

Task 2:

Non-Relational Database Design and Implementation

Keniyah Chestnut

Data Management — D597

SID: 012601305

A1: Business Problem

In Scenario 2, EcoMart's data is currently stored in JSON files, making it less accessible and challenging to analyze. Transitioning this data into a database will enhance its usability for querying and analysis. By importing the cosmetics dataset into a database, we can address several business questions:

1. How many orders does EcoMart receive that contain Buttermilk, and is it sufficient to be profitable?
2. How many moisturizers in the inventory are labeled as suitable for sensitive skin, and does this represent a high proportion?
3. How many orders include beef, and what would be the cost implications if there were a recall on beef?

Answering these questions will provide insights into product descriptions, inventory categorization, and sensitive items, aiding in inventory management and marketing strategies.

A2 & A3: Database Justification and Type

Given that our data is currently stored in JSON format, transitioning to a NoSQL database is a logical choice. Specifically, I selected a document-oriented NoSQL database, which is designed to store and manage data as flexible, self-contained documents.

To implement this, I used MongoDB, a widely used document store database that aligns seamlessly with JSON. It allows us to store and manage data without the constraints of a fixed schema.

We're dealing with two distinct datasets: groceries and cosmetics. Each has its own unique attributes and structure. Attempting to fit them into a traditional relational database would require forcing them into a uniform schema, which isn't ideal. With MongoDB's document model, we can store each dataset in its own collection, preserving their unique structures without compromise.

This flexibility not only simplifies the data import process but also makes future modifications straightforward. If we need to add new fields or adjust existing ones, we can do so without overhauling the entire schema. This adaptability is crucial for accommodating the evolving needs of the business.

In summary, choosing a document-oriented NoSQL database like MongoDB provides the flexibility, scalability, and efficiency needed to manage our diverse and evolving datasets.

A4: Data Usage

The two collections in the database serve distinct purposes, each providing valuable insights for EcoMart's operations.

The groceries collection compiles data on products purchased by various members. This centralized repository enables the analysis of customer buying patterns, such as identifying frequently purchased items or monitoring shopping frequency. Such insights can inform inventory decisions, promotional strategies, and customer engagement initiatives.

On the other hand, the cosmetics collection houses detailed product information, including brand names, rankings, descriptions, prices, and more. With this data structured in the database, the business can efficiently search for top-rated products, compare prices across brands, and detect trends in product popularity. This facilitates informed decisions in product stocking, marketing campaigns, and pricing strategies.

By leveraging these collections, EcoMart can transform raw data into actionable insights, driving better decision-making and enhancing overall business performance.

B: Scalability

Now that the data is structured and stored in collections, the database is far more scalable than when everything was just in a flat JSON file. Before, it wasn't easily searchable or practical for

everyday use. But with everything organized into MongoDB collections, scaling the database up or down is much easier.

New data can be added at any time, and if certain fields or records become unnecessary, we can trim the data down using queries. I also created indexes on frequently searched fields to make queries run faster. This will make a big difference as the dataset grows, helping ensure that searches remain quick even with a much larger volume of data.

C: Privacy and Security

Since the database contains sensitive information, the company should be proactive about privacy and security. Some records may include customer IDs or pricing data that shouldn't be publicly accessible.

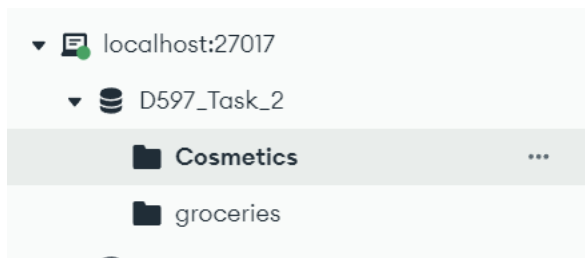
To keep the data safe, the company should at least encrypt the data and require two-factor authentication for anyone accessing the database. It's also important to run regular security checks to stay ahead of potential threats and vulnerabilities.

D1: Database Instance

The Script I used:

```
> use 597_Task_2
< switched to db 597_Task_2
597_Task_2 >
```

The database Instance:



D2: Insert Records

To populate the MongoDB collections for this project, I used the mongoimport utility included in MongoDB's bin directory. This allowed me to directly import structured JSON files into the appropriate collections within the D597_Task_2 database. The two datasets, groceries and cosmetics, were each imported using a JSON array format.

```
mongoimport --db D597_Task_2 --collection groceries --file "C:\Users\marri\Downloads\Task 2 Scenario 2\Task 2 Scenario 2\Task2Scenario2 Dataset 2_Groceries_dataset.json" --jsonArray
```

```
mongoimport --db D597_Task_2 --collection cosmetics --file "C:\Users\marri\Downloads\Task 2 Scenario 2\Task 2 Scenario 2\Task2Scenario2 Dataset 1_cosmetics.json" --jsonArray
```

```
C:\Program Files\MongoDB\Server\8.0\bin>mongoimport --db D597_Task_2 --collection groceries --file "C:\Users\marri\Downloads\Task 2 Scenario 2\Task 2 Scenario 2\Task2Scenario2 Dataset 2_Groceries_dataset.json" --jsonArray
2025-04-22T20:14:48.979-0400    connected to: mongodb://localhost/
2025-04-22T20:14:49.257-0400    38765 document(s) imported successfully. 0 document(s) failed to import.

C:\Program Files\MongoDB\Server\8.0\bin>mongoimport --db D597_Task_2 --collection cosmetics --file "C:\Users\marri\Downloads\Task 2 Scenario 2\Task 2 Scenario 2\Task2Scenario2 Dataset 1_cosmetics.json" --jsonArray
2025-04-22T20:15:17.135-0400    connected to: mongodb://localhost/
2025-04-22T20:15:17.170-0400    1472 document(s) imported successfully. 0 document(s) failed to import.
```

