Practicum #3: Non-Relational Database

By

E.R.M.S.C Ekanayake

Scenario

This document focuses on a scenario of Panther logistic company. Panther Logistics currently operates in the logistics and supply chain industry. This company provides a broad range of services such as providing transportation, warehouses, packaging, order fulfillment, packaging, inventory management, and so on. Timely information is a critical factor for the success of this logistics company. So they have different types of data requirements such as supplier data, shipment data, inventory details, tracking information, etc. To successfully handle this, Panther Logistics must have a robust data infrastructure that can meet their data requirements. This company owns a massive data collection of their clients and suppliers, but they do not maintain an effective database for this. Therefore, they have happened to face different problems with this. In the warehouses, they maintain a traditional paper-based record-keeping system. Several times they had to handle some serious problems after misplacing some of the inventory records. As a result of these problems, Panther Logistics intends to build a new database system that can process massive amounts of real-time data efficiently and effectively. Also, Panther Logistics needs this new database system to analyze and identify the patterns of the logistic requirements to make smooth their operations. However, the choice of database architecture becomes a crucial consideration in this scenario.

NoSQL DB

Relational databases, while powerful for structured data, struggle with Panther's diverse data types and high-velocity requirements. Their rigid schema and fixed relationships limit flexibility and scalability. Hence, Panther Logistics needs a NoSQL database for its messy, fast-flowing data and NoSQL's flexibility and scalability are the perfect fit.

NoSQL DB is not a tabular database. Different forms of NoSQL databases can be identified based on the data model.

Document–based NoSQL

These types of databases store data in JSON-like documents to make them enable managing various data structures. Document-based NoSQL databases can handle unstructured, and semi-structured data as well. MongoDB, CouchDB, Riak, and Amazon Simple DB can be identified as examples of document–based NoSQL.

Column-based NoSQL

These databases organize data into columns instead of rows which enables them to handle large volumes of data and to support high-speed transactions. These databases can perform well in sum functions such as SUM, AVG, COUNT, and MIN. Cassandra, HBase, and Hypertable are examples of Column-based NoSQL.

Graph-based NoSQL

These databases are considered entity recreationists. They store data by considering each entity as a node and identify relationships as their edges. A distinct identifier is assigned for each entity node and edge. Neo4J, Graph, FlockDB, and Infinite are examples of this.

Why MongoDB is the Right NoSQL Choice for Panther Logistics?

Flexibility- Panther's data is a dynamic form of structured, semi-structured, and even unstructured formats. MongoDB's schema-less documents can effortlessly accommodate new data types and structures without being missed.

Scalability- Panther Logistics has an ever-growing data volume. Relational databases can underperform in this situation. But MongoDB scales horizontally which makes Performance remain smooth, even during peak seasons, ensuring shipments reach their destinations on time, every time.

Real-time data – The success of Panther Logistics relies on real-time tracking and dynamic route optimization. MongoDB's indexing capabilities and document structure allow Panther to track shipments, analyze logistics patterns, and optimize routes in real time. Decisions rely on real-time data, leading to faster deliveries, reduced costs, and happy customers.

Cost-Effectiveness - Its open-source availability and horizontal scalability often translate to lower costs compared to traditional relational databases. Additionally, its ease of use and reduced maintenance requirements further sweeten the deal, allowing Panther to focus on their core business, not database headaches.

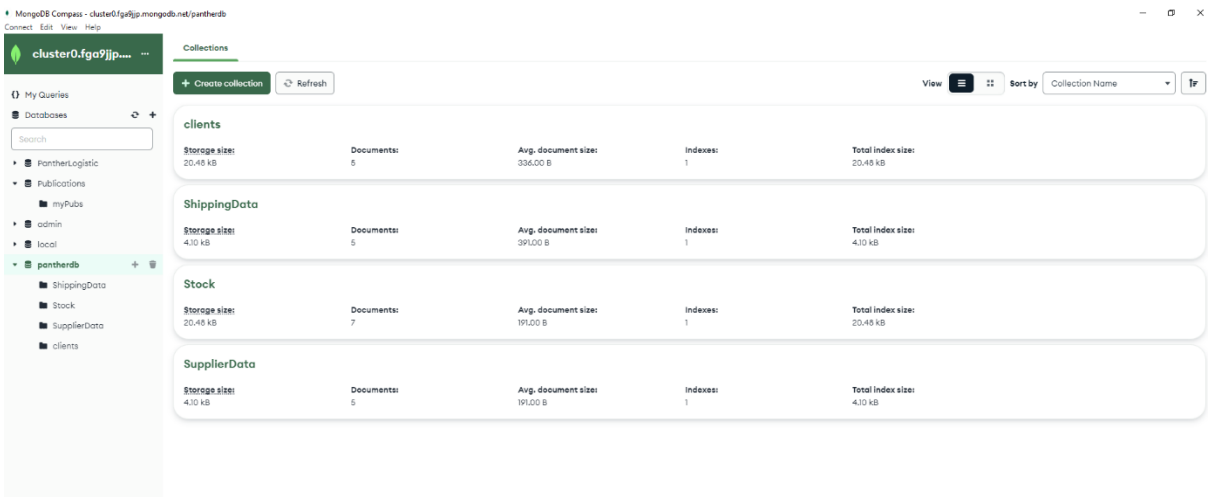
Further, we can compare MongoDB with other forms as follows.

