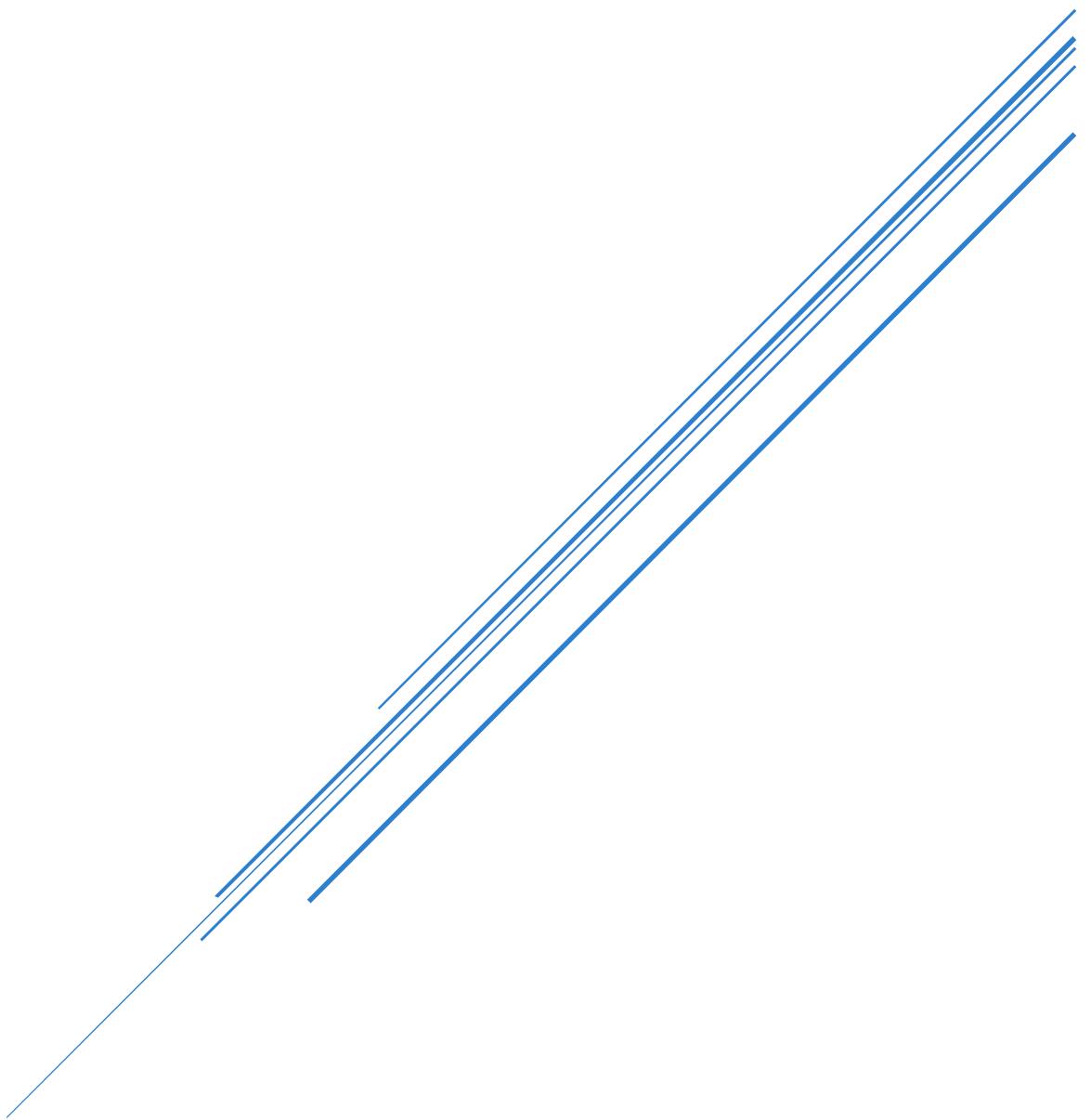


# HealthFit Database Design

Design, Implementation, and Optimization for Analytics



*Author: Joanna Ronchi  
Date: 05/24/2025*

## Table of Contents

<b>Part 1: HealthFit Design Document.....</b>	<b>3</b>
<b>A1. Description of the business problem .....</b>	<b>3</b>
Supporting the Business Mission .....	3
Top Business Needs .....	3
Top Solution Considerations .....	3
<b>A2. Justification for NoSQL database solution for the business problem.....</b>	<b>3</b>
Schema Flexibility.....	3
Horizontal Scalability .....	3
Supporting Real-Time Data Performance.....	3
Data Structure Compatibility.....	3
<b>A3. Proposed NoSQL database type to solve the identified business problem.....</b>	<b>4</b>
<b>A4. How the business data will be used within the database solution.....</b>	<b>4</b>
<b>B. How the proposed database design addresses scalability concerns.....</b>	<b>4</b>
NoSQL Database.....	4
Sharding and Horizontal Scaling.....	4
Real-Time Monitoring .....	4
Flexibility .....	4
Data Durability.....	5
Real-Time Analysis.....	5
<b>C. Privacy and security measures .....</b>	<b>5</b>
Compliance with HIPPA and GDPR.....	5
Security and Privacy of Health Data .....	5
Role-Based Access Control and Audit Logging .....	5
Backups .....	6
Specific MongoDB Features Strengthening Security.....	6
<b>Part 2: Implementation .....</b>	<b>6</b>
<b>D1. Database instance screenshots (Python and JavaScript).....</b>	<b>6</b>
Database Script Created in Visual Studio: .....	7
Screenshot of Database in MongoDB: .....	7
<b>D2. Script to insert and map the data records into the database instance. ....</b>	<b>8</b>
Successful Upload of Medical Data into D597_Task_2: .....	8
Successful Upload of Fitness Tracker Data into D597_Task_2: .....	8
MongoDB Screenshot – Successful Mapping of Medical Data into D597_Task_2.....	9
MongoDB Screenshot – Successful Mapping of Fitness Trackers into D597_Task_2.....	9
<b>D3. Query Scripts.....</b>	<b>10</b>
QUERY 1 .....	10
QUERY 2 .....	11
QUERY 3 .....	13
<b>D4. Optimization Techniques.....</b>	<b>16</b>
QUERY 1 OPTIMIZATION.....	16

<b>QUERY 2 OPTIMIZATION.....</b>	<b>18</b>
<b>QUERY 3 OPTIMIZATION.....</b>	<b>20</b>
<b>References.....</b>	<b>21</b>

## ***Part 1: HealthFit Design Document***

### ***A1. Description of the business problem***

#### *Supporting the Business Mission*

HealthFit is a health-focused business selling a data solution that enables customers to track their health data. The company provides data-driven recommendations for customers and their healthcare providers to help them improve health.

#### *Top Business Needs*

The business needs to upgrade its current database solution to address scalability, improve performance, and integrate data from heterogeneous sources. The top business problems include data silos, resulting in delaying health recommendations, and reducing the value of Health Fit's HealthTrack platform. The business also suffers from scalability bottlenecks due to rapid growth. The bottlenecks slow down the processing of data and hinder business progress. Health Track's real-time data processing issues, caused by large volumes of incoming data, create issues during the processing stage of the data lifecycle, resulting in processing delays or incorrect outputs.

#### *Top Solution Considerations*

To solve the problem, the business needs to increase the capacity of computing resources. HealthFit needs the new system to integrate complex facts – including biometric, clinical, and lifestyle data - for accurate analysis and insights. The business needs a solution to integrate and unify multiple source data, handle nested information, and unify patient information from various sources into a single document. HealthFit Innovations needs to connect the siloed and disconnected data to provide value for customers and ensure the accuracy of crucial health insights.

### ***A2. Justification for NoSQL database solution for the business problem***

#### *Schema Flexibility*

A NoSQL database will provide schema flexibility for storing varying data from multiple sources and categories. Since the company's health data varies dramatically between heart rate, blood glucose, step count, sleep and activity patterns, blood oxygen saturation, body temperature, or GPS data (for measuring distance traveled), the relational data model cannot effectively accommodate HealthFit Innovations' business needs.

#### *Horizontal Scalability*

No SQL solutions will allow the system to scale out using sharding, enabling scalability as the volume of data grows. NoSQL databases fit nicely into dynamically changing, semi-structured data from wearable devices because they do not sacrifice performance as they distribute the load between several nodes (Dey, 2020).

