures patient data is handled the right way and kept confidential.
- **Backup and Disaster Recovery**: We'll set up regular backups to prevent data loss in case something goes wrong. And we'll have a disaster recovery plan in place, so if the system crashes, we can quickly get the data back.

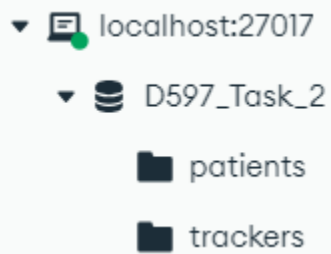
D.Implementation Phase of Proposed Database Design

D1.Creation of Database:(all code used is highlighted in Green)

use D597_Task_2

```
>_MONGOSH  
  
> use D597_Task_2;  
< switched to db D597_Task_2  
D597_Task_2>
```

Database instance:



A screenshot of the MongoDB Compass interface. It shows a tree view of the database instance. The root node is 'localhost:27017', which is expanded to show a database named 'D597_Task_2'. This database is further expanded to show two collections: 'patients' and 'trackers'.

- ▼ localhost:27017
 - ▼ D597_Task_2
 - patients
 - trackers

D2. Insert data Via JSON file:

Note:MongoImport was unavailable to use so I showed how I would generate that information below: (I used the Import files section on MongoDB Compass).

```
mongoimport --db D597_Task_2 --collection patients --file  
"C:\Users\StaphonSmith\Desktop\medical.json" --jsonArray
```

```
mongoimport --db D597_Task_2 --collection trackers --file  
"C:\Users\StaphonSmith\Desktop\Task2Scenario1 Dataset 1_fitness_trackers.json" --jsonArray
```

```
>_MONGOSH  
  
> use D597_Task_2  
< switched to db D597_Task_2  
D597_Task_2> mport --db D597_Task_2 --collection patients --file "C:\Users\StaphonSmith\Desktop\medical.json" --jsonArray  
mport --db D597_Task_2 --collection trackers --file "C:\Users\StaphonSmith\Desktop\Task2Scenario1 Dataset 1_fitness_trackers.json" --jsonArray
```

Here is a screenshot showing the data in the collections:

The screenshot shows the MongoDB Compass interface. The 'Documents' tab is selected, showing 100.0K documents. A query filter is set to '{ field: 'value' }' or 'Generate query'. The interface includes buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. The document list shows three patient records:

Document
<pre>{ "_id": ObjectId('678757afe61aacdf8a185349'), "patient_id": 1, "name": "Scott Webb", "date_of_birth": "4/28/1967", "gender": "F", "medical_conditions": "None", "medications": "No", "allergies": "None", "last_appointment_date": "7/26/2022", "Tracker": "Band 4" }</pre>
<pre>{ "_id": ObjectId('678757afe61aacdf8a18534a'), "patient_id": 2, "name": "Rachel Frederick", "date_of_birth": "4/4/1977", "gender": "M", "medical_conditions": "None", "medications": "No", "allergies": "None", "last_appointment_date": "2/14/2023", "Tracker": "Band 3" }</pre>
<pre>{ "_id": ObjectId('678757afe61aacdf8a18534b'), "patient_id": 3, "name": "Eric Kline", "date_of_birth": "5/10/1968", "gender": "M", "medical_conditions": "None", "medications": "No", "allergies": "None", "last_appointment_date": "2/14/2023", "Tracker": "Band 3" }</pre>

D3. Three Queries to retrieve specific information

Query 1: Fitness Trackers with a Rating Greater Than 4.0

Query:

"Rating (Out of 5)": { \$gt: 4 }

Generate query ↗

Explain

Reset

Find

</>

Options ▾

Project

{

"Brand Name": 1,

"Model Name": 1,

"Rating (Out of 5)": 1

}

Sort

{ "Rating (Out of 5)": -1 }

Collation

{ locale: 'simple' }

Index Hint

{ field: -1 }

Max Time MS

60000








Skip

0

Limit






0

Query Performance Summary

-  **385** documents returned
-  **565** documents examined
-  **0 ms** execution time
-  **Is not** sorted in memory
-  **0** index keys examined
-  **No index available for this query.** 

After Optimization:

Query Performance Summary


-  **385** documents returned
-  **385** documents examined
-  **1 ms** execution time
-  **Is not** sorted in memory
-  **385** index keys examined

Query used the following index:

Rating (Out of 5) ↑

Query 2: All Patients Sorted by Last Appointment Date


Query:


 Type a query: { field: 'value' } or [Generate query](#) 

Explain

Reset

Find



Options 

Project

{ "name": 1, "last_appointment_date": 1 }

Sort

{ "last_appointment_date": -1 }

Max Time MS

60000

Collation

{ locale: 'simple' }

Skip

0








Limit

0

Index Hint






{ field: -1 }


Query Performance Summary

-  100000 documents returned
-  100000 documents examined
-  320 ms execution time
-  Is sorted in memory
-  0 index keys examined
-  No index available for this query. 

After optimization:

Query Performance Summary

-  100000 documents returned
-  100000 documents examined
-  858 ms execution time
-  Is not sorted in memory
-  100000 index keys examined
- Query used the following index:

last_appointment_date 

Query 3: Retrieve Patients Using a “Band 4” Tracker

Query:

`{ Tracker: "Band 4" }`

[Generate query](#)

[Explain](#)

[Reset](#)

[Find](#)

[Options](#)

Project

`{ name: 1, last_appointment_date: 1, Tracker: 1 }`

Sort

`{ field: -1 } or [['field', -1]]`

Max Time MS

Collation

`{ locale: 'simple' }`

Skip Limit

Index Hint

`{ field: -1 }`

Query Performance Summary

4638 documents returned

100000 documents examined

56 ms execution time

Is not sorted in memory

0 index keys examined

No index available for this query.

After Optimization:

Query Performance Summary

4638 documents returned

4638 documents examined

7 ms execution time

Is not sorted in memory

4638 index keys examined

Query used the following index:

Tracker ↑

D4. Optimization Techniques

```
db.patients.createIndex({ "last_appointment_date": -1 });  
db.trackers.createIndex({ "Rating (Out of 5)": 1 });  
db.patients.createIndex({ "Tracker": 1 });
```

```
>_MONGOSH  
  
> use D597_Task_2  
< switched to db D597_Task_2  
  
> db.patients.createIndex({ "last_appointment_date": -1 });  
    db.trackers.createIndex({ "Rating (Out of 5)": 1 });  
    db.patients.createIndex({ "Tracker": 1 });  
  
< Tracker_1  
D597_Task_2>
```

The optimization techniques applied to the three queries indexing on the **last_appointment_date**, **Rating (Out of 5)**, and **Tracker** fields are designed to significantly improve the performance and run time of the database queries. By creating an index on the **last_appointment_date** field, for example, the system can quickly sort patient records by their most recent appointments, which is a common operation. Similarly, indexing the **Rating (Out of 5)** field will speed up searches for fitness trackers with ratings above a certain threshold, enabling faster filtering of products for healthcare providers. Lastly, indexing the **Tracker** field helps optimize the retrieval of patient data associated with specific fitness tracker models, improving the efficiency of queries that check tracker usage. These indexing strategies reduce the time it takes to execute the queries, especially as the dataset grows, ensuring that data retrieval remains swift and efficient.