Feature	Document (MongoDB)	Column (Cassandra)	Graph (Neo4j)
Flexibility	High (schema-less)	Moderate (flexible schema)	Low (fixed relationships)
Scalability	Horizontal scaling	Vertical scaling	Horizontal scaling (limited)
Real-time Data	Excellent	Good	Moderate (complex queries)
Data Structures	Diverse (JSON-like)	Structured & semi-structured	Relationships & entities
Performance	Good for queries	Excellent for large data sets	Good for graph-specific queries

When considering all the above factors which align with the data requirements of Panther Logistics and the comparison with other forms it proves that MongoDB is the best solution for this scenario.

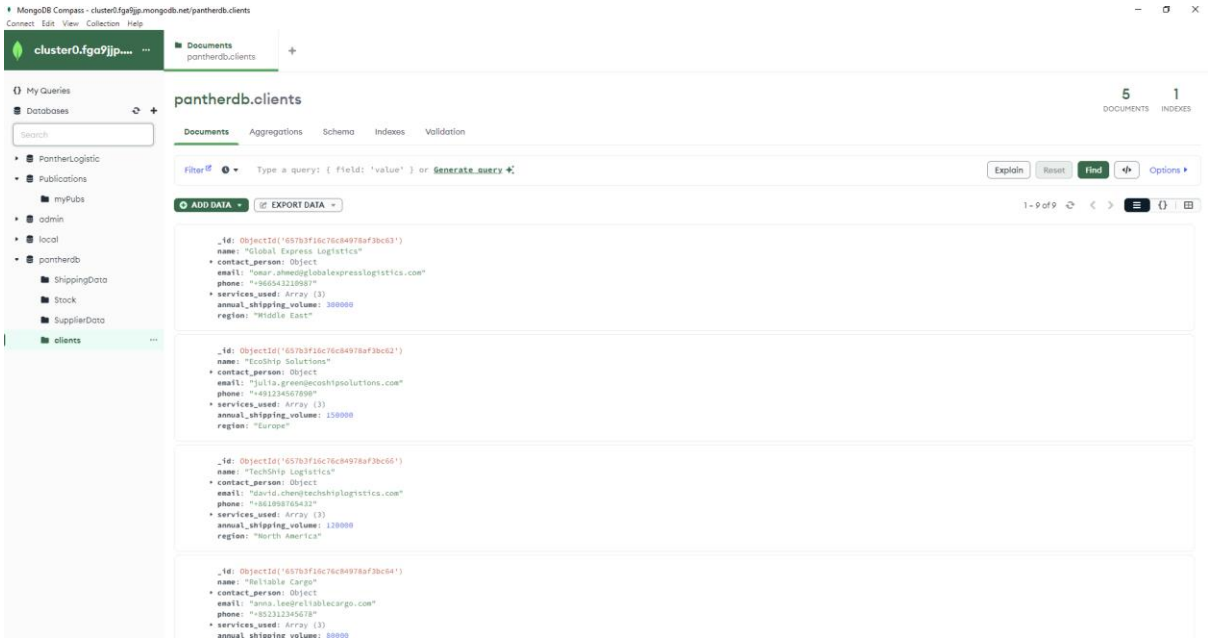
Barebones Prototype

By considering the requirements of Panther Logistics, the following database is built in MongoDB. In my prototype, there are four data collections under the database called “pantherdb”. Four collections are clients, stock, supplierdata and shippingdata.