#### *Supporting Real-Time Data Performance*

The business collects real-time data from wearable devices, needs to support real-time data monitoring, and requires high write throughput.

#### *Data Structure Compatibility*

The NoSQL database will support document-based data more effectively than relational models by supporting nested data and arrays. The NoSQL data model directly represents hierarchical relationships and manages complex structures by including nested arrays and objects in the data

documents. The structure eliminates the need for complex tables and designs in relational databases, improving performance and easier access to data with simpler queries (Blefqih, 2024).

### **A3. Proposed NoSQL database type to solve the identified business problem**

MongoDB will support storing and analyzing semi-structured customer health data, such as JSON documents from wearable devices and applications. The database will give the business flexibility and support inputs from various users, including healthcare providers and health-data reporting systems. MongoDB will support real-time data reporting and provide performance optimization capability, support indexing, aggregation functionality, and real-time analytics (Wolniak, 2023).

### **A4. How the business data will be used within the database solution**

The data from HealthTrack will be used for real-time health-data monitoring and analysis to quickly identify and provide critical health alert data, such as heart rate measurements and alerts to customers. The database's functionality will enable the generation of health insights based on past data and identify potential health threats using predictive analytics. The solution will use data to analyze early warning signs of illness and allow users to make health-related decisions. The system will enable dashboard generation for various users, including patients and their healthcare providers, to manage and improve patient health, track improvement progress, and generate customized health tips.

## **B. How the proposed database design addresses scalability concerns**

### *NoSQL Database*

Rigid, relational schema designs are unsuitable for the HealthTrack business, primarily due to inflexibility. HealthFit Innovations solution needs a scalable database that supports millions of users uploading real-time, continuous, and diverse data related to varying health metrics. A document-oriented model offered by MongoDB will accommodate the scalability and performance and offer flexibility for the business to handle the massive amounts of data coming into the system.

### *Sharding and Horizontal Scaling*

MongoDB allows for the distribution of data across servers, which is known as sharding. The nodes distribute data based on parameters such as `user_id` or `device_id`. The database will allow Heath Fit innovations to add more users and increase without sacrificing performance by adding servers to accommodate the growth. The additional servers can handle specified portions of read/write traffic, enabling throughput and preventing failure due to traffic increases. Health monitoring data from mobile devices is high-velocity and requires strong, scalable, and flexible systems. NoSQL databases support mobile device health data and efficiently scale (He, 2021).

### *Real-Time Monitoring*

NoSQL databases allow for high-speed writes of health data from wearable devices and can effectively store time-series data. Healthcare IoT devices handle data better than traditional relational models (Dey, 2020).

### *Flexibility*

The variability of health data requires flexible data schemas to monitor diverse and evolving data structures such as heart rate metrics, blood glucose levels, medication compliance logs, or sleep

and activity data. Each type of health data requires a customized model, and the NoSQL database will enable the business to structure this data for accurate analysis (Alshammari, 2019). MongoDB offers a schema-less design and adds data without typical relational model complexities. The flexibility of the MongoDB database will enable agile development and growth of the data from various users as the business expands its customer base.

### *Data Durability*

NoSQL models provide data redundancy by storing data in multiple copies across servers and safeguarding the data against loss. In case of failure of a single node, another server assumes control over the data, minimizing downtime. The fault tolerance capability supports data availability, which is particularly important in environments that handle critical health data.

### *Real-Time Analysis*

MongoDB offers a unique aggregation pipeline capability to process, transform, and analyze data. HealthFit data can use this feature to quickly and efficiently generate insights from real-time incoming patient/customer data and enables quick generation of trigger alerts such as abnormal heart rate (Zhang, 2019). Aggregation pipelines are efficient, flexible, and important, especially for health monitoring and IoT in healthcare settings. The business can provide critical health insights, offer value to customers, and advertise the feature's availability for effective marketing campaigns.

## **C. Privacy and security measures**

### *Compliance with HIPPA and GDPR*

The U.S. Health Insurance Portability and Accountability Act (HIPPA) and the European Union's General Data Protection Regulation (GDPR) Act require businesses dealing with health information to implement guardrails to protect the safety and privacy of individual health information. Pseudonymization replaces data that identifies individuals with fields such as patientID or userID. Anonymization is another technique HealthFit should use to perform the analysis while protecting individual privacy. The data masking methods will enable safe data sharing and meet the legal obligations of a company dealing with highly sensitive health data (Wang, Zhang, 2020).

### *Security and Privacy of Health Data*

HealthFit Innovations has legal obligations to protect the privacy and security of its users. The database must provide field-level security to protect patient names, social security numbers, and other sensitive data (Fernandez, 2013). HealthFit needs to protect the data at rest and in transit. MongoDB supports the industry-standard AES-256 and Transport Level Security, offering additional client-level encryption (MongoDB, n.d., p. 1).

### *Role-Based Access Control and Audit Logging*

The business must follow the principle of least privilege to protect patient and user personal information based on necessity, especially in cloud-based health data (Gayanajake, 2017). Users and patients should be able to access all their data and assign permissions to their healthcare providers. Data analysts and scientists should only have access to anonymous and pseudonymized data sets. The business will implement audit logging to track and monitor who and when accessed the data and made changes. Configuring the database with RBAC (Role-Based Access Controls) is important. An important MongoDB feature that will enable data confidentiality, integrity, and

access controls is the system built-in `security.authorization: enabled` setting, which configures MongoDB to require users' authentication based on their roles, for example admin, read-only, or root access.

### *Backups*

MongoDB will provide automated backups to prevent data loss, which is critical in healthcare environments. The database will redundantly and securely store data in several locations using snapshots and encryption. MongoDB enables regular, scheduled backups at convenient times. The database offers replicas where one node writes, and a secondary node reads the data, preventing loss and failure. MongoDB documentation has extensive information on replication functionality, assuring the business that the solution will provide real-time configured backups and increase the security of data (MongoDB, n.d., p. 2).

### *Specific MongoDB Features Strengthening Security*

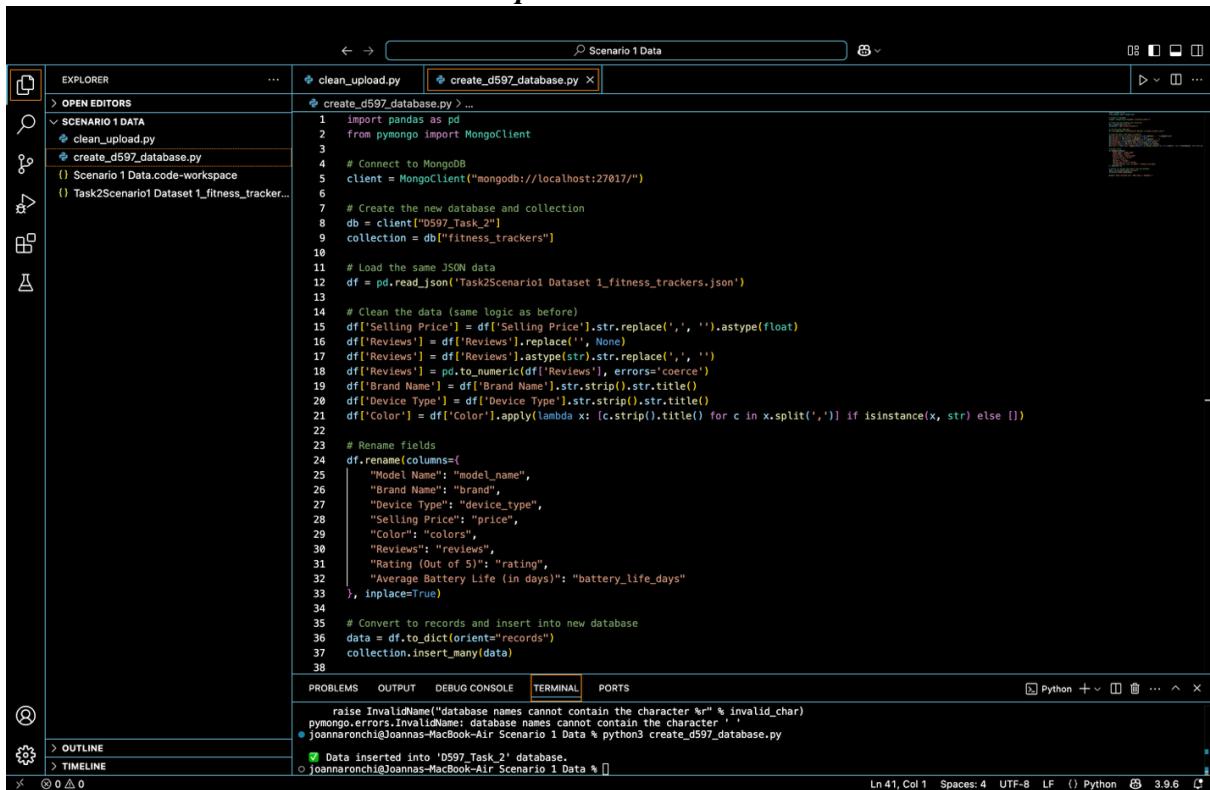
MongoDB comes with built-in security features. It is important to configure settings to prevent attacks and protect sensitive data. The recommended settings include:

- `net.port`—The feature changes the default listens to on position, reducing cyberattack vulnerabilities that scan MongoDB ports.
- `net.bindIp`—This feature limits access to the database to only specified IP addresses and prevents unauthorized access and misappropriated data use.
- `javascript-enabled` – set to disable or prevent Java script injection attacks. MongoDB will reduce the attack surface from untrusted queries or that are not sanitized inputs (MongoDB, n.d., p. 3)

## *Part 2: Implementation*

### *D1. Database instance screenshots (Python and JavaScript)*

### Database Script Created in Visual Studio:



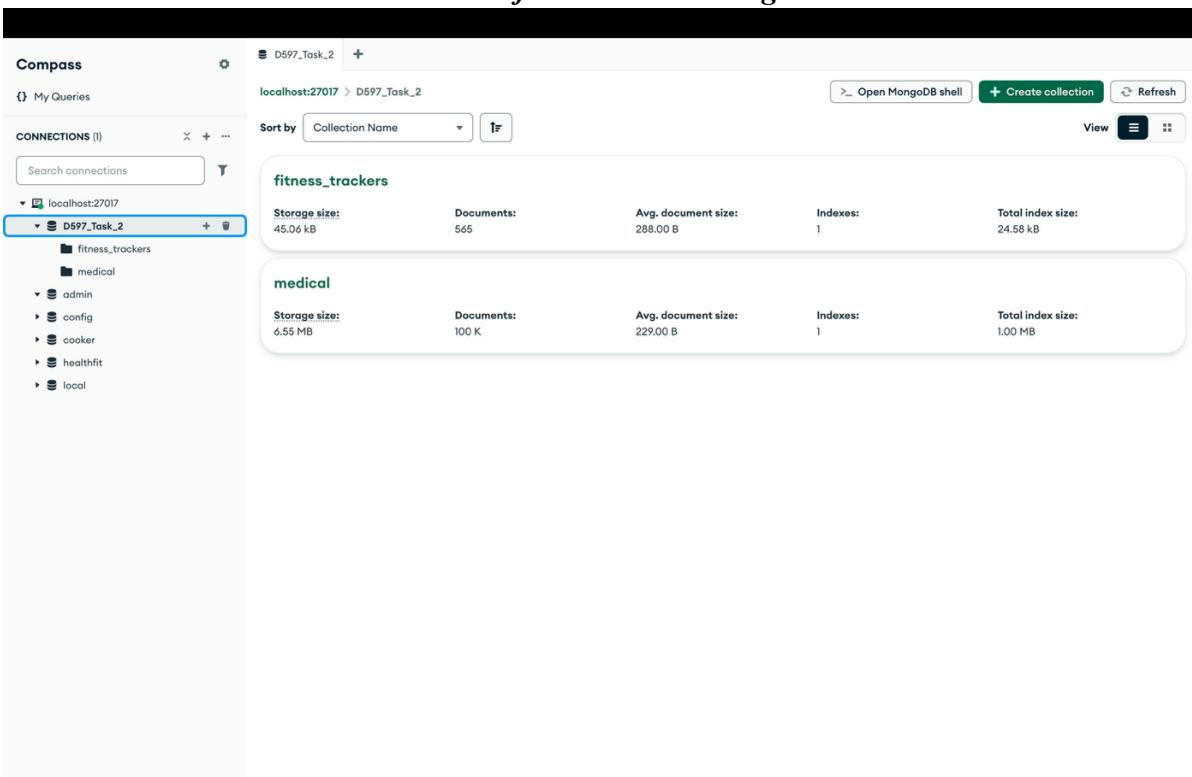
```

Scenario 1 Data> Scenario 1 Data> clean_upload.py > create_d597_database.py > ...
1 import pandas as pd
2 from pymongo import MongoClient
3
4 # Connect to MongoDB
5 client = MongoClient("mongodb://localhost:27017/")
6
7 # Create the new database and collection
8 db = client["D597_Task_2"]
9 collection = db["fitness_trackers"]
10
11 # Load the same JSON data
12 df = pd.read_json('Task2Scenario1 Dataset 1_fitness_trackers.json')
13
14 # Clean the data (same logic as before)
15 df['Selling Price'] = df['Selling Price'].str.replace(',', '').astype(float)
16 df['Reviews'] = df['Reviews'].replace('', None)
17 df['Reviews'] = df['Reviews'].astype(str).str.replace(',', '')
18 df['Reviews'] = pd.to_numeric(df['Reviews'], errors='coerce')
19 df['Brand Name'] = df['Brand Name'].str.strip().str.title()
20 df['Device Type'] = df['Device Type'].str.strip().str.title()
21 df['Color'] = df['Color'].apply(lambda x: [c.strip().title() for c in x.split(',')]) if isinstance(x, str) else []
22
23 # Rename fields
24 df.rename(columns={
25     "Model Name": "model_name",
26     "Brand Name": "brand",
27     "Device Type": "device_type",
28     "Selling Price": "price",
29     "Color": "colors",
30     "Reviews": "reviews",
31     "Rating (Out of 5)": "rating",
32     "Average Battery Life (in days)": "battery_life_days"
33 }, inplace=True)
34
35 # Convert to records and insert into new database
36 data = df.to_dict(orient="records")
37 collection.insert_many(data)
38
raise InvalidName("database names cannot contain the character %s" % invalid_char)
pymongo.errors.InvalidName: database names cannot contain the character ' '
joannarochi@Joannas-MacBook-Air: Scenario 1 Data % python3 create_d597_database.py
Data inserted into 'D597_Task_2' database.
joannarochi@Joannas-MacBook-Air: Scenario 1 Data %

```

The screenshot shows a Visual Studio Code interface. The left sidebar has sections for EXPLORER, OPEN EDITORS, SCENARIO 1 DATA, and OUTLINE. The SCENARIO 1 DATA section contains files: clean\_upload.py and create\_d597\_database.py. The right pane displays the content of create\_d597\_database.py. The code uses pandas to read a JSON file, clean the data, and then insert it into a MongoDB database named 'D597\_Task\_2'. A terminal tab at the bottom shows the command `python3 create\_d597\_database.py` was run and completed successfully.

### Screenshot of Database in MongoDB:



The screenshot shows the Compass MongoDB interface. On the left, the connections sidebar lists 'localhost:27017' and 'D597\_Task\_2'. Under 'D597\_Task\_2', there are two collections: 'fitness\_trackers' and 'medical'. The 'fitness\_trackers' collection has the following statistics:

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
45.06 kB	565	288.00 B	1	24.58 kB

The 'medical' collection has the following statistics:

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
6.55 MB	100 K	229.00 B	1	1.00 MB

## D2. Script to insert and map the data records into the database instance.

### Successful Upload of Medical Data into D597\_Task\_2:

```

upload_medical.py
1 #!/usr/bin/python3
2 from pymongo import MongoClient
3
4 # Step 1: Load the JSON file
5 df = pd.read_json('medical.json')
6
7 # DEBUG: Show column names before cleaning
8 print("Original columns:", df.columns.tolist())
9
10 # Step 2: Clean column names
11 df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
12
13 # DEBUG: Show cleaned column names
14 print("Cleaned columns:", df.columns.tolist())
15
16 # Step 3: Now try cleaning date fields using standardized names
17 if 'birth_date' in df.columns:
18     df['birth_date'] = pd.to_datetime(df['birth_date'], errors='coerce')
19
20 if 'last_appointment_date' in df.columns:
21     df['last_appointment_date'] = pd.to_datetime(df['last_appointment_date'], errors='coerce')
22
23 # Step 4: Clean gender field (if it exists)
24 if 'gender' in df.columns:
25     df['gender'] = df['gender'].astype(str).str.strip().str.title()
26
27 # Step 5: Print preview and missing data
28 print("\nCleaned data preview:\n", df.head())
29 print("\nMissing values:\n", df.isnull().sum())
30
31 # Step 6: Upload to MongoDB
32 client = MongoClient("mongodb://localhost:27017/")
33 db = client["D597_Task_2"]
34 collection = db["medical"]
35 collection.delete_many({})
36 collection.insert_many(df.to_dict(orient='records'))
37

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

medications      0
allergies        0
last_appointment_date  0
tracker          0
dtype: int64

```

Cleaned medical.json uploaded to 'medical' collection in D597\_Task\_2.

