

NO SQL APPLICATION FOR MACY'S BUSINESS



MOHAMED NAUSATH FATHIMA ILMA

APIIT Code: CB013369 — SU Code: 23039813

DECISION ANALYTICS

JUNE 11, 2024

Abstract

When analyzing the Macy's retail store, the current key problem they are facing is to optimize inventory allocation across Macy's stores in the apparel section. However, they are facing many challenges such as an increase in consumer demand, changes in market trends, operational inefficiencies, and more! Hence, to effectively solve, analyze and provide key strategies, data-driven approaches and the data pipeline method to solve the business problem by implementing a NoSQL application for Macy's business through the MongoDB family of database by using tools such as Mongoosh shell, and MongoDB Compass. Thus, it would help to identify the crooks and crannies to enhance inventory management practices, and strategic opportunities for Macy's to improve forecasting accuracy, minimize stockouts, and maximize sales potential.

Acknowledgement

I would like to express my sincere gratitude to Dr. Rajitha Nawaratne for being a great mentor, providing exceptional guidance and teaching on all the essential aspects to successfully complete this report. He showcased excellent organization and taught lectures effectively and punctually with concise explanations, and valuable practical problem-solving techniques. Dr. Nawaratne consistently made himself accessible to address any doubts or concerns with the subject area.

Contents

1	Introduction	2
1.1	Background Study About Macy's Business	2
1.2	Problem Context Related To Macy's Business	4
1.2.1	Key Questions	5
1.3	Data Analysis Which Can Be Done To Solve The Problem	5
1.4	Identifying The Challenges In Solving The Problem Of Macy's Stores For The Apparel Section	7
1.5	Solving The Problem Using Data Pipelines	9
2	Technical Tools Used to Analyse Macy's Problem Context	11
2.1	Usage Of SQL And NoSQL Pipelines	11
2.1.1	SQL Pipelines	11
2.1.2	NoSQL Pipelines	13
2.2	What Is NoSQL?	13
2.2.1	Characteristics Of NoSQL	14
2.2.2	Applications Of NoSQL	15
2.2.3	Strengths And Limitations Of NoSQL	16
2.3	What Is MongoDB?	17
2.3.1	Characteristics Of MongoDB	17
2.4	What Is MongoDB Shell?	18
2.4.1	The Applications Of MongoDB Shell	19

2.4.2	Strengths And Limitations Of Mongo Shell	19
2.5	What Is MongoDB Compass?	21
2.5.1	Characteristics Of MongoDB Compass	21
2.5.2	Strengths And Limitations Of MongoDB Compass	22
3	Creating Data Engineering Pipelines using MongoDB To Solve Macy Store's Problem	24
3.1	Database Design For Macy's Business	24
3.2	Mongoosh Shell Commands Used For Macy's Business	26
3.2.1	How To Create A Database In MongoDB?	26
3.2.2	To Show All The Existing Database	27
3.2.3	To Show The Currently Working Database	28
3.2.4	Usage Of 'use' Command Or Ways To Create A Database	28
3.2.5	To Create A Collection	29
3.2.6	To Show All Collections Inside The Working Database	30
3.2.7	To Insert One Or More Documents Into The Collection	31
3.2.8	To View All Documents	35
3.2.9	To View All The Formatted Documents	35
3.2.10	To Find Specific Documents	36
3.3	To Analyse Whether The Database Is Functioning Properly	38
3.3.1	To Sort Documents Inside The Specific Collection	38
3.3.2	To Count The Documents Inside The Collection	40
3.3.3	To Limit The Number Of Documents To Be Displayed Inside A Specific Collection	41
3.3.4	To Sort And Limit The Number Of Documents Displayed As Output In A Specific Collection (Chaining)	42
3.3.5	To Find One Document Inside A Specific Collection	45
3.3.6	To Find Documents With Specific Fields	46
3.3.7	Update One Document	47

3.3.8	Increment The Field Value Of One Or More Than One Document . . .	48
3.3.9	Rename Fields In Documents	49
3.3.10	Delete One Or More Documents	50
3.3.11	Adding Index To The Collection	51
3.3.12	Using Text Search For The Created Index	51
3.3.13	To Find By Element In An Array In A Collection	52
3.3.14	To Find Documents Greater Or Less Than Specific Values Of Specific Fields	53
3.3.15	Usage Of \$regex Command In Documents	57
3.3.16	Usage Of \$in Command In Documents	58
3.3.17	Usage Of \$ne Command In Documents	59
3.3.18	Usage Of \$nin Command In Documents	60
4	Using MongoDB Compass To Analyze the Key Questions To Solve The Prob- lem	62
4.1	Analysis On Key Question 1	62
4.2	Analysis On Key Question 2	64
4.3	Analysis On Key Question 3	66
4.4	Analysis On Key Question 4	69
4.5	Analysis On Key Question 5	71
5	Discussion on Hypothesis related to Macy's Store	74
5.1	Hypothesis 1: Customers Of Chicago, IL' Are Richer Than Detroit, MI.'	74
5.2	Hypothesis 2: Seasonal Months Have More Sales	75
5.3	Hypothesis 3: The Top 5 Products Bring More Sales	76
5.4	Hypothesis 4: Younger Customers Make More Purchases Than Older Customers	77
6	Conclusion	79

List of Figures

1.1	The Logo of Macy's Business	2
1.2	Website view of Macy's Business	3
1.3	Data Pipeline	9
2.1	SQL Relational Database	12
2.2	The key characteristics of NoSQL	14
2.3	Logo of MongoDB Application for understanding	17
2.4	The MongoDB Shell is a command-line interface (CLI) tool	18
3.1	Database Design for Macy's Business	25
3.2	Navigation to the location where the mongosh.exe file is located	26
3.3	MongoDB CLI depicting the connection to localhost	27
3.4	Displays all the existing databases in the connected localhost server	28
3.5	Displays the currently working database connected to the localhost server . . .	28
3.6	Creates a new database called "MacyStore"	29
3.7	Creates a new collection called 'ProductCatalog'	30
3.8	Displays all the collections existing inside the 'MacyStore'	30
3.9	Insert one document into the "ProductCatalog" collection	32
3.10	Insert three documents into the "ProductCatalog" collection	34
3.11	Displays all the existing documents inserted inside the ProductCatalog collection	35
3.12	Displays all the existing documents inserted inside the ProductCatalog collection in a JSON-like format	36

3.13 Finds and outputs the documents where the location.address field	37
3.14 Finds and outputs the documents where the store_name field matches	38
3.15 List of documents where the price field is sorted in ascending order	39
3.16 List of documents where the price field is sorted in descending order	40
3.17 Displays the number of documents available inside the StoreInventoryDetails collection	41
3.18 Displays only the first two documents stored inside the “StoreInventoryDetails” collection	42
3.19 Sorting of price field in ascending order	43
3.20 Displays the sorting of price field in descending order	44
3.21 Displays the list of store_name field along with the text “Store Name”	45
3.22 Displays the specific address specified in the command	46
3.23 Displays the documents having the fields store_name and inventory_levels.product_id fields	47
3.24 Updates the document based on the specified command	48
3.25 Displays the result of incrementing the field values	49
3.26 Rename fields in documents	50
3.27 Delete one or more documents	50
3.28 Displays the command and result of creating search index	51
3.29 Using text search for the created index	52
3.30 Finding by element in Array in the collection	53
3.31 Usage of \$gt command	54
3.32 Usage of \$gte command	55
3.33 Usage of \$lt command	56
3.34 Usage of \$lte command	57
3.35 Using \$regex command in documents	58
3.36 Using \$in command in documents	59
3.37 Using \$ne command in documents	60