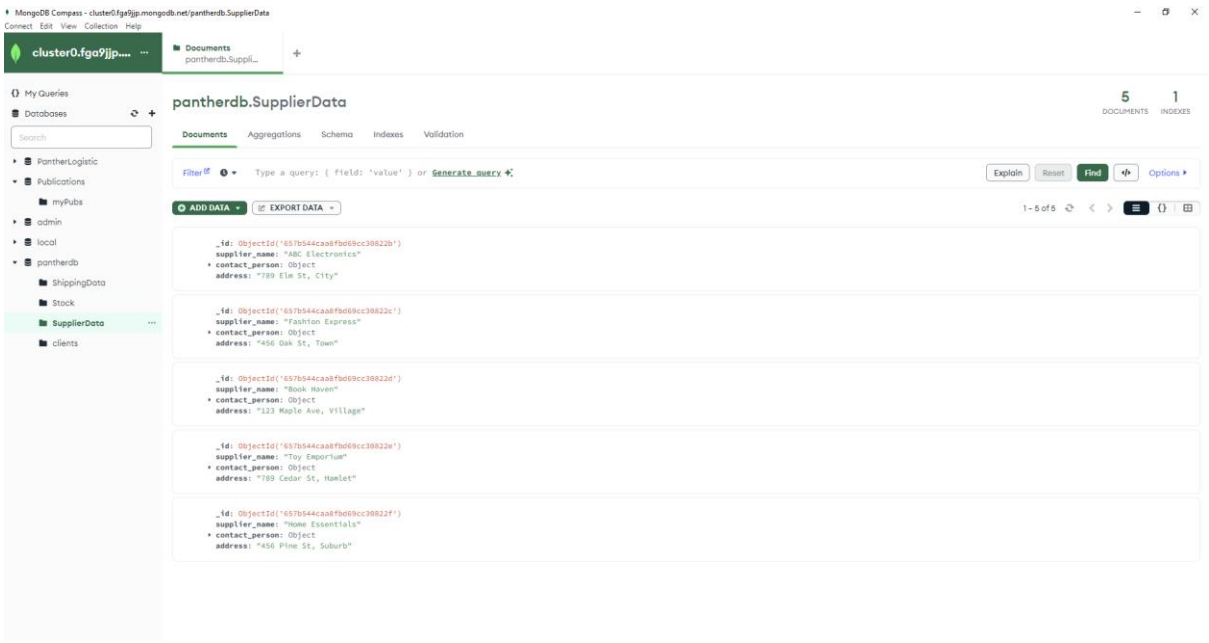


The following is summary of each data collection

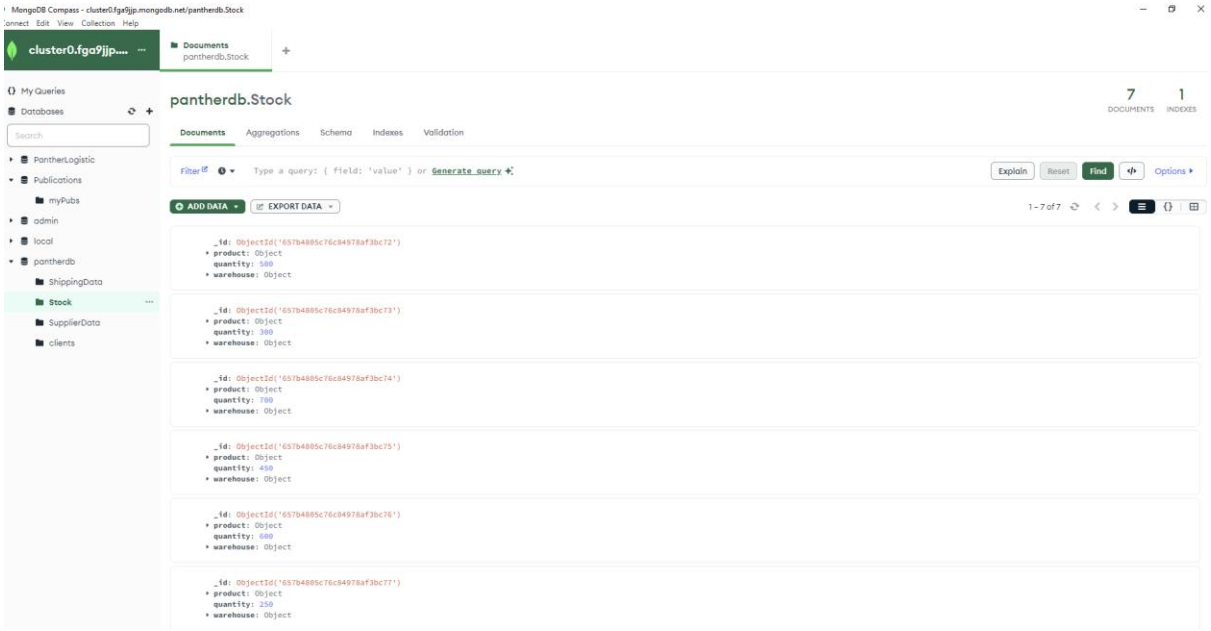
Clients- name, contact_person details, details of services used, shipping volume, region