### Successful Upload of Fitness Tracker Data into D597\_Task\_2:

```

upload_fitness_trackers.py
1 import pandas as pd
2 from pymongo import MongoClient
3
4 # 1. Load the JSON file
5 df = pd.read_json('Task2Scenario1 Dataset 1_fitness_trackers.json')
6
7 # 2. Clean 'Selling Price' column
8 df['Selling Price'] = df['Selling Price'].str.replace(',', '').astype(float)
9
10 # 3. Clean 'Reviews' column
11 df['Reviews'] = df['Reviews'].replace('', None)
12 df['Reviews'] = df['Reviews'].astype(str).str.replace(' ', '')
13 df['Reviews'] = pd.to_numeric(df['Reviews'], errors='coerce')
14
15 # 4. Standardize text formatting
16 df['Brand Name'] = df['Brand Name'].str.strip().str.title()
17 df['Device Type'] = df['Device Type'].str.strip().str.title()
18
19 # 5. Convert 'Color' field to list
20 df['Color'] = df['Color'].apply(lambda x: [c.strip().title() for c in x.split(',') if isinstance(x, str) else []])
21
22 # 6. Rename fields for MongoDB (no spaces, lowercase with underscores)
23 df.rename(columns={
24     "Model Name": "model_name",
25     "Brand Name": "brand",
26     "Device Type": "device_type",
27     "Selling Price": "price",
28     "Color": "colors",
29     "Reviews": "reviews",
30     "Rating (Out of 5)": "rating",
31     "Average Battery Life (in days)": "battery_life_days"
32 }, inplace=True)
33
34 # 7. Remove any duplicates
35 df.drop_duplicates(inplace=True)
36

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

● joannarochi@joannas-MacBook-Air Scenario 1 Data % /u
● sbin/python3 "/Users/joannarochi/Desktop/NGU MDA
ata Management D597/Task 2/Scenario 1/Scenario 1 Data
/Task2Scenario1 Dataset 1_fitness_trackers.json"
● Data cleaned and uploaded successfully.
● joannarochi@joannas-MacBook-Air Scenario 1 Data % python3 upload_fitness_trackers.py
● Data cleaned and uploaded successfully.
○ joannarochi@joannas-MacBook-Air Scenario 1 Data %

```

### MongoDB Screenshot – Successful Mapping of Medical Data into D597\_Task\_2

The screenshot shows the Compass MongoDB interface with the 'medical' collection selected. The interface includes a sidebar for connections and a main area for document viewing. The documents list shows four entries, each representing a patient record with their details.

```

_id: ObjectId('68325b3154bdde7171856e91')
patient_id: 1
name: "Scott Webb"
date_of_birth: "4/28/1967"
gender: "M"
medical_conditions: "None"
medications: "None"
allergies: "None"
last_appointment_date: 2022-07-26T00:00:00.000+00:00
tracker: "Band 4"

_id: ObjectId('68325b3154bdde7171856e92')
patient_id: 2
name: "Rachel Frederick"
date_of_birth: "4/4/1977"
gender: "M"
medical_conditions: "None"
medications: "None"
allergies: "None"
last_appointment_date: 2023-02-14T00:00:00.000+00:00
tracker: "Band 3"

_id: ObjectId('68325b3154bdde7171856e93')
patient_id: 3
name: "Eric Kline"
date_of_birth: "5/18/1926"
gender: "M"
medical_conditions: "Watch"
medications: "Yes"
allergies: "None"
last_appointment_date: 2021-04-24T00:00:00.000+00:00
tracker: "Band 51"

_id: ObjectId('68325b3154bdde7171856e94')

```

### MongoDB Screenshot – Successful Mapping of Fitness Trackers into D597\_Task\_2

The screenshot shows the Compass MongoDB interface with the 'fitness\_trackers' collection selected. The interface includes a sidebar for connections and a main area for document viewing. The documents list shows three entries, each representing a fitness tracker with its specifications.

```

_id: ObjectId('683240c08aa9d37ac574de53')
brand: "Xiaomi"
device_type: "Fitnessband"
model_name: "Smart Band 5"
colors: Array (1)
price: 2499
Original Price: "2,999"
Display: "AMOLED Display"
rating: 4.1
Strap Material: "Thermoplastic polyurethane"
battery_life_days: 14
reviews: NaN

_id: ObjectId('683240c08aa9d37ac574de54')
brand: "Xiaomi"
device_type: "Fitnessband"
model_name: "Smart Band 4"
colors: Array (1)
price: 2099
Original Price: "2,499"
Display: "AMOLED Display"
rating: 4.1
Strap Material: "Thermoplastic polyurethane"
battery_life_days: 14
reviews: NaN

_id: ObjectId('683240c08aa9d37ac574de55')
brand: "Xiaomi"
device_type: "Fitnessband"
model_name: "HMSH01GE"
colors: Array (1)
price: 1722
Original Price: "2,099"
Display: "LCD Display"
rating: 3.5
Strap Material: "Leather"

```

### D3. Query Scripts

#### QUERY 1

##### Query1\_HealthFit\_Patient\_Demographics - Stage 1&2 (out of 4)

The screenshot shows the Compass MongoDB interface with the following details:

- Connections:** localhost:27017, D597\_Task\_2 (selected), medical.
- Aggregations:** Stage 1 (\$addFields) and Stage 2 (\$addFields).
- Stage 1 (\$addFields):**

```

1 // Convert date_of_birth from
2 // string to date
3 {
4   date_of_birth: {
5     $dateFromString: {
6       dateString: "$date_of_birth",
7       format: "%m/%d/%Y"
8     }
9   }
10 }

```
- Stage 2 (\$addFields):**

```

1 // calculate age
2 {
3   age: {
4     $dateDiff: {
5       startDate: "$date_of_birth",
6       endDate: "$$NOW",
7       unit: "year"
8     }
9   }
10 }

```
- Output after Stage 1 (\$addFields) stage (Sample of 10 documents):**

```

_id: ObjectId('68325b3154bdde7171856e91')
patient_id: 1
name: "Scott Webb"
date_of_birth: 1967-04-28T00:00:00.000+00:00
gender: "M"
medical_conditions: "None"
medications: "No"
allergies: "None"
last_appointment_date: 2022-07-26T00:00:00

```
- Output after Stage 2 (\$addFields) stage (Sample of 10 documents):**

```

_id: ObjectId('68325b3154bdde7171856e91')
patient_id: 2
name: "Rachel Frederick"
date_of_birth: 1977-04-04T00:00:00.000+00:00
gender: "F"
medical_conditions: "None"
medications: "No"
allergies: "None"
last_appointment_date: 2023-02-14T00:00:00

```

##### Query1\_HealthFit\_Patient\_Demographics - Stage 3&4

The screenshot shows the Compass MongoDB interface with the following details:

- Connections:** localhost:27017, D597\_Task\_2 (selected), medical.
- Aggregations:** Stage 3 (\$addFields) and Stage 4 (\$group).
- Stage 3 (\$addFields):**

```

1 // create age group
2 {
3   age_group: {
4     $cond: [
5       {
6         $lt: ["$age", 30]
7       },
8       {
9         $cond: [
10           {
11             $cond: [
12               {
13                 $lt: ["$age", 30]
14               },
15               "30-59",
16               "60 and over"
17             ]
18           }
19         ]
20       }
21     ]
22   }
23 }

```
- Stage 4 (\$group):**

```

1 // group by age and gender
2 {
3   _id: {
4     gender: "$gender",
5     age_group: "$age_group"
6   },
7   count: {
8     $sum: 1
9   }
10 }