3.38	Using \$nin command in documents	61
4.1	Analyzing the prices of Products	63
4.2	Result of analyzing the prices of Products	64
4.3	Analyzing a specific supplier's products	65
4.4	Results after analyzing a specific supplier's products	66
4.5	Analyzing the CustomerPreference collection	67
4.6	Results during the analysis of the CustomerPreference collection	68
4.7	Products interested by the target demographics	69
4.8	The amount of sales occurrence	70
4.9	Usage of sales query in MongoDB Compass	71
4.10	Analyzing the Store Inventory levels	72
4.11	Results after analyzing the Store Inventory levels	73
5.1	Results of Hypothesis 1	75
5.2	Results of Hypothesis 3	76
5.3	Results of Hypothesis 3	77
5.4	Results of Hypothesis 4	78

List of Tables

1.1	Identifying the type of analysis and the expected outcome to solve the problem using the proposed NoSQL Application	6
1.2	Identifying the type of challenges to be faced and methods to overcome the challenge	8
2.1	Strengths and Limitations of NoSQL	16
2.2	Strengths and Weakness of Mongo Shell	20
2.3	Advantages and disadvantages of MongoDB Compass	22

Chapter 1

Introduction

1.1 Background Study about Macy's Business

Macy's is one of the **most iconic names in American retail**, Macy's Inc Macy (2023). Its headquarters is located in New York City. Currently, it is one of the cornerstones of the nation's consumer landscape. It was founded in 1858 by Rowland Hussey Macy and today has surpassed different stages including department stores, digital platforms, and spanning over towards exclusive brands.



Figure 1.1: The Logo of Macy's Business

Macy's has a **decade of ups and downs** including economic, technological, and changes in consumer preferences. However, it has bloomed to be one of the best retail industries in the market.

They have a **wide range of product categories** but are not limited to:

- Fashion
- Beauty
- Home
- Kitchen
- Dining
- Luggage
- Health and wellness
- Electronics
- Apparels

Alongside, they partner with worldwide brands and have private brands such as Bloomingdale's, Bloomingdale's The Outlet, Bloomies, Market by Macy's, Macy's Backstage, and Bluemercury.

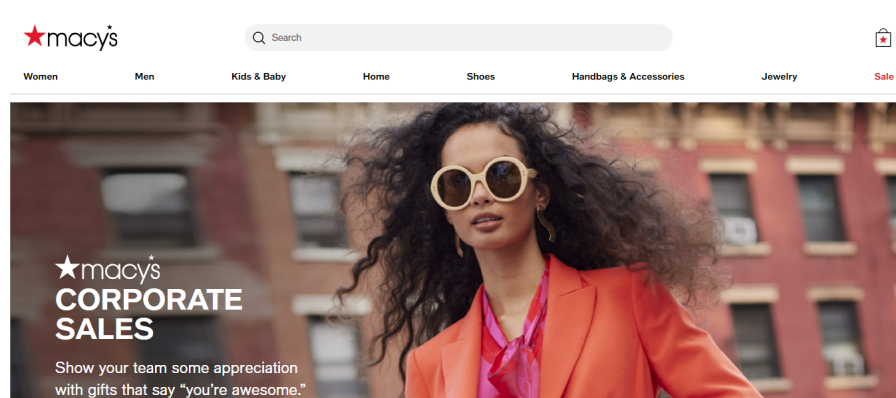


Figure 1.2: Website view of Macy's Business

1.2 Problem Context Related to Macy's Business

Even though they are a fast-paced retail sector selling a wide range of products, one of the **key challenges** they face is to **maintain optimal inventory levels** Alvin.Magestore (2018). Since they have a wide range of categories of products and many stores (particularly in the apparel section), **inventory management has been challenging**.

Due to a lack of a proper mechanism to track and maintain the inventory levels, they have faced the following **problems**:

- Experience fluctuations in consumer demand
- Economic pressures
- Inability to meet supplies
- Inability to place on-time orders from suppliers based on demands.
- Shift in market trends
- Lack of proper data-driven inventory management.

Thus, an **effective data and analytics solution** will help Macy to forecast sales demand, optimize receipt timing from suppliers, identify the high sales periods, and enhance supply chain efficiency.

Considering the **problem context** of Macy to **refine the inventory management strategies and efficiency in the apparel section**, helps to overcome the above-listed problems.

To overcome these challenges, this study focuses on **developing a NoSQL application to optimize inventory allocation specifically for the apparel section**. Thus, data analytics, automation, and flexible inventory allocation strategies can be implemented by Macy's to enhance their business.

1.2.1 Key Questions

The following are the key questions which helps to solve the problem of Macy's store:

- How to the analysis of product prices within a specific range help Macy's maintain optimal inventory levels?
- What insights can be obtained by analyzing a specific supplier in managing inventory more effectively?
- How to identify the Customer Preference to identify potential fluctuations in consumer demand?
- How to optimize inventory management during high sales periods?
- How to analyze the store inventory levels to prevent the inability to meet supplies and place on-time orders?
- How can data-driven insights from these analyses enhance supply chain efficiency and address economic pressures?

1.3 Data Analysis which can be done to solve the problem

Based on the problem context of lack of optimizing strategy for inventory allocation in Macy's stores for the apparel section, the below are the **expected data analysis which can be used to solve the problem**:

Table 1.1: Identifying the type of analysis and the expected outcome to solve the problem using the proposed NoSQL Application

Type of analysis required to solve the problem	Metrics used for the Analysis
Historical Sales Analysis	<ul style="list-style-type: none"> • Analyze historical sales data of the apparel products to identify product trends, seasonality, and popular items. • Ability to identify sales patterns and revenue over time to identify the demand.
Inventory Turnover Analysis	<ul style="list-style-type: none"> • Identify the inventory turnover rates for apparel products and stores. • Identify slow-moving inventory items and stores with high inventory levels to balance the inventory effectively.
Demand Forecasting	<ul style="list-style-type: none"> • Implement marketing activities based on seasonal trends, promotions, and external events. • Optimize inventory levels and prevent stockouts or excess inventory.
Customer Segmentation	<ul style="list-style-type: none"> • Identify customer segments based on purchasing power, preferences, and demographics. • Understand the preferences for apparel products.
Geospatial Analysis	<ul style="list-style-type: none"> • Use geospatial data to analyze regional analysis for apparel products

1.4 Identifying the Challenges in Solving the Problem of Macy's Stores for the Apparel Section

The table includes the challenges addressed to be faced while solving the problem context of Macy's Store and how they can be addressed:

Table 1.2: Identifying the type of challenges to be faced and methods to overcome the challenge

Challenge determined	Explanation on the challenge determined to be faced	Addressing methods to overcome the challenge
Data Integration	Macy's business stores data in various systems which are located in different departments and locations.	To overcome this challenge, using integration tools helps to streamline the steps to transform the data from various sources.
Demand Forecasting	Predicting customer demands for apparel products accurately is challenging as the market trends and consumer preferences not constant.	Advanced analytics techniques such as machine learning models are handy for analyzing historical sales data, demographic information, seasonal trends, and external factors accurately.
Inventory Optimization	Balancing inventory levels located in different stores to meet customer demand, costs and stockouts is a complex optimization issue.	Inventory optimization algorithms can be used to identify the optimal allocation of inventory based on several factors such as sales velocity, store location, seasonality, and product popularity.
Supply Chain Coordination	To effectively manage the inventory allocation, it is important to coordinate it effectively with suppliers.	Collaborative supply chain management tools and practices, such as vendor-managed inventory (VMI) and just-in-time (JIT) inventory systems can be used.
Performance Monitoring	It is challenging to continuously monitor the key performance metrics and identify areas for improvement.	Real-time analytics dashboards and automated alerts can be used to track Macy's inventory levels, sales performance, and other relevant metrics to handle inventory allocation strategies.