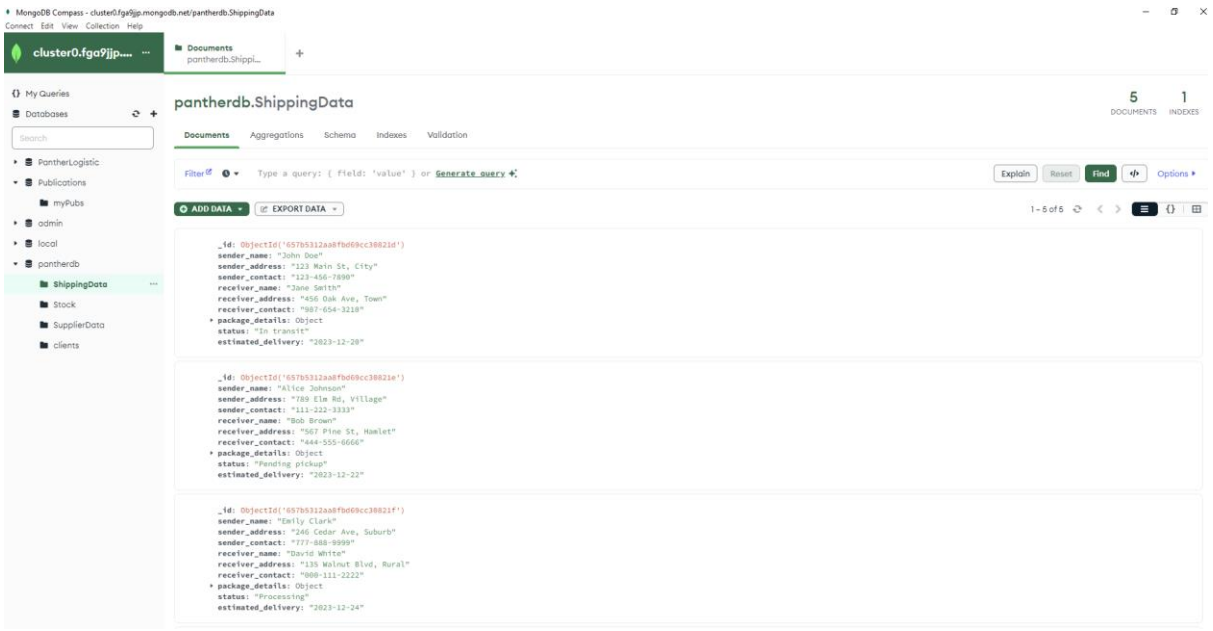
Supplierdata- name, contact information, address



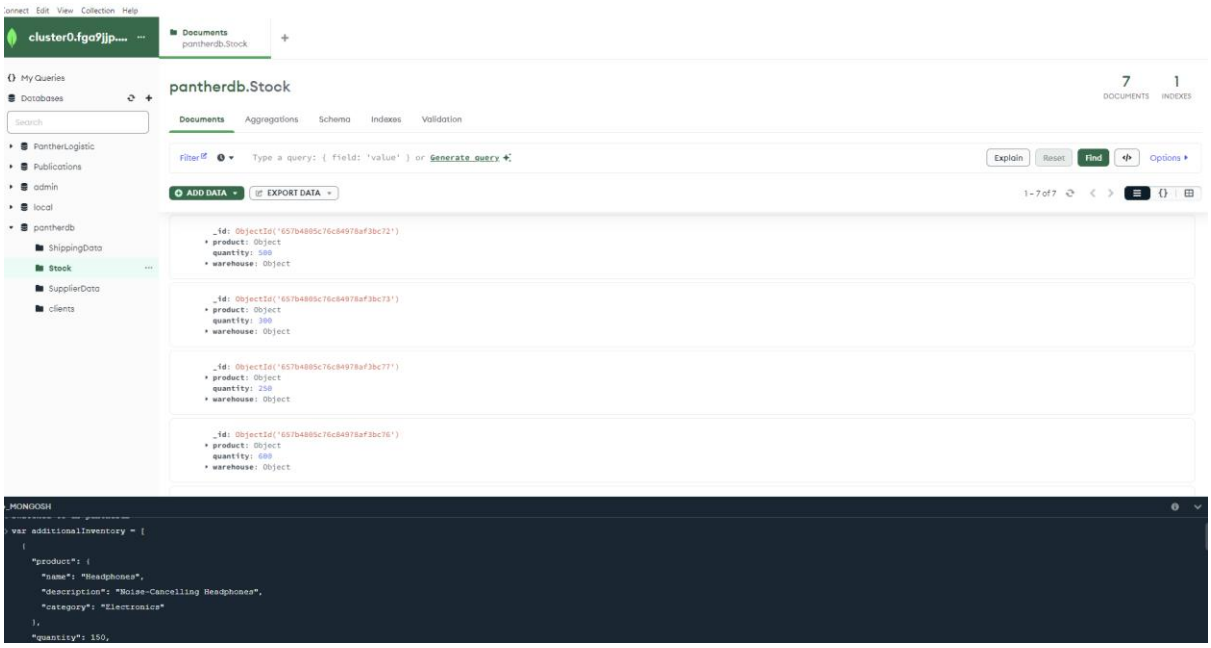
Stock- product details, quantity, warehouse details