```
- Output after Stage 3 (\$addFields) stage (Sample of 10 documents):**

```

_id: ObjectId('68325b3154bdde7171856e91')
patient_id: 1
name: "Scott Webb"
date_of_birth: 1967-04-28T00:00:00.000+00:00
gender: "M"
medical_conditions: "None"
medications: "No"
allergies: "None"
last_appointment_date: 2022-07-26T00:00:00
tracker: "Band 4"
age: 58
age_group: "30-59"

```
- Output after Stage 4 (\$group) stage (Sample of 6 documents):**

```

_id: Object
count: 21984

```

## Query 1 Results

The screenshot shows the Compass MongoDB interface with the following details:

- Connections:** localhost:27017, D597\_Task\_2, medical.
- Aggregations:** \$addFields, \$addFields, \$addFields, \$group.
- Results:** ALL RESULTS (Showing 1 ~ 6 count results)
- Data:**

  - Gender: "M", Age Group: "60 and over", Count: 21993
  - Gender: "F", Age Group: "60 and over", Count: 21984
  - Gender: "F", Age Group: "30-59", Count: 14914
  - Gender: "M", Age Group: "30-59", Count: 15034
  - Gender: "M", Age Group: "Under 30", Count: 13181
  - Gender: "F", Age Group: "Under 30", Count: 12894

## QUERY 2

### Query2\_HealthFit\_Patient\_Demographics - Stage 1&2 (out of 3)

The screenshot shows the Compass MongoDB interface with the following details:

- Connections:** localhost:27017, D597\_Task\_2, medical.
- Aggregations:** \$group, \$sort.
- Stages:**
  - Stage 1 \$group:**

```

1 // _id defines how to group the documents:
2 //by tracker, age group, and gender.
3
4 //count calculates how many documents (use
5 //are in each group.
6
7 {
8   _id: {
9     tracker: "$tracker",
10    age_group: "$age_group",
11    gender: "$gender"
12  },
13  count: {
14    $sum: 1
15  }
16 }

```
  - Stage 2 \$sort:**

```

1 //Sorts the grouped results in descending
2 //showing the most common tracker-demogra...
3 //combinations first.
4
5 {
6   count: -1
7 }

```
- Output:**
  - After Stage 1 (\$group): Sample of 10 documents. Shows a list of documents with \_id, tracker, age\_group, gender, and count.
  - After Stage 2 (\$sort): Sample of 10 documents. Shows a sorted list of documents by count in descending order.

### Query2\_HealthFit\_Patient\_Demographics - Stage 3 (out of 3)

The screenshot shows the Compass interface for a MongoDB database named 'localhost:27017 > D597\_Task\_2 > medical'. The 'Aggregations' tab is selected. The aggregation pipeline consists of three stages:

```

1 //showing the most common tracker-demographic combinations first.
2
3
4
5 {
6   count: -1
7 }
  
```

**Stage 3 (\$project):**

```

1 //Removes the _id object and flattens
2 //the fields for easy reading or export
3
4 {
5   _id: 0,
6   tracker: "$_id.tracker",
7   age_group: "$_id.age_group",
8   gender: "$_id.gender",
9   count: 1
10 }
  
```

The output after the \$project stage (sample of 10 documents) shows the flattened fields:

count	tracker	age_group	gender
1578	"Band 5"	"60 and over"	"F"
1513	"Band 5"	"60 and over"	"M"

### Query 2 Results

The screenshot shows the Compass interface for the same database and collection. The 'Aggregations' tab is selected, and the results page displays 20 count results:

count	tracker	age_group	gender
1578	"Band 5"	"60 and over"	"F"
1513	"Band 5"	"60 and over"	"M"
1300	"Amazfit Bip"	"60 and over"	"M"
1245	"Amazfit Bip"	"60 and over"	"F"
1081	"Band 3"	"60 and over"	"M"
1057	"Amazfit Bip S"	"60 and over"	"M"

**QUERY 3****Query3\_BatteryLife\_by\_Tracker\_Brand – Stage 1 & 2 (out of 7)**

The screenshot shows the Compass MongoDB interface with the aggregation pipeline for Query3\_BatteryLife\_by\_Tracker\_Brand. Stage 1 uses the \$addFields stage to convert the date\_of\_birth field from a string to a date object. Stage 2 uses the \$addFields stage to calculate the age of the patient. The output documents show the converted date and calculated age.

```

Stage 1 $addFields
1 // Convert DOB to a date
2
3 {
4   date_of_birth: {
5     $dateFromString: {
6       dateString: "$date_of_birth",
7       format: "%m/%d/%Y"
8     }
9   }
10 }

Stage 2 $addFields
1 // calculate age
2
3 {
4   age: {
5     $dateDiff: {
6       startDate: "$date_of_birth",
7       endDate: "$$NOW",
8       unit: "year"
9     }
10 }

```

**Output after \$addFields stage (Sample of 10 documents)**

```

_id: ObjectId('68325b3154bdde7171856e91')
patient_id: 1
name: "Scott Webb"
date_of_birth: 1967-04-28T00:00:00.000+00:00
gender: "M"
medical_conditions: "None"
medications: "No"
allergies: "None"
last_appointment_date: 2022-07-26T00:00:00.000+00:00

```

**Output after \$addFields stage (Sample of 10 documents)**

```

_id: ObjectId('68325b3154bdde7171856e91')
patient_id: 1
name: "Scott Webb"
date_of_birth: 1967-04-28T00:00:00.000+00:00
gender: "M"
medical_conditions: "None"
medications: "No"
allergies: "None"
last_appointment_date: 2022-07-26T00:00:00.000+00:00

```

**Query3\_BatteryLife\_by\_Tracker\_Brand – Stage 3 & 4 (out of 7)**

The screenshot shows the Compass MongoDB interface with the aggregation pipeline for Query3\_BatteryLife\_by\_Tracker\_Brand. Stage 3 uses the \$match stage to filter patients aged 60+. Stage 4 uses the \$lookup stage to join the fitness\_trackers collection based on the patient\_id. The output documents show the filtered patients and their corresponding tracker information.

```

Stage 3 $match
1 // Filter only 60+ patients
2
3 {
4   age: {
5     $gte: 60
6   }
7 }

Stage 4 $lookup
1 // Join fitness_trackers info
2
3 {
4   from: "fitness_trackers",
5   localField: "patient_id",
6   // in 'medical'
7   foreignField: "model_name",
8   // in 'fitness_trackers'
9   as: "tracker_info"
10 }

Stage 5 Sunwind

```

**Output after \$match stage (Sample of 10 documents)**

```

_id: ObjectId('68325b3154bdde7171856e93')
patient_id: 3
name: "Eric Kline"
date_of_birth: 1926-05-18T00:00:00.000+00:00
gender: "F"
medical_conditions: "Watch"
medications: "Yes"
allergies: "None"
last_appointment_date: 2021-04-24T00:00:00.000+00:00

```

**Output after \$lookup stage (Sample of 10 documents)**

```

_id: ObjectId('68325b3154bdde7171856e93')
patient_id: 3
name: "Eric Kline"
date_of_birth: 1926-05-18T00:00:00.000+00:00
gender: "F"
medical_conditions: "Watch"
medications: "Yes"
allergies: "None"
last_appointment_date: 2021-04-24T00:00:00.000+00:00

```

**Output after \$lookup stage (Sample of 10 documents)**

```

_id: ObjectId('68325b3154bdde7171856e96')
patient_id: 4
name: "James Rodriguez"
date_of_birth: 1954-07-20T00:00:00.000+00:00
gender: "M"
medical_conditions: "None"
medications: "No"
allergies: "None"
last_appointment_date: 2022-05-26T00:00:00.000+00:00

```

### Query3\_BatteryLife\_by\_Tracker\_Brand – Stage 5 & 7 (out of 7)

The screenshot shows the Compass interface for MongoDB, displaying the aggregation pipeline for the 'medical' collection. The pipeline consists of several stages:

- Stage 5 (\$unwind):** Unwinds the 'tracker\_info' array to work with each item individually.
- Stage 6 (\$group):** Groups by tracker brand to calculate average battery life.