1.5 Solving The Problem Using Data Pipelines

Since the key question were addressed in the problem context of the Macy Store, data pipelines needs to be used by the data analysts or data scientists accordingly.

Data pipelines is a process of transforming the data under series of stages IBM (n.d.). It helps to obtain the data from various sources and makes it easier to clean, aggregate, and analyze information. However, MongoDB provides a wide range of functions to streamline the process including advanced querying capabilities, indexing, and aggregation capabilities.

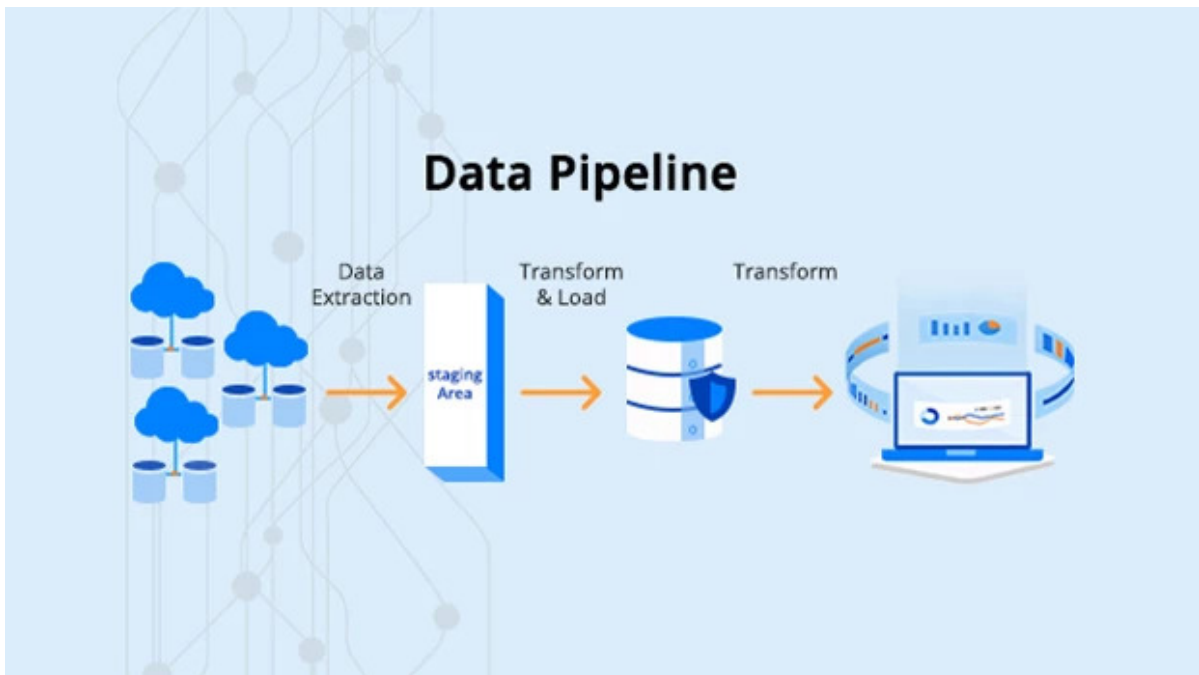


Figure 1.3: Data Pipeline

By using the MongoDB's insert, update, find, and delete operations and other query options, datasets can be managed effectively. Alongside, indexing helps to optimize the data query performance, and perform the aggregation, complex data transformations and analyses effectively. Also, it supports administrative commands to manage the database and collection structures through data pipelines and streamline the work process.

Specifically, the following MongoDB tools Simplilearn (2023) will play a key role in the integral to our data pipeline solution/NoSQL Application:

- MongoDB Shell
- MongoDB Compass

By combining these MongoDB tools with a well-structured data pipeline, the Macy's NoSQL Application will be built to solve the key questions and obtain meaningful insights from the data.

Chapter 2

Technical Tools Used to Analyse Macy's Problem Context

2.1 Usage of SQL and NoSQL Pipelines

SQL and NoSQL pipelines are specific implementation methods which are available in the broader level of data pipelines. These pipelines can be used in various types of pipelines implementations based on the data processing requirements.

2.1.1 SQL Pipelines

SQL pipelines includes the usage of SQL databases such as MySQL, PostgreSQL, or SQL Server for data processing activities Dataiku (n.d.). SQL databased are used in traditional relational data processing activities. The following are the main types of SQL database pipelines:

- **Batch Processing Pipelines:** Generally, SQL databases are used for batch processing tasks. They help to execute complex queries on large datasets.
- **ETL Pipelines:** SQL is also used for data transformation process in the ETL processes. In this process, the data is queried and transformed using SQL operations. All

these activities are performed before loading the process into another database or data warehouse.

- **Data Integration Pipelines:** Data Integration pipelines in SQL helps to combine data from different relational databases. Also, they help to perform activities such as joins, unions, and other operations to create a unified dataset.

What Is SQL?

SQL (Structured Query Language) is a standard programming language which is used to manage relational databases Services (n.d.). It is important to perform operations such as query, update, and manage data in relational database management systems (RDBMS). However, SQL helps to perform wide range of operations, to create and modify database structures.



Figure 2.1: SQL Relational Database

The following are the key characteristics of SQL:

- Uses declarative language to help users identify what they want based on the data
- is a standard query language which ensures to maintain consistency in various RDBMSs.
- Data Definition Language (DDL) such as CREATE, ALTER, and DROP to define database structure.

- Uses Data Manipulation Language (DML) such as SELECT, INSERT, UPDATE, and DELETE to manipulate data.
- Uses Data Control Language (DCL) such as GRANT and REVOKE to support control access to data.

2.1.2 NoSQL Pipelines

NoSQL pipelines includes the usage of NoSQL databases such as MongoDB, Cassandra, and Redis DB (n.d.). They are widely suitable for scenarios where data of unstructured or semi-structured format is used. Also, they are used in modern, scalable data processing activities to perform various types of data pipelines:

- **Stream Processing Pipelines:** NoSQL databases are used for stream processing of pipelines consisting of high write throughput such as Cassandra or Redis.
- **ELT Pipelines:** NoSQL databases are useful to store raw data in ETL pipelines. Also, it is used before transformation steps are applied within the same database or based on the usage of external tools.
- **Data Lake Pipelines:** NoSQL databases can be used as data lakes to store large amount of unstructured data.

However, in this problem context of Macy's Store, the student will be using **NOSQL data pipelines** by integrating with **MongoDB** tool.

2.2 What is NoSQL?

NoSQL stands for "**Not only SQL**". It is the technical term which helps to describe a specific category of databases.



Figure 2.2: The key characteristics of NoSQL

The **key feature of NoSQL databases** is that it **stores data in unstructured or semi-structured data**. Thus it is completely different from Traditional Database Management Systems (RDBMS).

2.2.1 Characteristics of NoSQL

The following are the **key characteristics of NoSQL**:

- Supports flexible schema which supports both dynamic and ad-hoc changes.
- Ability to scale horizontally to distribute data across many servers.
- Supports high availability due to built-in features aiding replication, fault tolerance, and reliability.
- Supports high performance related to real-time analytics, high-speed transactions, or content management.

- Supports a wide range of data models such as key-value stores, document stores, column-family stores, and graph databases.
- Well suited to be used to handle big data applications.

Hence, since MongoDB uses NoSQL, it is considered highly effective, and reliable by supporting high performances and efficiency in applications.