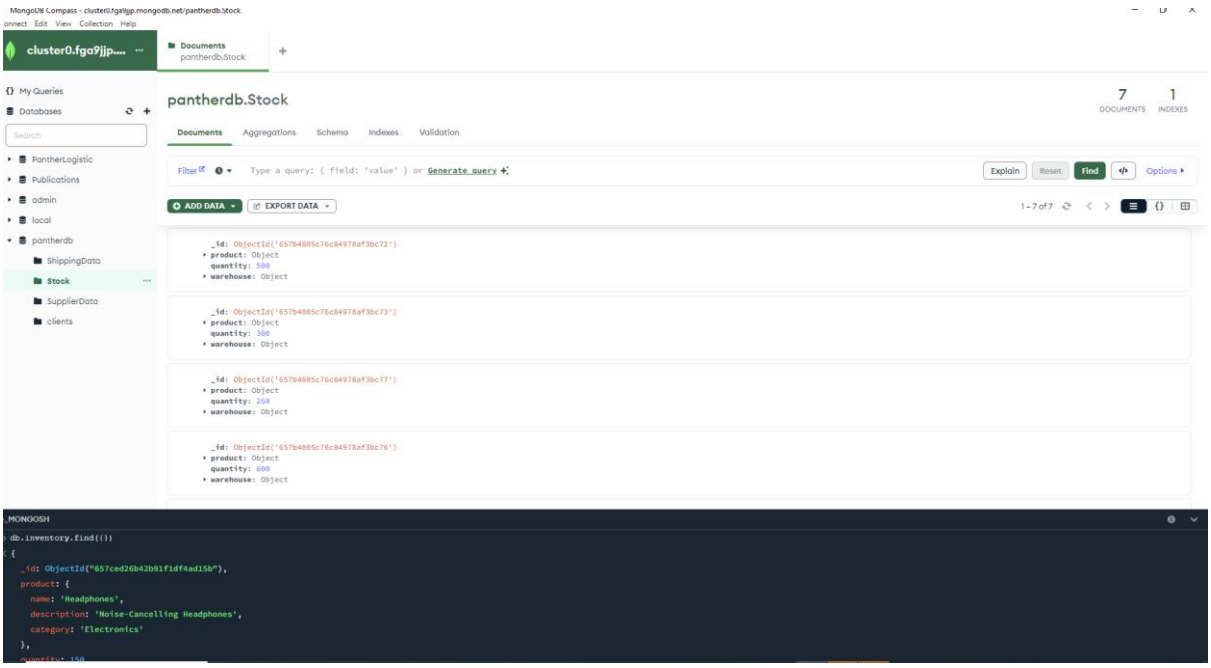


Shippingdata- sender details, receiver details, package details, status, estimated delivery