**Output after \$unwind stage (Sample of 10 documents):**

```

1 // Unwind tracker_info array
2 // to work with each item individually
3 // joining or matching on array items
4 +
5   path: "$tracker_info",
6   preserveNullAndEmptyArrays: true
7
  
```

**Output after \$group stage (Sample of 9 documents):**

```

1 // Group by tracker brand to calculate
2 // average battery life
3 +
4   _id: "$tracker_info.brand",
5   average_battery_life: {
6     $avg: "$tracker_info.battery_life_days"
7   },
8   user_count: {
9     $sum: 1
10 }
11
  
```

### Query3\_BatteryLife\_by\_Tracker\_Brand – Stage 7 (out of 7)

The screenshot shows the Compass interface for MongoDB, displaying the final stage of the aggregation pipeline:

- Stage 7 (\$sort):** Sorts the documents by average battery life.

**Output after \$sort stage (Sample of 9 documents):**

```

1 // Sort by battery life
2 +
3   average_battery_life: -1
4
  
```

### Query 3 Results

**Compass**

localhost:27017 > D597\_Task\_2 > medical

Documents 100.0K Aggregations Schema Indexes 1 Validation

ALL RESULTS

Showing 1–9 count results

`_id: "Huami"`  
`average_battery_life : 18.642364847846494`  
`user_count : 87346`

`_id: "Oppo"`  
`average_battery_life : 14`  
`user_count : 2142`

`_id: "Realme"`  
`average_battery_life : 12.347676419965577`  
`user_count : 8134`

`_id: "Honor"`  
`average_battery_life : 11.24505415655858`  
`user_count : 58534`

`_id: "Boat"`  
`average_battery_life : 7.983078424991865`  
`user_count : 6146`

`_id: "Xiaomi"`  
`average_battery_life : 7`  
`user_count : 4172`

`_id: "Huawei"`  
`average_battery_life : 7`  
`user_count : 6062`

`user_count : 2142`

`_id: "Realme"`  
`average_battery_life : 12.347676419965577`  
`user_count : 8134`

`_id: "Honor"`  
`average_battery_life : 11.24505415655858`  
`user_count : 58534`

`_id: "Boat"`  
`average_battery_life : 7.983078424991865`  
`user_count : 6146`

`_id: "Xiaomi"`  
`average_battery_life : 7`  
`user_count : 4172`

`_id: "Huawei"`  
`average_battery_life : 7`  
`user_count : 6062`

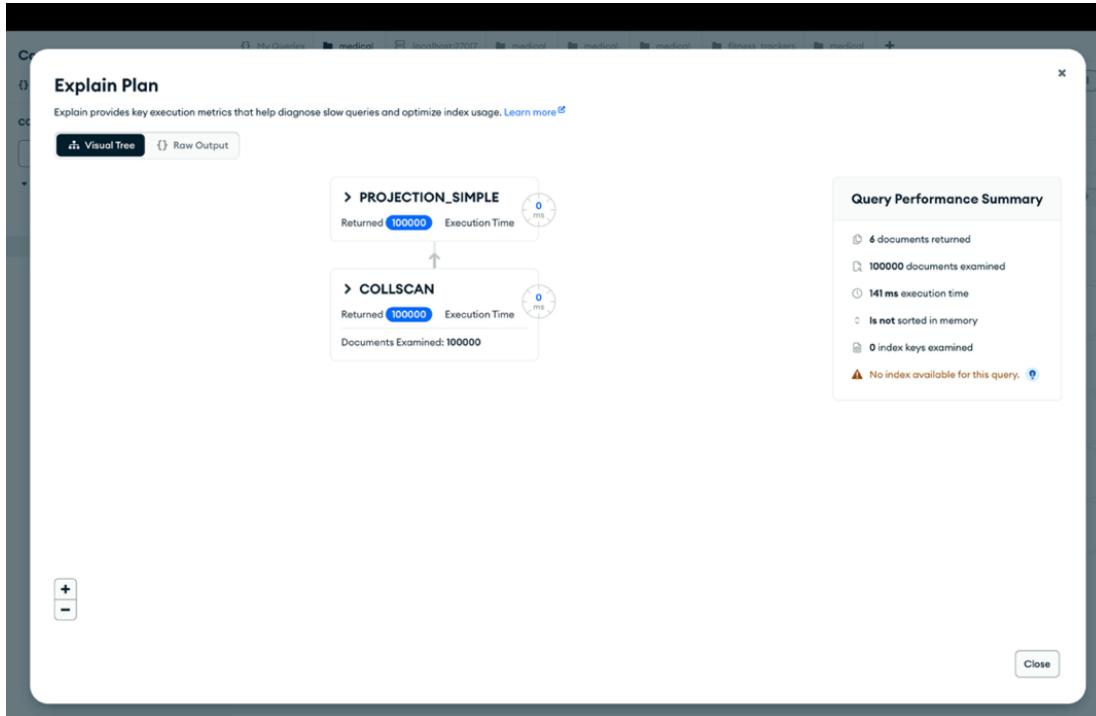
`_id: "Infinix"`  
`average_battery_life : 4`  
`user_count : 6182`

`_id: null`  
`average_battery_life : null`  
`user_count : 7969`

## D4. Optimization Techniques

### QUERY 1 OPTIMIZATION

#### Query 1 Execution Without Optimization (141ms)



#### Query 1 Optimization Techniques

The screenshot shows the Compass interface for MongoDB. The left sidebar shows connections and databases, with 'localhost:27017' selected. Under 'D597\_optimization', 'medical\_optimization' is selected. The main area shows the aggregation pipeline for 'Query\_1'. Stage 1 (\$match) filters documents where gender is 'M' or 'F' and age\_group is 'Under 30', '30-59', or '60 and over'. Stage 2 (\$group) groups by gender and age\_group, summing the count. The output after each stage is shown in the preview pane.

```

Stage 1 [$match]
1: {
  gender: { $in: ["M", "F"] },
  age_group: { $in: ["Under 30", "30-59", "60 and over"] }
}

Stage 2 [$group]
1: {
  _id: { gender: "$gender", age_group: "$age_group" },
  count: { $sum: 1 }
}

```

The screenshot shows the Compass MongoDB interface. The left sidebar displays connections and databases, including 'D597\_task\_2' and 'D597\_optimization'. The main area shows an aggregation pipeline for the 'medical\_optimization' collection. The pipeline consists of three stages:

```

1. $match: {
  _id: {
    gender: "$gender",
    age_group: "$age_group"
  }
}
2. $group: {
  count: {
    $sum: 1
  }
}
3. $sort: {
  "_id.gender": 1,
  "_id.age_group": 1
}

```

The results show two documents from the output stage:

- Document 1: `_id: Object count: 15034`
- Document 2: `_id: Object count: 13181`

Below the pipeline, a preview of the output after the `$sort` stage is shown, containing a sample of 6 documents.

*Query 1 After Optimization (92ms)*

The screenshot shows the 'Explain Plan' dialog in Compass. It displays the execution plan for the query, which includes three stages:

- GROUP**: Returned 6 documents, Execution Time 0 ms.
- FETCH**: Returned 100000 documents, Execution Time 15 ms.
- IXSCAN**: Returned 100000 documents, Execution Time 34 ms. Index Name: `gender_1`, Multi Key Index: no.

The 'Query Performance Summary' section provides the following metrics:

- 6 documents returned
- 100000 documents examined
- 92 ms execution time
- Is not sorted in memory
- 100000 index keys examined

It also notes that the query used the `gender` index.