2.2.2 Applications of NoSQL

- **Big data analytics:** NoSQL databases are used in big data applications to maintain the stores and process large volumes effectively.
- **Real-time web applications:** NoSQL databases are widely used in real-time web applications, such as social networks, gaming, IoT and more systems.
- **Content management systems (CMS):** NoSQL databases are used in CMS platforms to store unstructured data such as text, images, and multimedia files.
- **E-commerce platforms:** NoSQL databases are used in e-commerce applications to manage product catalogs, user profiles, and transactional data.
- **Mobile app development:** In mobile app developments, NoSQL is used in backends to handle large user bases, offline synchronization, and data models.

2.2.3 Strengths and Limitations of NoSQL

The table discusses the strengths and limitations of NoSQL:

Table 2.1: Strengths and Limitations of NoSQL

Strengths of NoSQL	Limitations of NoSQL
<ul style="list-style-type: none"> • NoSQL databases can scale horizontally. Thus, it can store large and scaling data volumes and user loads. • NoSQL databases support a wide range of flexibility to handle schema designs and support the changing data requirements. Also, it helps to overcome the issue of using expensive schema migrations. • NoSQL databases support high throughput and low latency when handling read-heavy and write-heavy documents. • NoSQL databases are cost-effective as it consumes less hardware infrastructure and consists less operational costs. • Many NoSQL databases have built-in replication and automatic failover mechanisms. Thus, it helps to ensure high availability and data durability. 	<ul style="list-style-type: none"> • Even though NoSQL databases have a wide range of scalability, they cannot handle ACID transactions. Thus, it leads to data inconsistency issues in many scenarios. • NoSQL databases support less powerful query opportunities. Also, it supports less complex SQL-like queries which makes the data analysis and report handling to be challenging. • NoSQL databases have less mature tools and libraries compared to relational databases. Thus, when compared to other databases, it is considered to be less productive to the developer or the handler. • It is challenging to design effective data models for NoSQL databases compared to relational databases to effectively handle data modelling. • Also, in NoSQL, it is complex to maintain data relationships straightforwardly.

2.3 What is MongoDB?

MongoDB is an **open-source application** which includes a NoSQL database in it Geeks-forgeeks (2021). The **primary purpose** of MongoDB is to store and manage large amounts of data in an easy and effective approach.



Figure 2.3: Logo of MongoDB Application for understanding

The **key feature of MongoDB** is the usage of the **document-oriented data model**. It means that MongoDB uses a special methodology like JSON-like document format including effective schemas.

2.3.1 Characteristics of MongoDB

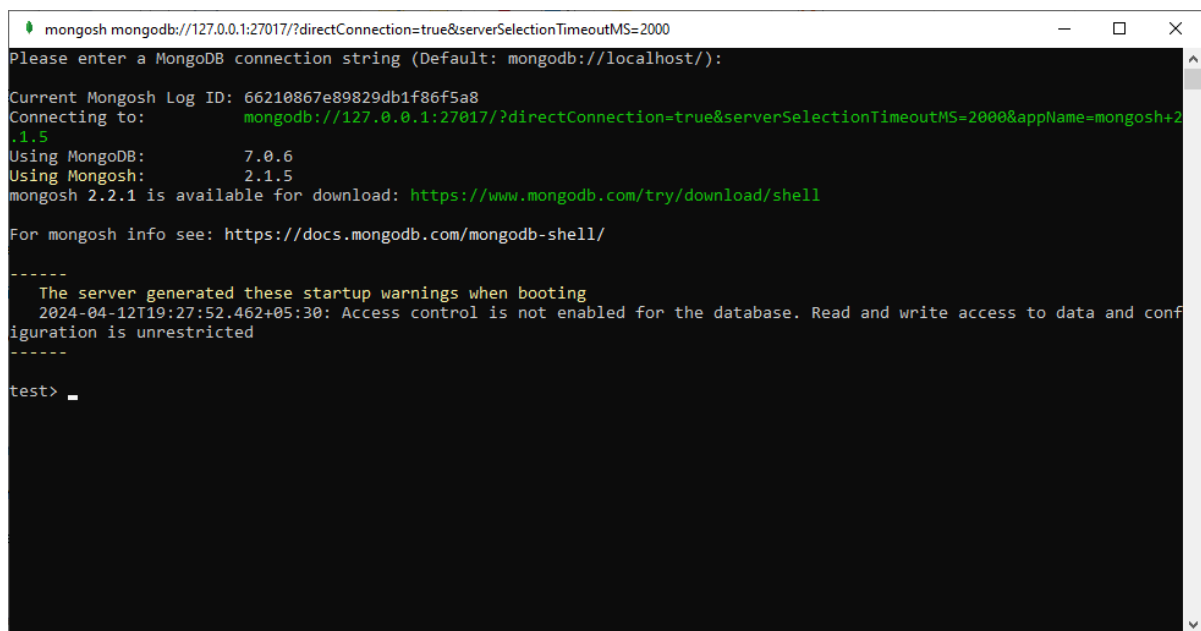
The following are the **key characteristics of MongoDB**:

- **Uses document-oriented approach:** MongoDB uses JSON-like documents called BSON (Binary JSON) to store the documents and the structure of documents may vary based on the structure.
- **Does not need a predefined schema:** MongoDB is schemaless. Hence, it doesn't use any predefined schema to store.
- **Supports high availability:** MongoDB provides high availability and horizontal scalability. Thus, it can effectively automatically replicate data in many servers.

- **Supports horizontal scalability:** MongoDB horizontally scales data and distributes data across multiple servers effectively.
- **Indexing:** MongoDB uses many types of indexes such as compound indexes, geospatial indexes, and text indexes to effectively retrieve data.
- **Uses Ad Hoc queries:** MongoDB uses many types of ad hoc query approaches to perform operations such as filtering, sorting, and projection.

2.4 What is MongoDB Shell?

The MongoDB Shell is a **command-line interface (CLI) tool** which is supported and provided by MongoDB. Using MongoDB Shell you can effectively interact with MongoDB databases. Thus, MongoDB Shell is used through a JavaScript-based interface which is very flexible and effective in usage.



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Please enter a MongoDB connection string (Default: mongodb://localhost/):
Current Mongosh Log ID: 66210867e89829db1f86f5a8
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.5
Using MongoDB:      7.0.6
Using Mongosh:      2.1.5
mongosh 2.2.1 is available for download: https://www.mongodb.com/try/download/shell
For mongosh info see: https://docs.mongodb.com/mongodb-shell/

-----
The server generated these startup warnings when booting
2024-04-12T19:27:52.462+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----