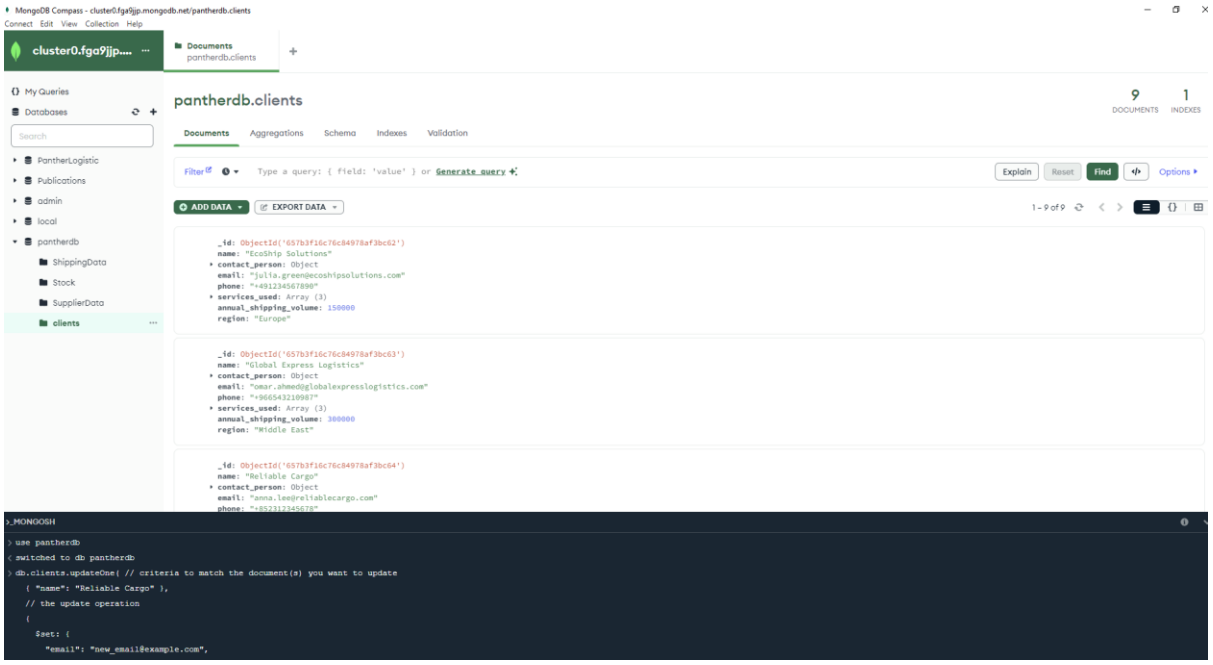


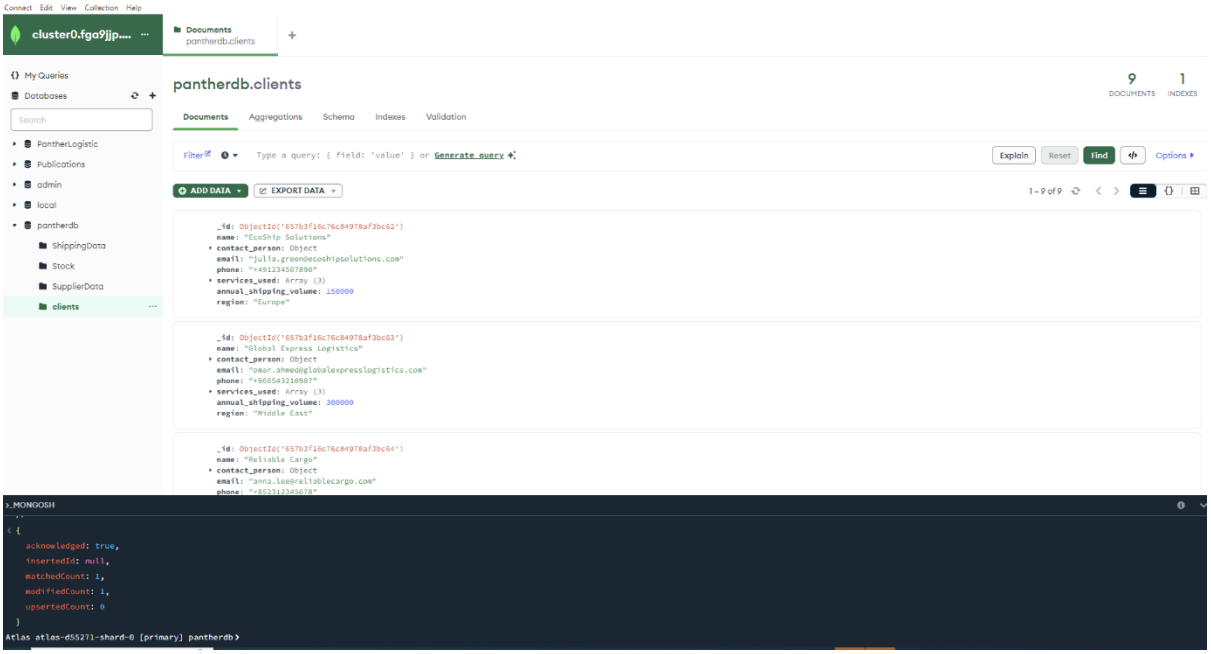
Add function





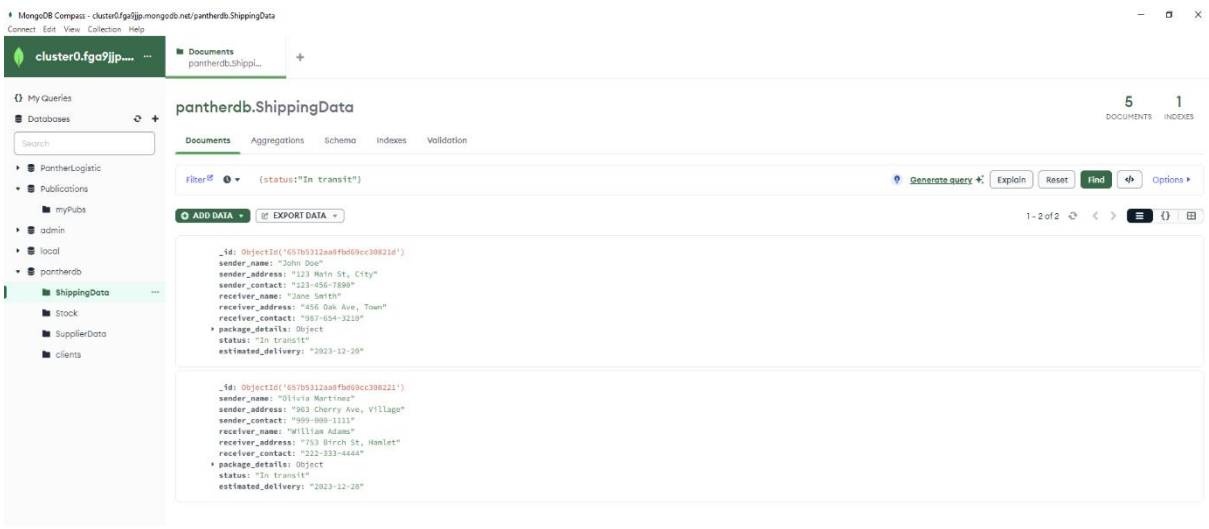
Update function





Filter Functions

Not like in panthers's traditional database, with MongoDB now they can have quick access to information as required. For example, by using Filter function they can check what are the shipments "In transit" by using the code {status:"In transit"}.



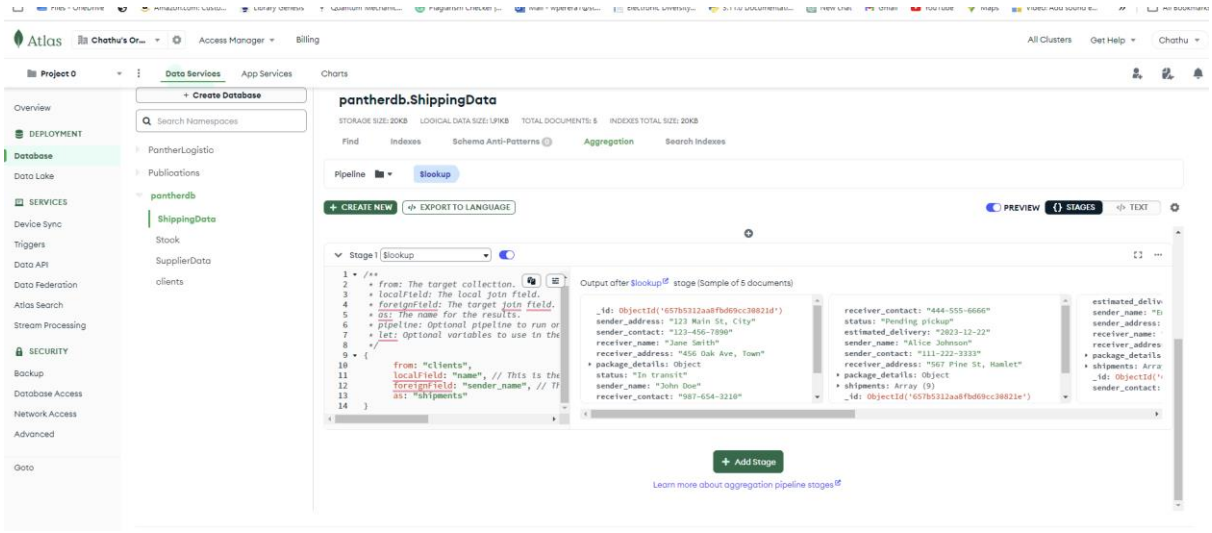
We can use other types of filter functions as below to fulfill the data requirements of Panther Logistics.

- $match: This operator filters documents based on specific conditions. Panther can use it to filter shipments by origin, destination, date, or other relevant criteria.
- $regex: This operator searches for regular expressions within document fields. Panther can use it to find shipments containing specific keywords like "fragile" or "urgent."
- $elemMatch: This operator filters documents based on conditions within an array field. Panther can use it to find shipments containing specific items or exceeding a certain weight limit.

