# Data Management — D597 Task 2 Report
**By Eric Williams**

## Part 1: Design Document

### A1: BUSINESS PROBLEM
For EcoMart in scenario 2, the data is currently stored in json files and aren't very accessible. Putting the data into a database will make the company's data much more useful for both accessing and querying. A few business problems I will solve later for the company using queries are:
1. How many orders does EcoMart get containing Whole Milk, and is it enough to be profitable?
2. How many moisturizers in their inventory are labeled oily, and is it too high of a proportion?
3. How many orders does EcoMart get that include chicken? How costly would it be if there was a recall on chicken?

### A2: DATABASE JUSTIFICATION and A3: DATABASE TYPE
Because the data is currently a json file, it is very compatible with NoSQL, MongoDB, and schema-less databases. Document Stores are an excellent tool for data with flexible schema requirements, especially for internet commerce platforms. In this exercise, I will create a database with two collections (Groceries and Cosmetics) with a flexible schema. This design makes much more sense than a relational database because the columns and entries in the two collections are not compatible or even very similar--they store very different information. Neither dataset needs to be altered to fit the other--with a flexible schema, they don't have to be altered.

### A4:DATA USAGE
The two collections in the database I will create will be useful for different purposes. The groceries collection will create a searchable database of all the products different members have bought. This could be useful for a variety of reasons, such as looking at trends in customer buying patterns or seeing how frequent certain customers shop.

The cosmetics collection, on the other hand, stores product data. It includes everything from brand names, rankings, descriptions, prices, and more about each product. Now that it is in the database, the business can easily search for higher ranking products, compare prices, look for trends in brand names and rankings, or any other question they can think of involving the products.

### B:SCALABILITY
The database is much more scalable now since it is organized. With the data just in a json file, it wasn't searchable or especially useful without importing it. Now that the data is organized into a

collection of documents, it would be much easier to scale it up or scale it down. New data could be added to the databases, or they could work to pare down the data using queries so that it just includes data they find useful. Also, I created indexes (described below) to make the searches much faster, which will also help with scalability when the database becomes more massive, allowing for fast searches despite the large size.

**C:PRIVACY AND SECURITY**

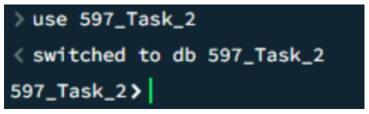
The company should address a few privacy concerns because the database contains sensitive information. The data tracks specific customers by ID and product and pricing data that could be dangerous if made public. To address this, the company should, at minimum, encrypt the data and require two factor authentication. It is also a best practice to do regular security checks.

**Part 2: Implementation**

In this section, I will describe how I built the database, ran queries to answer business questions, and optimized the queries for faster results. Here is my script on how to replicate the process.

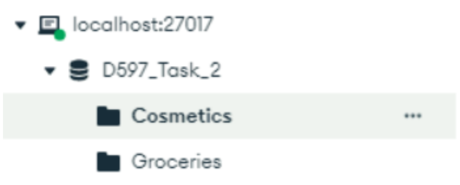
**D1: Database Instance**

The script:



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

The database instance:

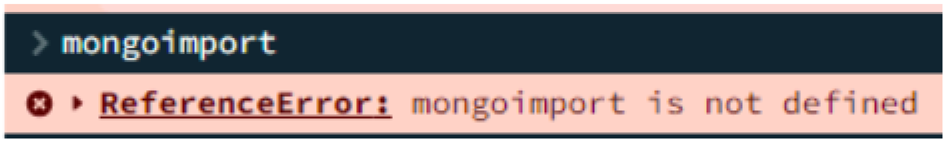


**D2: Insert Records**

The script:

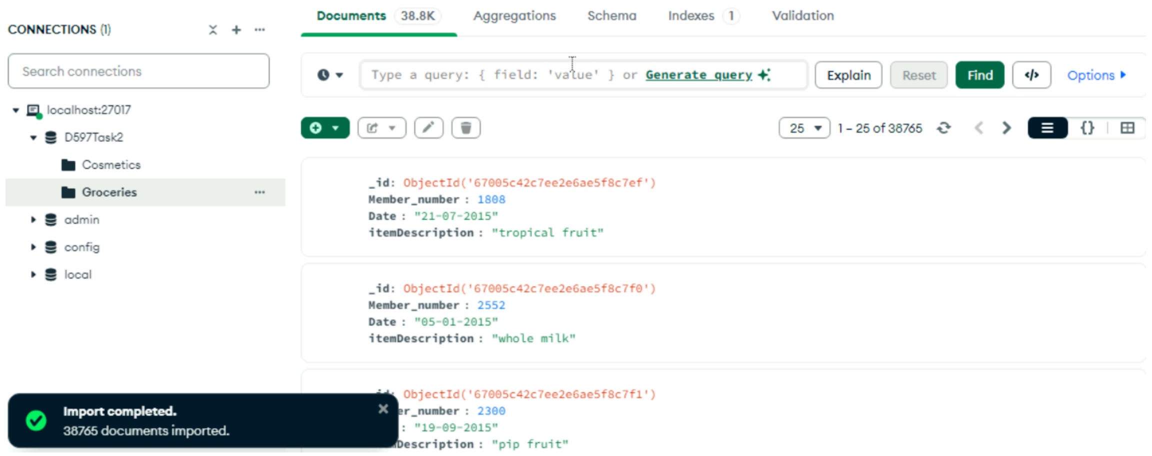
mongoimport --db db --collection groceries --file "C:\WGU\D597\Task 2\Scenario 2\D597 Task 2 Dataset 2_Groceries_dataset.json" --jsonArray mongoimport --db db --collection cosmetics --file "C:\WGU\D597\Task 2\Scenario 2\D597 Task 1_cosmetics_dataset.json" --jsonArray

However, it’s worth noting that mongoimport is essential to importing these files using the shell. Because I can’t install anything on the virtual machine, I got this error.



```
> mongoimport
✖ ▸ ReferenceError: mongoimport is not defined
```

Nonetheless, I imported the data using the import tool and performed the rest of the task. The screenshot below shows the data populated into Groceries.



```
_id: ObjectId('67005c42c7ee2e6ae5f8c7ef')
Member_number : 1808
Date : "21-07-2015"
itemDescription : "tropical fruit"

_id: ObjectId('67005c42c7ee2e6ae5f8c7f0')
Member_number : 2552
Date : "05-01-2015"
itemDescription : "whole milk"

_id: ObjectId('67005c42c7ee2e6ae5f8c7f1')
Member_number : 2300
Date : "19-09-2015"
itemDescription : "pip fruit"
```

Import completed.
38765 documents imported.

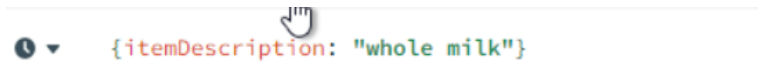
## D3: Queries and D4: Optimization

Here I will ask three business questions that can be answered with this newly constructed database. Below are the queries I used to solve them, as well as a screenshot showing that the query successfully executed. I will also cover the indexes I created to optimize the search.

**Question 1: How many orders were there for whole milk?**

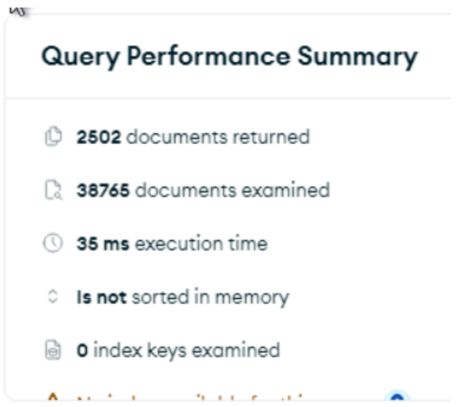
Perhaps the business is thinking of phasing out whole milk because they think there aren’t enough orders to justify production.