test> _
```

Figure 2.4: The MongoDB Shell is a command-line interface (CLI) tool

Hence, MongoDB Shell helps to **use a wide range of operations** such as querying,

inserting, updating, deleting, and managing databases, collections, indexes, and users.

2.4.1 The Applications of MongoDB Shell

The following are the key applications of MongoDB Shell:

- **Ability to effectively handle database administration:** MongoDB Shell helps to handle many database administration tasks such as creating and dropping databases, managing collections, creating and deleting indexes, and more.
- **Handle data querying and manipulation:** The users can use MongoDB Shell to query and handle data for finding documents, updating fields, deleting documents, and running pipelines.
- **Use database configuration:** MongoDB Shell helps to easily configure storage options, replication settings, sharding, and authentication settings.
- **Import and export data:** MongoDB Shell helps to import and export data effectively through MongoDB databases.
- **Testing and debugging purposes:** MongoDB Shell helps to test and debug MongoDB queries and scripts to identify and fix issues.

2.4.2 Strengths and Limitations of Mongo Shell

Table 2.2: Strengths and Weakness of Mongo Shell

Advantages of Mongo Shell	Disadvantages of Mongo Shell
<ul style="list-style-type: none"> • Helps to maintain direct interaction with MongoDB databases. • Ability to easily do tasks such as execute commands, queries, and handle data without the necessity of additional software. • Provides flexibility in scripting and automates tasks using JavaScript within MongoDB Shell. • Supports an easy learning curve for new users to use the command-line interface. • Ability to see results based on real-time data. • Supports a wide range of portability across many environments. 	<ul style="list-style-type: none"> • Since it works in a command-line interface there are inabilities to make visualizations. • Since the syntax is JavaScript-based, individuals who are not familiar with it may face issues. • MongoDB Shell provides direct access grants to the users exhibiting security risks. • There are scripting limitations as may face conflicts to perform advanced scripting tasks. • Requires proper maintenance when used for a large environment with many data and users.

2.5 What is MongoDB Compass?

MongoDB Compass is the **official graphical user interface (GUI)** provided for MongoDB. Geeksforgeeks (2024). Since the MongoDB Shell is not able to provide a graphical picture of the data, the MongoDB Compass fulfils this gap.

MongoDB Compass helps users to visually and manually interact with MongoDB databases, collections, and documents.

2.5.1 Characteristics of MongoDB Compass

The following are the key characteristics of MongoDB Compass:

- **Provides an effective GUI Interface:** MongoDB Compass's GUI interface helps to analyse the data, query, and handle MongoDB data.
- **Supports visual data exploration:** If you use MongoDB Compass, it helps to visually view the documents, collections, and databases.
- **Has a query builder tool:** The MongoDB Compass has a query builder tool to handle even complex queries without the MongoDB's query language syntax usage.
- **Supports index management:** The Compass supports the handling of many indexes, to create, modify, and delete indexes on MongoDB collections.
- **Provide real-time performance metrics:** The Compass, provides real-time performance metrics such as query execution times, index usage, and server status.

2.5.2 Strengths and Limitations of MongoDB Compass

Table 2.3: Advantages and disadvantages of MongoDB Compass

Strengths of MongoDB Compass	Limitations of MongoDB Compass