Lookup Functions

$lookup: This operator performs joins between collections. For example here the code code acts like a bridge between the "clients" and "shipping_data" collections. It connects each client to their associated shipments based on their matching names, and it creates a new "shipments" field within each client document to store these linked shipments. This allows them to easily

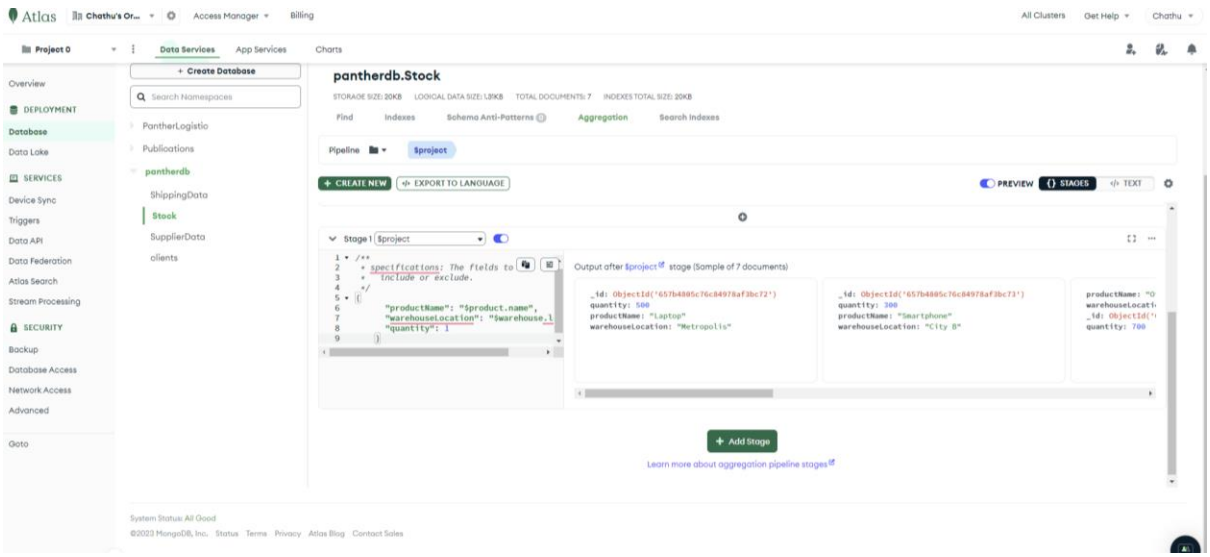
analyze client data alongside their corresponding shipping details in a single document, giving panther a more complete picture of their shipping activity.

- 

- $graphLookup: This advanced operator allows for multi-level joins across multiple collections. Panther can use it to explore complex relationships between shipments, suppliers, customers, and other entities within their data.
- $foreignField: This option within $lookup specifies the field in the "from" collection that references the documents in the "localField" collection. Panther can use it to ensure accurate joins between related collections.

Additional Functions:

- $project: This operator specifies which document fields to include or exclude in the query results. The code here code retrieves specific fields ("productName", "warehouseLocation", and "quantity") from the documents in the "inventory" collection and structures the output to include only these selected fields, making the result more focused and tailored to panther requirements.



- $unwind: This operator deconstructs an array field into separate documents for each element. Panther can use it to analyze individual items within a shipment or inventory list.
- $group: This operator aggregates documents based on shared characteristics. Panther can use it to group shipments by destination, calculate total weight or volume per region, and identify peak demand periods.
- $sort: This operator orders documents based on specified fields. Panther can use it to prioritize shipments based on urgency, delivery deadline, or other criteria.

Conclusion

MongoDB stands out as the best choice for Panther Logistics due to its flexibility, scalability, real-time capabilities, and cost-effectiveness. Its ability to handle diverse data types and high-velocity data aligns perfectly with Panther's needs, enabling efficient processing of massive real-time data and optimizing logistics operations effectively. Additionally, MongoDB's comprehensive functionalities, including filter, lookup, add, update and additional functions, offer tailored solutions to Panther's varied data requirements, ensuring smooth operations and informed decision-making.

References:

- https://www.mcgill.ca/continuingstudies/area-of-study/supply-chain-management-and-logistics
- https://fulltiltlogistics.com/what-is-a-logistics-company/#:~:text=Logistics%20companies%20plan%2C%20implement%2C%20and,on%20a%20client's%20logistical%20needs.
- https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-nosql/#:~:text=What%20Is%20NoSQL%3F-,NoSQL%20databases%20are%20non%2Dtabular%20and%20handle%20data%20storage%20differently,column%2C%20and%20key%2Dvalue.
- https://www.toptal.com/database/the-definitive-guide-to-nosql-databases
- https://www.youtube.com/watch?v=0MZFTiKIPnU&ab_channel=MongoDB