## QUERY 2 OPTIMIZATION

### Query 2 Optimization Techniques

**Compass**

localhost:27017 > D597\_optimization > medical\_optimization

Aggregations

QueryL\_final\_o... - modified

**Stage 1 \$sort**

```

1 // Sort by tracker, age_group, and gender
2 // In ascending order
3 // Helps MongoDB group more efficiently
4 // When using an Index that matches this
5
6 { tracker: 1, age_group: 1, gender: 1 }

```

Output after \$sort stage (Sample of 10 documents)

```

_id: ObjectId('68325b3154bdde7171857593')
patient_id: 1795
name: "Brandon Short"
date_of_birth: 1980-11-30T00:00:00.000Z
gender: "M"
medical_conditions: "None"
medications: "No"
allergies: "dietary"
last_appointment_date: 2021-07-28T00:00:00Z

```

```

_id: ObjectId('68325b3154bdde7171857597')
patient_id: 1819
name: "Amanda Garcia"
date_of_birth: 1952-05-07T00:00:00.000Z
gender: "F"
medical_conditions: "Watch"
medications: "Yes"
allergies: "None"
last_appointment_date: 2021-08-11T00:00:00Z

```

**Stage 2 \$group**

```

1 // Group documents by tracker, age_group,
2 // Count how many documents belong to each
3 {
4   _id: { tracker: "$tracker",
5         age_group: "$age_group",
6         gender: "$gender"
7       },
8   count: { $sum: 1 }
9 }

```

Output after \$group stage (Sample of 10 documents)

```

_id: Object
count: 1

```

```

_id: Object
count: 1

```

**Compass**

localhost:27017 > D597\_optimization > medical\_optimization

Aggregations

QueryL\_final\_o... - modified

**Stage 3 \$project**

```

1 // Reshape the output
2 // Promote fields from _id to
3 // top-level fields for cleaner results
4 {
5   _id: 0,
6   tracker: "$_id.tracker",
7   age_group: "$_id.age_group",
8   gender: "$_id.gender",
9   count: 1
10 }

```

Output after \$project stage (Sample of 10 documents)

```

count: 1
tracker: "Band 67"
age_group: "Under 30"
gender: "M"

```

```

count: 1
tracker: "Band 258"
age_group: "60 and over"
gender: "F"

```

**Stage 4 \$sort**

```

1 // Sort the grouped results
2 // by count in descending order
3 // This shows the most common
4 // tracker-demographic combinations first
5 { count: -1 }
6

```

Output after \$sort stage (Sample of 10 documents)

```

count: 1578
tracker: "Band 5"
age_group: "60 and over"
gender: "F"

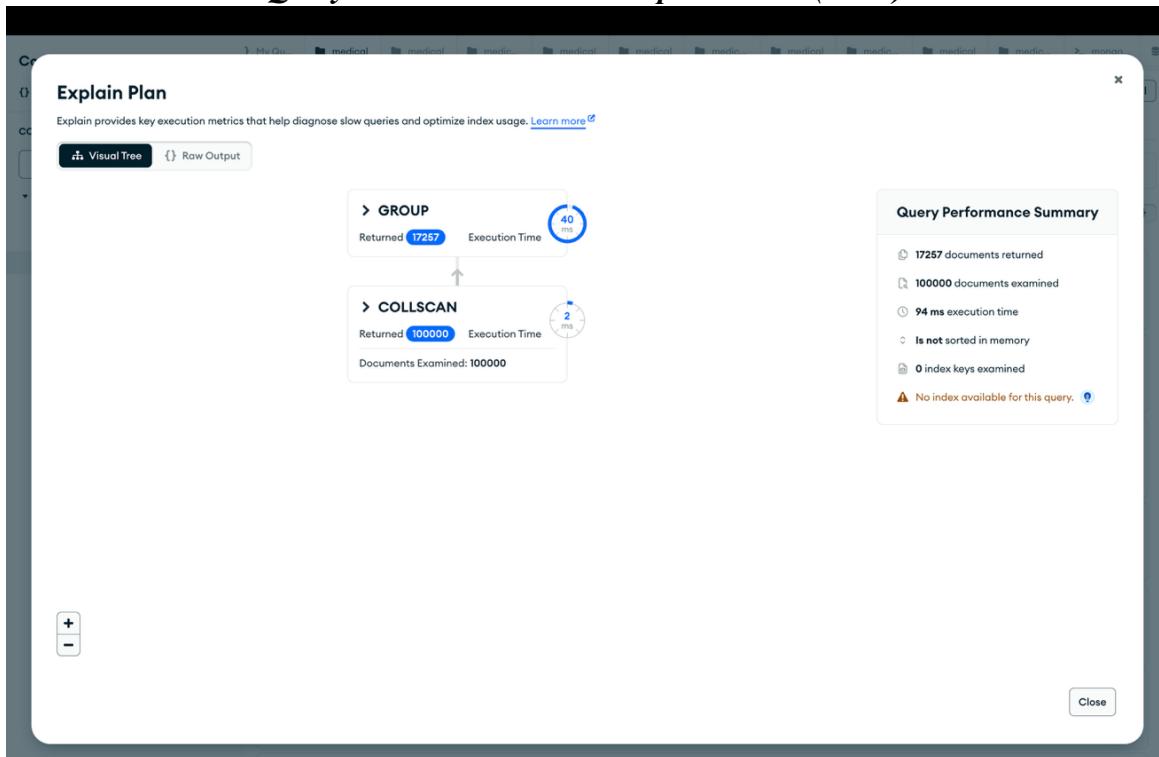
```

```

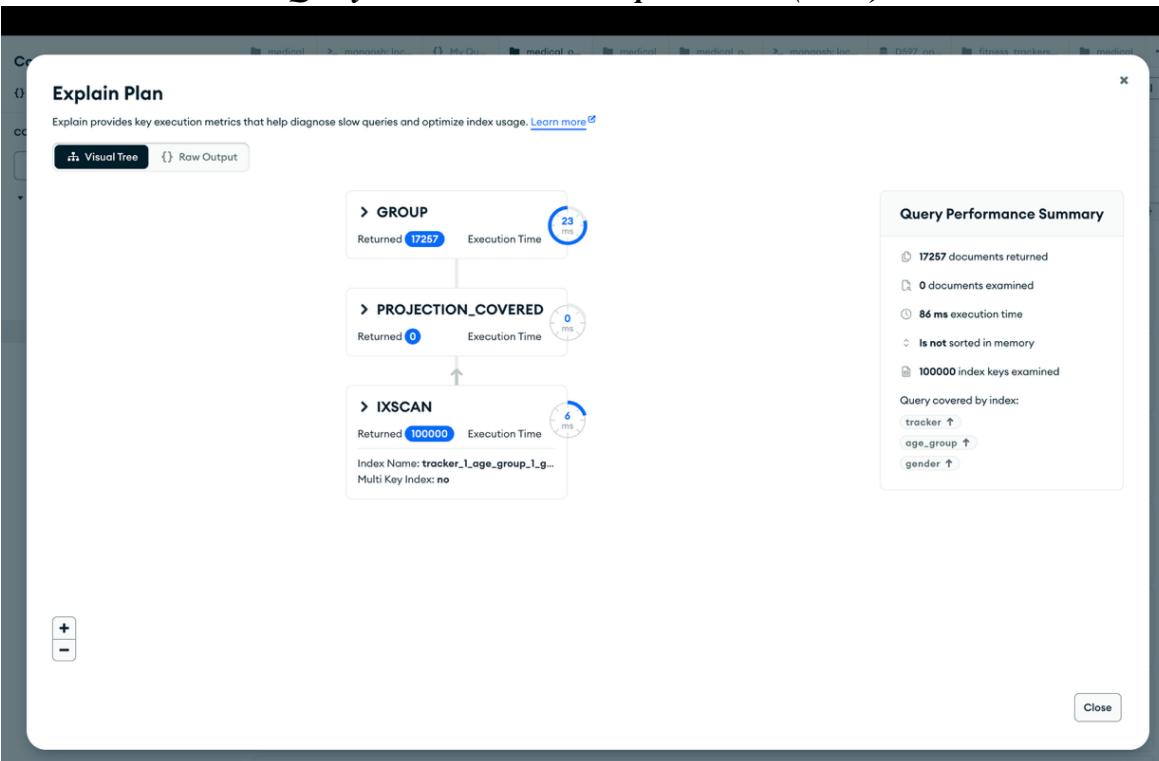
count: 1513
tracker: "Band 5"
age_group: "60 and over"
gender: "M"