<ul style="list-style-type: none"> • Has a user-friendly interface to handle data effectively. • Supports visual data analysis based on the MongoDB data structures for data analysis. • Has an effective query-building tool to easily write and execute MongoDB queries. • Supports effective index management to optimize query performance. • Has a performance metrics feature to monitor MongoDB server performance in real-time. 	<ul style="list-style-type: none"> • Considered to be resource-intensive when you work with large datasets or advanced queries. • Has limited feature sets as some advanced users find it lacks certain features compared to other tools. • Certain MongoDB Compass features are dependent on specific MongoDB server versions. • This may give rise to security concerns such as authenticating users with MongoDB servers. • Certain advanced MongoDB Compass features require a paid subscription.

Chapter 3

Creating Data Engineering Pipelines using MongoDB To Solve Macy Store's Problem

In this section, the MongoDB shell would be used to create the NOSQL Application to solve the problem context of Macy Store. Hence, the data engineering process would start from database design, adding collections, analyzing the collections based on different functions and finding answers to key questions.

3.1 Database Design for Macy's Business

To solve the problem of maintaining optimal inventory levels in Macy's stores, the database design **consists of four collections** as follows:

- **Product Catalog Collection:** Includes the details of 100 products such as product identification code, name, description, product category, price, supplier information, and more details.
- **Store Inventory Collection:** Includes the store inventory details such as store identification code, store location, available inventory levels, and more details.

- **Sales History Collection:** Includes the sales-related details such as transaction identification code, sales product details, and the date and time of the transaction.
- **Customer Preferences Collection:** Includes customers related to the stores such as customer identification code, product details of interest, and demographic information of the customer.

The Figure 3.1 depicts the database design established for Macy's business to solve the problem context:

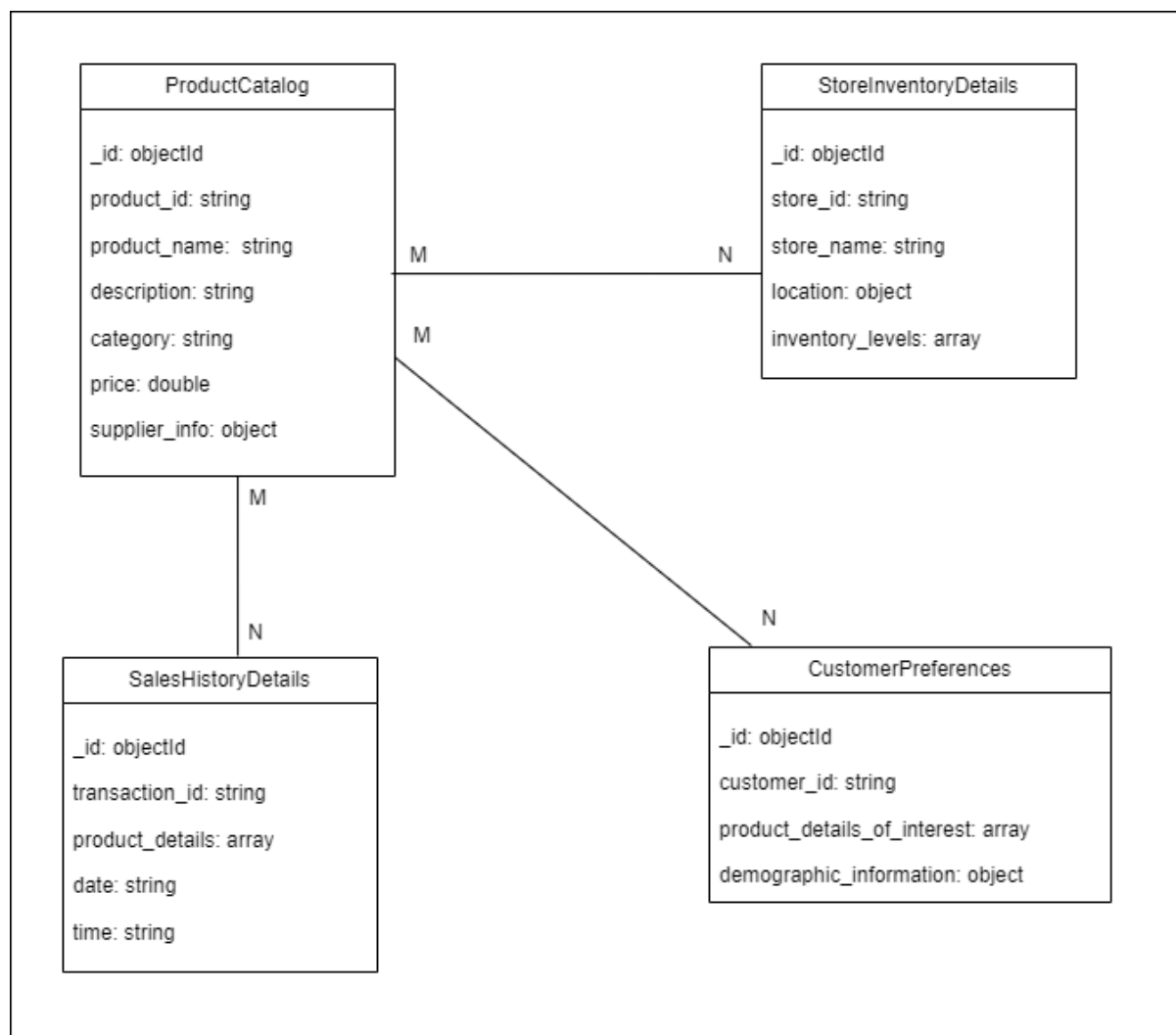


Figure 3.1: Database Design for Macy's Business

3.2 Mongoosh Shell Commands Used for Macy's Business

To create the NOSQL application using MongoDB, a database needs to be created including the collections and based on the database design discussed above. To follow this step, mongoosh shell will be used to create the database, add collections and perform the proceeding activities.

3.2.1 How to create a database in MongoDB?

The command line prompt of the MongoDB shell is the “Mongosh”. All the command line arguments or codes are entered in Mongosh which connects with the localhost of MongoDB.

To start Mongosh, navigate to the location where the mongosh.exe file is located as shown in Figure 3.2. In the student's computer, it is located in the “E:\Programs\MongoDB\mongosh-2.1.5-win32-x64\mongosh-2.1.5-win32-x64\bin.”

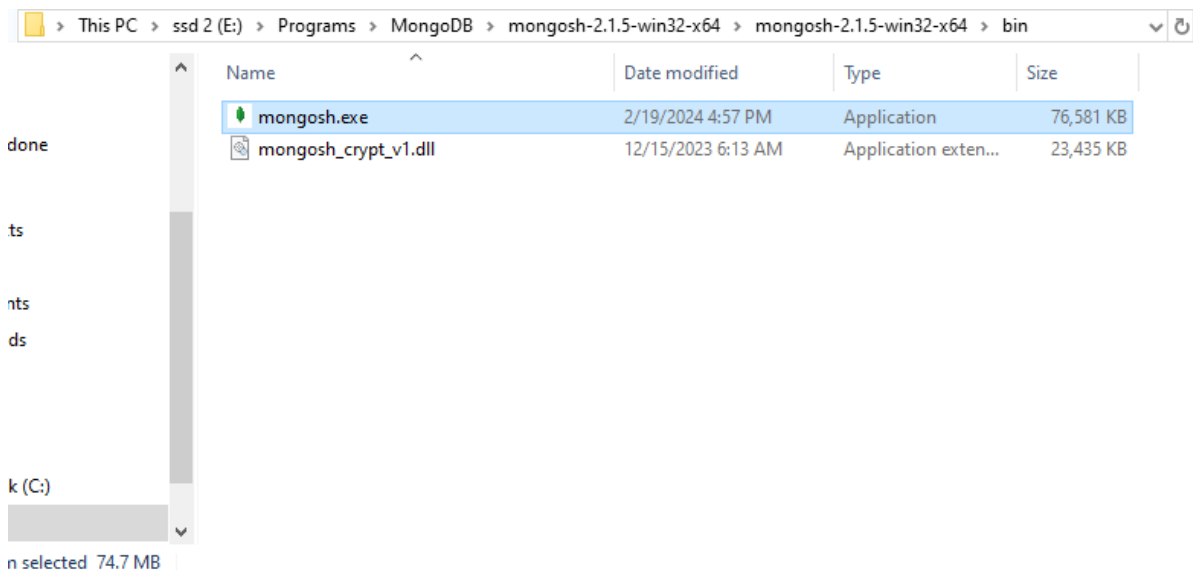
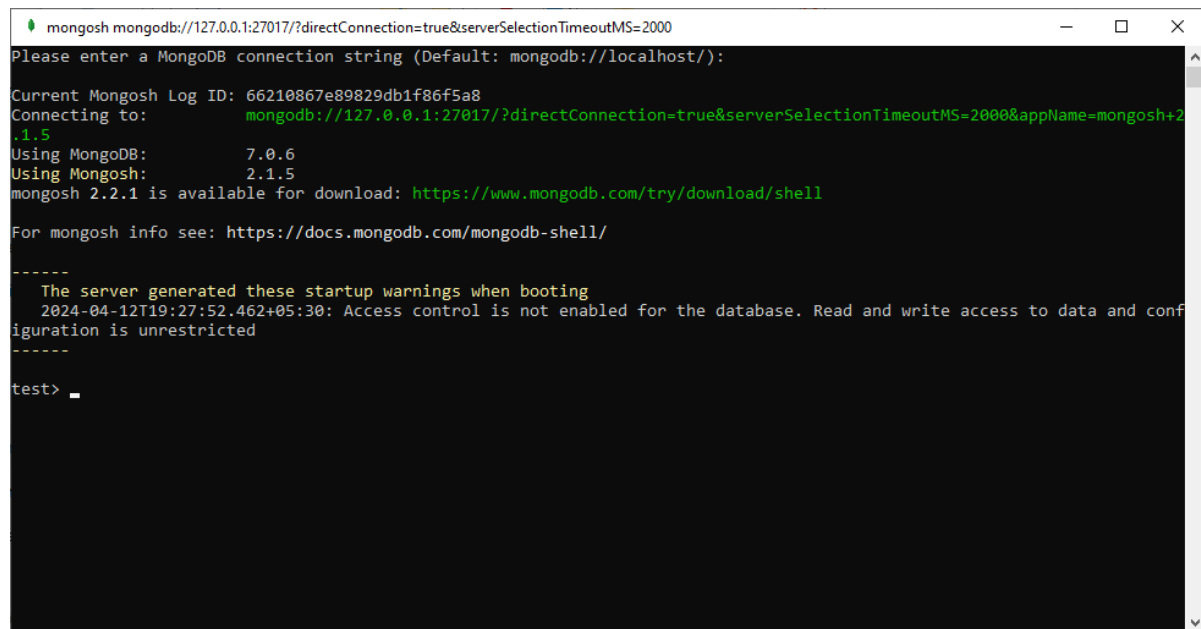


Figure 3.2: Navigation to the location where the mongosh.exe file is located

After opening the ”moongosh.exe file”, it automatically connects with the default MongoDB localhost as shown in the Figure 3.3.



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Please enter a MongoDB connection string (Default: mongodb://localhost/):
Current Mongosh Log ID: 66210867e89829db1f86f5a8
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.5
Using MongoDB:      7.0.6
Using Mongosh:       2.1.5
mongosh 2.2.1 is available for download: https://www.mongodb.com/try/download/shell
For mongosh info see: https://docs.mongodb.com/mongodb-shell/

-----
  The server generated these startup warnings when booting
  2024-04-12T19:27:52.462+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----
test> _
```