The Query:



```
{itemDescription: "whole milk"}
```

The results of the query:


Query Performance Summary

2502 documents returned

38765 documents examined

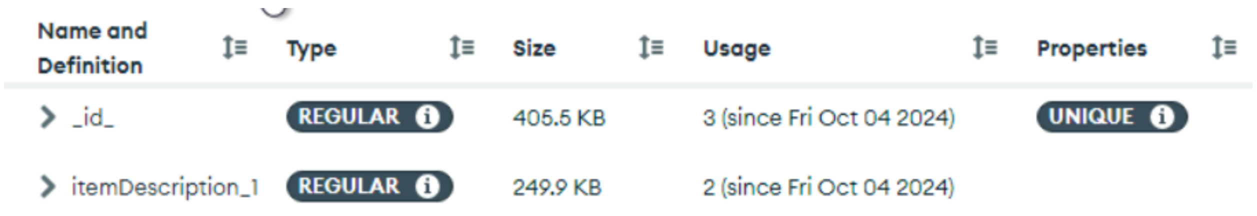
35 ms execution time

Is not sorted in memory

0 index keys examined

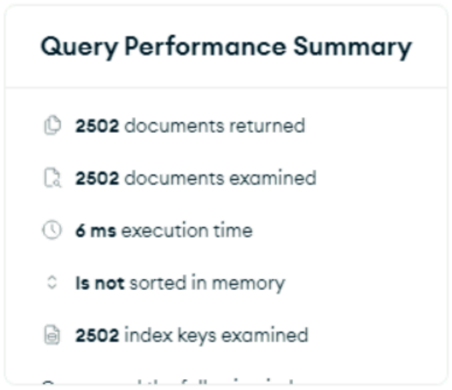
Note that there were 38,765 documents scanned and it took 35 ms.

**<u>How I optimized the query</u>**: I created an index for the item descriptions:



| Name and Definition | Type | Size | Usage | Properties |
|---|---|---|---|---|
| _id_ | REGULAR | 405.5 KB | 3 (since Fri Oct 04 2024) | UNIQUE |
| itemDescription_1 | REGULAR | 249.9 KB | 2 (since Fri Oct 04 2024) | |

I ran the same query again, and this time it only scanned 2502 documents and took 6ms, meaning the query and load times were now more optimized..


Query Performance Summary

2502 documents returned

2502 documents examined

6 ms execution time

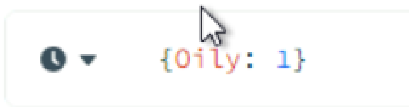
Is not sorted in memory

2502 index keys examined

## <u>Question 2 - How many moisturizers are oily?</u>

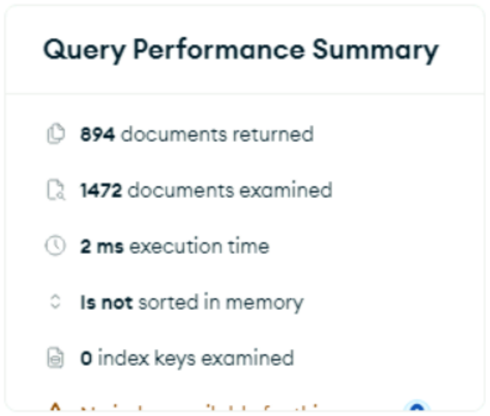
Perhaps the EcoMart is getting complaints that there are too many moisturizers that are oily and they want to see if most moisturizers are classified as oily.

The query:



```
{Oily: 1}
```

The result:



**Query Performance Summary**

**894** documents returned

**1472** documents examined

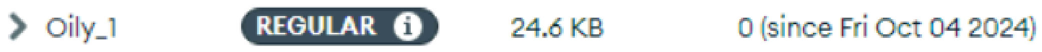
**2 ms** execution time

**Is not** sorted in memory

**0** index keys examined

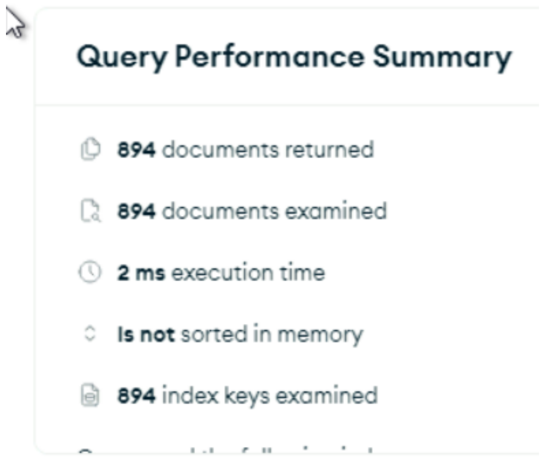
Note that 1,472 documents were examined and the execution took 2ms.

**<u>How I optimized the query: </u>** I created an index for moisturizers classified as oily.



Oily_1 REGULAR 24.6 KB 0 (since Fri Oct 04 2024)

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



**Query Performance Summary**

**894** documents returned

**894** documents examined

**2 ms** execution time

**Is not** sorted in memory

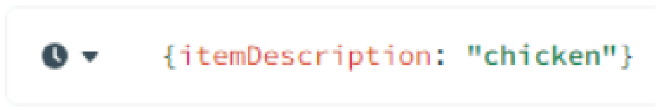
**894** index keys examined

Since this was already a very short query, the execution time remained at 2ms, but the number of documents examined is much smaller. In the future when there are many more entries to this database, the execution time will be significantly slower.

**Question 3 - How many orders include chicken?**

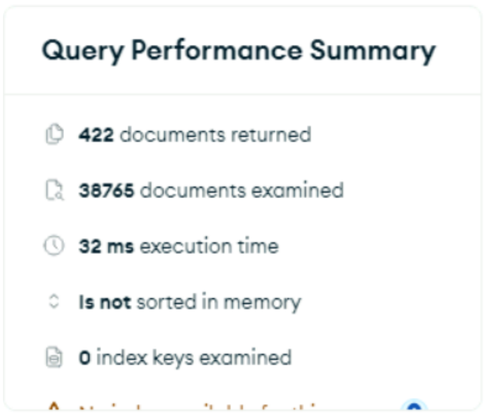
Perhaps there is a recall on chicken and EcoMart wants to check how many products need to be recalled. They could use this query to find out:

The query:



```
{itemDescription: "chicken"}
```

The result:



Query Performance Summary

422 documents returned

38765 documents examined

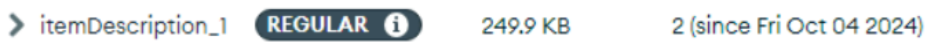
32 ms execution time

Is not sorted in memory

0 index keys examined

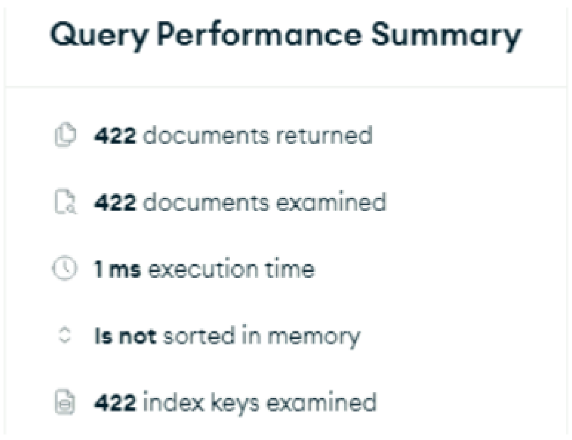
Note that 38,756 documents were examined and the execution took 32ms.

**How I optimized the query:** I created an index for the item description.



itemDescription_1 REGULAR 249.9 KB 2 (since Fri Oct 04 2024)

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



Query Performance Summary

422 documents returned

422 documents examined

1 ms execution time

Is not sorted in memory

422 index keys examined

The execution time was cut from 32ms to 1ms and the number of documents examined is only 422 instead of over 38,000. In the future when there are many more entries to this database, the execution time will be significantly slower.

## Sources

No sources were used except for WGU official course materials.