After executing the script, I verified the data was correctly inserted by connecting to the database using the mongo shell and running the following commands:

```
> show collections
< cosmetics
  groceries
> db.groceries.countDocuments()
< 38765
> db.cosmetics.countDocuments()
< 1472
D597_Task_2>
```

D3: Queries and D4: Optimization

In this section, I'll present three business questions that can be addressed using the newly constructed MongoDB database. For each question, I'll provide the corresponding query used to obtain the answer. Additionally, I'll discuss the indexes created to optimize query performance.

Question 1: How many orders were there for buttermilk?

The business is evaluating the demand for buttermilk to determine if it's worth continuing its production. To assess this, I executed the following query:

```
🕒 ▼ {itemDescription:"butter milk"}
```

The results of query 1:

Query Performance Summary

- 📄 **263** documents returned
- 📄 **38765** documents examined
- 🕒 **11 ms** execution time
- ↕ **Is not** sorted in memory
- 📄 **0** index keys examined
- ⚠️ **No index available for this query.** 💡

Note: It took 11ms to scan all 36,765 documents in the collection.

How I Optimized the Query: To improve query performance, I created an index on the itemDescription field. This allows MongoDB to quickly locate relevant documents without scanning the entire collection, making the queries run faster.

localhost:27017 > D597_Task_2 > groceries Open MongoDB

Documents 38.8K Aggregations Schema **Indexes 2** Validation

Create Index Refresh VIEWING INDEXES SEARCH INDEXES

Name & Definition	Type	Size	Usage	Properties	Status
> _id_	REGULAR	417.8 KB	4 (since Fri Apr 18 2025)	UNIQUE	READY
> itemDescription_1	REGULAR	249.9 KB	4 (since Fri Apr 18 2025)		READY

After re-running the query with the index, it scanned just 263 documents and completed in 0ms, showing a clear improvement in speed and efficiency.

Query Performance Summary

- 263 documents returned
- 263 documents examined
- 0 ms execution time
- Is not sorted in memory
- 263 index keys examined

Query used the following index:

itemDescription ↑


Query 2: How many orders include beef?


Maybe the company wants to keep an eye on how often beef is ordered, especially in case of a recall or supply issue. Here is the query I ran for that:


🕒 ▼ {itemDescription:"beef"}


The result:


Query Performance Summary



 **516** documents returned

 **38765** documents examined

 **12 ms** execution time

 **Is not** sorted in memory

 **0** index keys examined

 **No index available for this query.** 

Note: It took 12ms to run 38765 documents to return the answer.

How I Optimized the Query: To improve query performance, I used the same index on the itemDescription field that I created for Question 1. This index helps MongoDB quickly find matching documents without scanning the entire collection, which made this query run much faster too.

localhost:27017 > D597_Task_2 > groceries Open MongoDB

Documents 38.8K Aggregations Schema **Indexes 2** Validation






Create Index Refresh

VIEWING INDEXES SEARCH INDEXES

Name & Definition	Type	Size	Usage	Properties	Status
> _id	REGULAR	417.8 KB	4 (since Fri Apr 18 2025)	UNIQUE	READY
> itemDescription_1	REGULAR	249.9 KB	4 (since Fri Apr 18 2025)		READY

I ran the same script again and this was the result:

Query Performance Summary

-  **516** documents returned
-  **516** documents examined
-  **0 ms** execution time
-  **Is not** sorted in memory
-  **516** index keys examined

Query used the following index:

itemDescription ↑

This time only 516 documents were examined and returned and it was done in 0ms.






Query 3:How many moisturizers are labeled for sensitive skin?



To understand the inventory of moisturizers suitable for sensitive skin, I ran the following query:

```
{ Sensitive: 1 }
```

The result:

Query Performance Summary

-  **756** documents returned
-  **1472** documents examined
-  **1 ms** execution time
-  **Is not** sorted in memory
-  **0** index keys examined

 No index available for this query. 

Note: It took 1ms to to examine 1472 documents.

How I Optimized the Query: I created an index on the sensitive_1 field to speed up the search for moisturizers labeled for sensitive skin, which helped the query run faster and more efficiently.

localhost:27017 > D597_Task_2 > Cosmetics Open MongoDB

Documents 1.5K Aggregations Schema **Indexes 2** Validation

Create Index Refresh VIEWING INDEXES SEARCH

Name & Definition	Type	Size	Usage	Properties	Status
> _id	REGULAR	28.7 KB	3 (since Fri Apr 18 2025)	UNIQUE	READY
> Sensitive_1	REGULAR	24.6 KB	0 (since Fri Apr 18 2025)		READY

The results running the query again with the index:

Query Performance Summary

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

Query used the following index:

Sensitive ↑

Since this was already a very short query, the execution time stayed low at just 1ms, but the number of documents examined was noticeably smaller. As the database grows and more entries are added, this index will help prevent slower execution times in the future.

References

No external sources were used for this submission. All information came from official WGU course materials.