```

### *Query 2 Execution Without Optimization (94ms)*

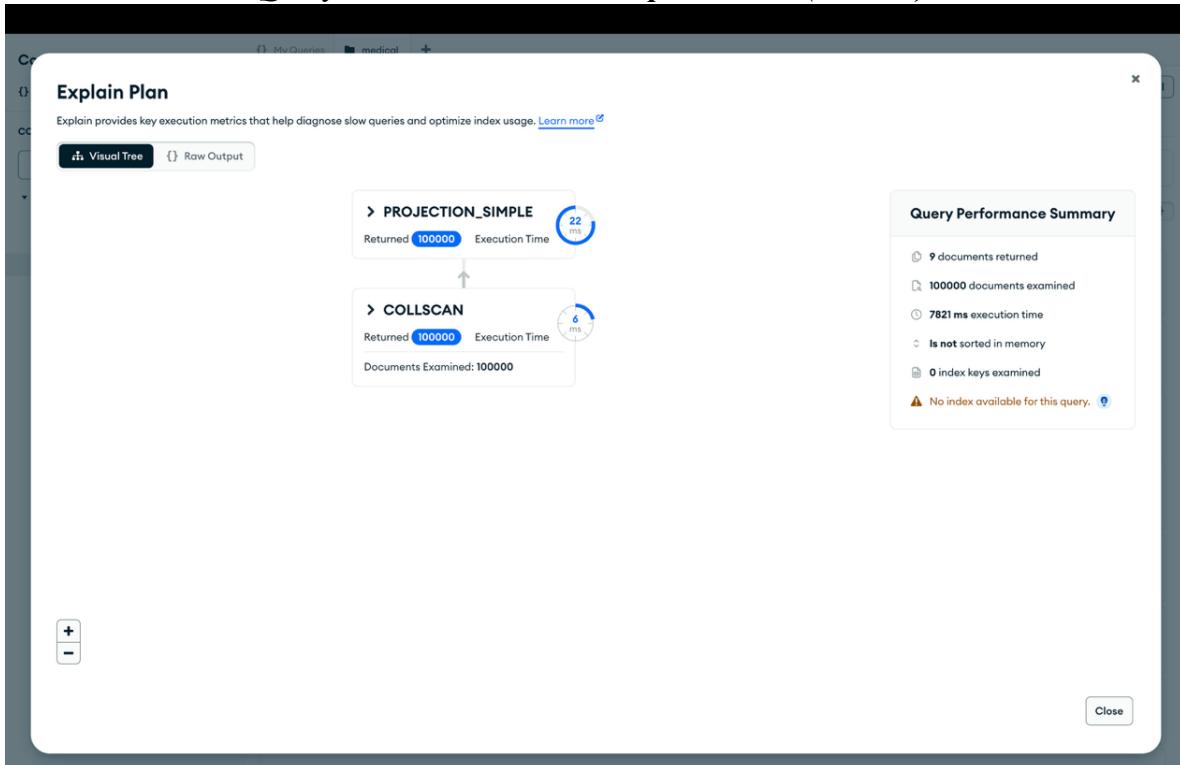


### *Query 2 Execution With Optimization (86ms)*

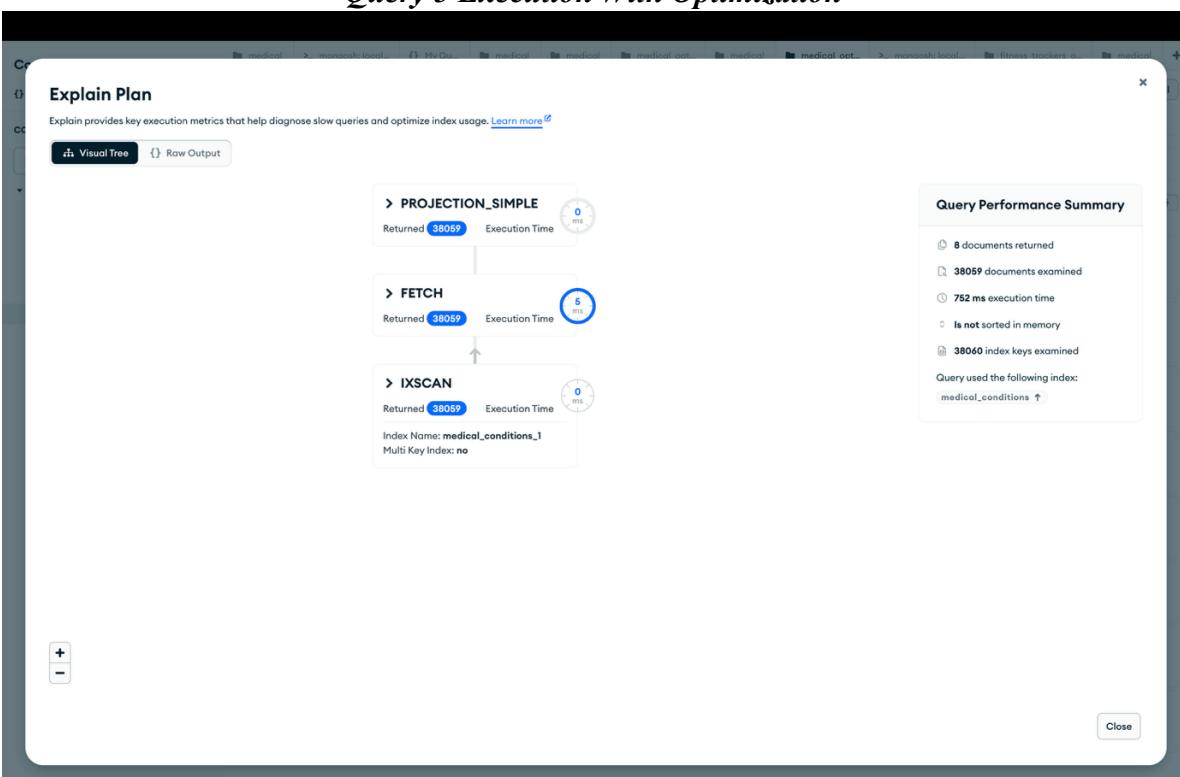


## QUERY 3 OPTIMIZATION

### Query 3 Execution Without Optimization (7821ms)



### Query 3 Execution With Optimization



## References

- Y. He, S. Lee, R. Zhang, & M. S. Kim. (2021). *Scalable storage of real-time personal health data using NoSQL databases in smart health applications*. *Sensors*, 21(2), 535. <https://doi.org/10.3390/s21020535>
- M. Dey, M. Kabir, & A. Ghosh (2020). *Evaluating NoSQL databases for scalable real-time health monitoring systems*. *Journal of Big Data*, 7(1). <https://doi.org/10.1186/s40537-020-00320-6>
- G. Alshammari, A. Alhaidari, & A. Almogren. (2019). *NoSQL for healthcare big data: A performance evaluation of MongoDB and Cassandra*. *IEEE Access*, 7, 149848–149855. <https://doi.org/10.1109/ACCESS.2019.2946975>
- Y. Zhang, J. Dang, & L. Zhang. (2019). *Real-time health monitoring and analytics with big data*. *IEEE Transactions on Industrial Informatics*, 15(1), 384–393. <https://doi.org/10.1109/TII.2018.2856580>
- Fernández-Alemán, J. L., Señor, I. C., Lozoya, P. Á. O., & Toval, A. (2013). *Security and privacy in electronic health records: A systematic literature review*. *Journal of Biomedical Informatics*, 46(3), 541–562. <https://doi.org/10.1016/j.jbi.2012.12.003>
- MongoDB, Inc. (n.d. 1). *Client-side field level encryption*. MongoDB Documentation. <https://www.mongodb.com/docs/manual/core/security-client-side-encryption/>
- Gajanayake, R., Iannella, R., & Sahama, T. (2017). Security and privacy in the use of cloud computing for health data. *Health Information Science and Systems*, 5(1). <https://doi.org/10.1007/s13755-017-0020-5>
- MongoDB, Inc. (n.d. 2). *Replication*. MongoDB Documentation. <https://www.mongodb.com/docs/manual/replication/>
- Xie, H., Wang, F., & Zhang, L. (2020). Data masking approaches for privacy-preserving electronic health record sharing. *BMC Medical Informatics and Decision Making*, 20(Suppl 3), 132. <https://doi.org/10.1186/s12911-020-01125-4>
- MongoDB, Inc. (n.d. 3). *Security checklist*. MongoDB Documentation. <https://www.mongodb.com/docs/manual/administration/security-checklist/>
- Dey, M., Kabir, M. A., & Ghosh, A. (2020). Evaluating NoSQL databases for scalable real-time health monitoring systems. *Journal of Big Data*, 7(1), 1–25. <https://doi.org/10.1186/s40537-020-00320-6>
- Wolniak, R. (2023). Functioning of real-time analytics in business. *Scientific Papers of Silesian University of Technology: Organization and Management Series*, (172), 659–672. <https://doi.org/10.29119/1641-3466.2023.172.40>

Belefqih, S., Zellou, A., & Berquedich, M. (2024). Semantic schema extraction in NoSQL databases using BERT embeddings. *Data Science Journal*, 23(1), 1–15.  
<https://doi.org/10.5334/dsj-2024-057>