Figure 3.3: MongoDB CLI depicting the connection to localhost

3.2.2 To show all the existing database

Before beginning to create a Macy's database, the developer **should check the existing databases**. It helps to check the list of available databases. Also, if there are any identical names for the database name the developer has planned to create, it helps to forecast the issue in advance.

To display all the existing databases, the following command is used as shown in Figure 3.5:

- **Command:** `show dbs`
- **Result:** Displays all the existing databases in the connected localhost server.

The Figure 3.5 displays the usage of the above-mentioned command and the expected result of using it:

```
test> show dbs
AripicoRetail  216.00 KiB
CabDriver      104.00 KiB
MacyStore      348.00 KiB
Restuarants    1.22 MiB
admin          40.00 KiB
apiitbatch     192.00 KiB
config         60.00 KiB
local          72.00 KiB
staffsStudents 148.00 KiB
test           4.07 MiB
test>
```

Figure 3.4: Displays all the existing databases in the connected localhost server

3.2.3 To show the currently working database

Suppose while working in Mongo Compass and if it **lose track of the currently working database**, the below command can be enetered to check it.

- **Command:** db
- **Result:** Displays the currently working database connected to the localhost server.

The Figure 3.5 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
test> db
test
test> _
```

Figure 3.5: Displays the currently working database connected to the localhost server

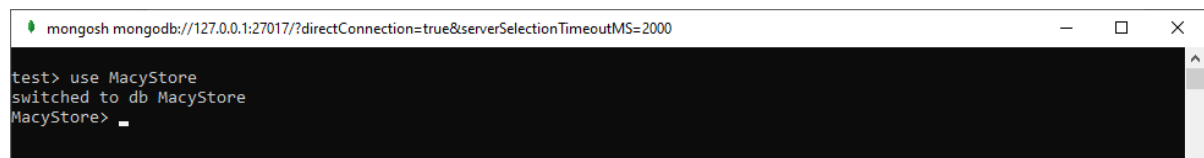
3.2.4 Usage of 'use' command or ways to create a database

To **start a new database for Macy's store**, first it is important create a database. Enter the below command for this purpose:

- **Command:** use MacyStore

- **Result:** Creates a new database called “MacyStore” if the given name is not available and automatically switches to the newly created database. Also, if the name already exists or if the database is already created, using the command `use MacyStore` will automatically switch to the specified database as the currently working database.

The Figure 3.6 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
test> use MacyStore
switched to db MacyStore
MacyStore> _
```

Figure 3.6: Creates a new database called “MacyStore”

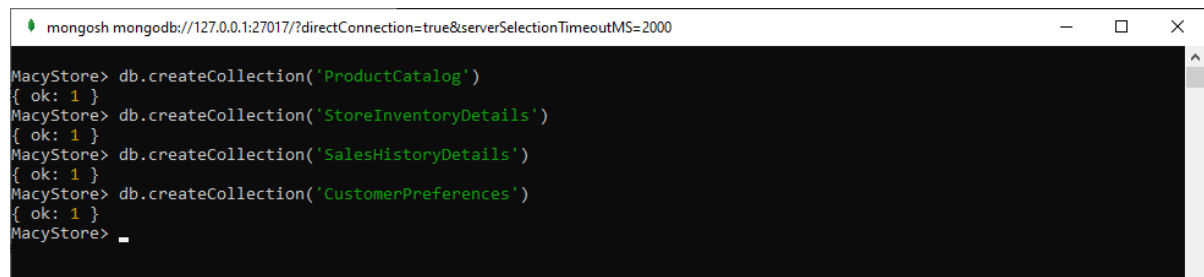
3.2.5 To create a collection

What are collections? In MongoDB, a collection is a group of MongoDB documents. For example, in a relational database, it is a table and in MongoDB, we call it a collection. However, collections in MongoDB don't have a specific schema and have varied structures.

As discussed previously, four collections are essential for Macy's store. The following is the command used to **create a collection** in MongoDB in the Mongosh shell:

- **Command:** `db.createCollection('ProductCatalog')`
- **Result:** Creates a new collection called 'ProductCatalog' inside the 'MacyStore' database.

The Figure 3.7 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.createCollection('ProductCatalog')
{ ok: 1 }
MacyStore> db.createCollection('StoreInventoryDetails')
{ ok: 1 }
MacyStore> db.createCollection('SalesHistoryDetails')
{ ok: 1 }
MacyStore> db.createCollection('CustomerPreferences')
{ ok: 1 }
MacyStore> _
```

Figure 3.7: Creates a new collection called 'ProductCatalog'

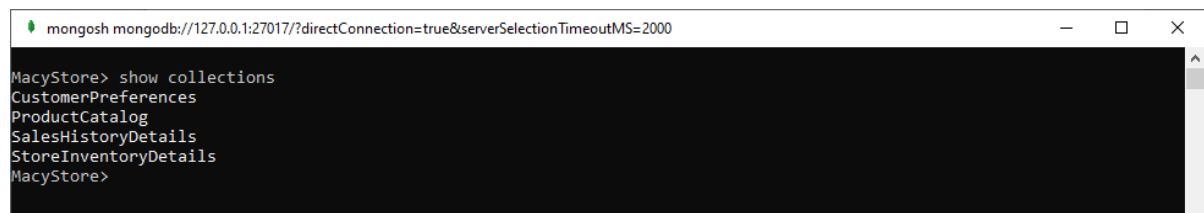
3.2.6 To Show all collections inside the working database

After creating collections or while understanding the database, it is important to **know the available collections inside MacyStore database**.

The following is the command used to view all the created collections inside the working database in MongoDB in the Mongosh shell:

- **Command:** `show collections`
- **Result:** Displays all the collections created or existing inside the 'MacyStore' or currently working database in a list format.

The Figure 3.8 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> show collections
CustomerPreferences
ProductCatalog
SalesHistoryDetails
StoreInventoryDetails
MacyStore>
```

Figure 3.8: Displays all the collections existing inside the 'MacyStore'

3.2.7 To insert one or more documents into the collection

What are documents? In MongoDB, a document is the basic unit of data storage. For example, in a relational database, it is a row and in MongoDB, it is called a document. However, documents have JSON-like data structures but they are not specifically in JSON format. They primarily have field-value pairs and documents are stored inside the collections.

In Mongosh, you can **insert documents one by one** using the `insertOne()` command or insert documents as a huge lot consisting of more than one document at a time using the `insertMany()` command.

The following is the command used to insert only one document into MacyStore database inside the collection in MongoDB in the Mongosh shell:

Command:

```
{ "product_id": "P001",  
  "product_name": "Men's Casual Shirt",  
  "description": "Soft cotton shirt for everyday wear",  
  "category": "Apparel",  
  "price": 29.99,  
  "supplier_info":  
  { "supplier_id": "S001",  
    "supplier_name": "Fashion Suppliers Inc.",  
    "contact_info": {  
      "email": "contact@fashionsuppliers.com",  
      "phone": "+1234567890"  
    } }  
}
```

Result: Inserts one document into the “ProductCatalog” collection which exists inside the ‘MacyStore’ or currently working database. If the insertion is successful, display the results in green color that the document is inserted successfully as shown in Figure 3.9.



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.ProductCatalog.insertOne({
...   "product_id": "P001",
...   "product_name": "Men's Casual Shirt",
...   "description": "Soft cotton shirt for everyday wear",
...   "category": "Apparel",
...   "price": 29.99,
...   "supplier_info": {
...     "supplier_id": "S001",
...     "supplier_name": "Fashion Suppliers Inc.",
...     "contact_info": {
...       "email": "contact@fashionsuppliers.com",
...       "phone": "+1234567890"
...     }
...   },
...   "historical_sales_data": [
...     { "date": "2023-01-01", "sales_volume": 120, "revenue": 3597.60 },
...     { "date": "2023-01-02", "sales_volume": 90, "revenue": 2699.10 }
...   ]
... })
{ acknowledged: true,
  insertedId: ObjectId('6621125aa4ffeb94b3cf3554') }
MacyStore> _
```

Figure 3.9: Insert one document into the “ProductCatalog” collection

The following is the command used to **insert more than one document at a time** inside the collection in MongoDB in the Mongosh shell:

Command:

```
db.ProductCatalog.insertMany([{"product_id": "P002", "product_name": "  
"Women'sSkinnyJeans",  
"description": "Stretchdenimjeansforacomfortablefit",  
"category": "Apparel", "price": 39.99,  
"supplier_info": {"supplier_id": "S002", "supplier_name": "DenimDelights",  
"contact_info": {"email": "contact@denimdelights.com", "phone": " + 1234567890"}},  
{"product_id": "P003", "product_name": "Men'sLeatherBelt",  
"description": "Genuineleatherbeltforaclassiclook", "category": "Apparel", "price": 24.99,  
"supplier_info": {"supplier_id": "S003", "supplier_name": "LeatherWorks",  
"contact_info": {"email": "contact@leatherworks.com", "phone": " + 1234567890"}},  
{"product_id": "P004", "product_name": "Women'sFloralDress",  
"description": "Elegantfloraldressforspecialoccasions", "category": "Apparel",  
"price": 49.99,  
"supplier_info": {"supplier_id": "S004", "supplier_name": "FashionTrendsLtd.",  
"contact_info": {"email": "contact@fashiontrends.com", "phone": " + 1234567890"} } ]])
```

Result: Inserts three documents into the “ProductCatalog” collection which exists inside the ‘MacyStore’ or currently working database. If the insertion is successful, display the results in green color that the three documents are inserted successfully.

The Figure 3.10 displays the usage of the above-mentioned command and the expected result of using the specified command in the ‘MacyStore’ collection:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.ProductCatalog.insertMany([
...   {
...     "product_id": "P002",
...     "product_name": "Women's Skinny Jeans",
...     "description": "Stretch denim jeans for a comfortable fit",
...     "category": "Apparel",
...     "price": 39.99,
...     "supplier_info": {
...       "supplier_id": "S002",
...       "supplier_name": "Denim Delights",
...       "contact_info": {
...         "email": "contact@denimdelights.com",
...         "phone": "+1234567890"
...       }
...     },
...     "historical_sales_data": [
...       { "date": "2023-01-01", "sales_volume": 150, "revenue": 5998.50 },
...       { "date": "2023-01-02", "sales_volume": 110, "revenue": 4399.90 }
...     ]
...   },
...   {
...     "product_id": "P003",
...     "product_name": "Men's Leather Belt",
...     "description": "Genuine leather belt for a classic look",
...     "category": "Apparel",
...     "price": 24.99,
...     "supplier_info": {
...       "supplier_id": "S003",
...       "supplier_name": "Leather Works",
...       "contact_info": {
...         "email": "contact@leatherworks.com",
...         "phone": "+1234567890"
...       }
...     },
...     "historical_sales_data": [
...       { "date": "2023-01-01", "sales_volume": 80, "revenue": 1999.20 },
...       { "date": "2023-01-02", "sales_volume": 70, "revenue": 1749.30 }
...     ]
...   },
...   {
...     "product_id": "P004",
...     "product_name": "Women's Floral Dress",
...     "description": "Elegant floral dress for special occasions",
...     "category": "Apparel",
...     "price": 49.99,
...   }
... ])
```

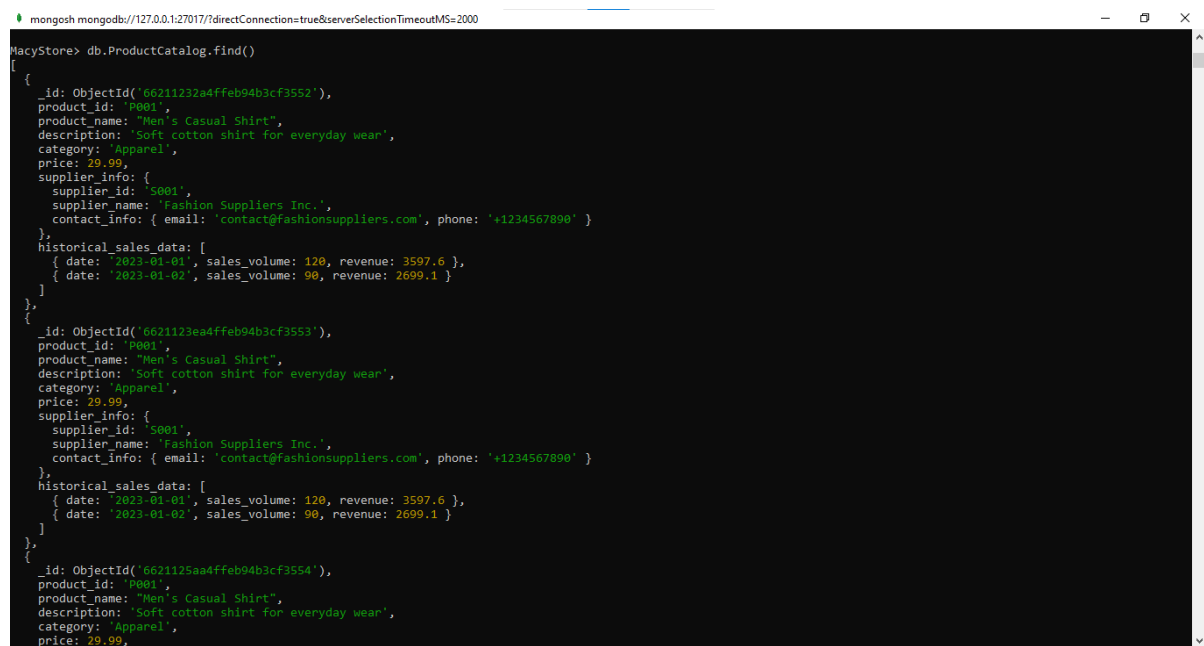
Figure 3.10: Insert three documents into the “ProductCatalog” collection

3.2.8 To view all documents

To double-check whether all the documents are inserted or to **view all documents inside a ProductCatalog collection**, the following command is used:

- **Command:** `db.ProductCatalog.find()`
- **Result:** Displays all the existing documents inserted inside the `ProductCatalog` collection located inside the `MacyStore` database.

The Figure 3.11 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.ProductCatalog.find()
[
  {
    _id: ObjectId('60211232a4ffeb94b3cf3552'),
    product_id: 'P001',
    product_name: 'Men's Casual Shirt',
    description: 'Soft cotton shirt for everyday wear',
    category: 'Apparel',
    price: 29.99,
    supplier_info: {
      supplier_id: 'S001',
      supplier_name: 'Fashion Suppliers Inc.',
      contact_info: { email: 'contact@fashionsuppliers.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 120, revenue: 3597.6 },
      { date: '2023-01-02', sales_volume: 90, revenue: 2699.1 }
    ]
  },
  {
    _id: ObjectId('6021123ea4ffeb94b3cf3553'),
    product_id: 'P001',
    product_name: 'Men's Casual Shirt',
    description: 'Soft cotton shirt for everyday wear',
    category: 'Apparel',
    price: 29.99,
    supplier_info: {
      supplier_id: 'S001',
      supplier_name: 'Fashion Suppliers Inc.',
      contact_info: { email: 'contact@fashionsuppliers.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 120, revenue: 3597.6 },
      { date: '2023-01-02', sales_volume: 90, revenue: 2699.1 }
    ]
  },
  {
    _id: ObjectId('6021125aa4ffeb94b3cf3554'),
    product_id: 'P001',
    product_name: 'Men's Casual Shirt',
    description: 'Soft cotton shirt for everyday wear',
    category: 'Apparel',
    price: 29.99,
    supplier_info: {
      supplier_id: 'S001',
      supplier_name: 'Fashion Suppliers Inc.',
      contact_info: { email: 'contact@fashionsuppliers.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 120, revenue: 3597.6 },
      { date: '2023-01-02', sales_volume: 90, revenue: 2699.1 }
    ]
  }
]
```

Figure 3.11: Displays all the existing documents inserted inside the `ProductCatalog` collection

3.2.9 To view all the formatted documents

To display or view all the documents inside the `ProductCatalog` collection in a **JSON-like format** in a pretty or enhanced manner, the following command is used:

- **Command:** `db.ProductCatalog.find().pretty()`

- **Result:** Displays all the existing documents inserted inside the ProductCatalog collection in a JSON-like format with an enhanced structure located inside the MacyStore database.

The Figure 3.12 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```
MacyStore> db.ProductCatalog.find().pretty()
{
  {
    _id: ObjectId('6021125aa4ffeb94b3cf3554'),
    product_id: 'P001',
    product_name: 'Men's Casual Shirt',
    description: 'Soft cotton shirt for everyday wear',
    category: 'Apparel',
    price: 29.99,
    supplier_info: {
      supplier_id: 'S001',
      supplier_name: 'Fashion Suppliers Inc.',
      contact_info: { email: 'contact@fashionsuppliers.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 120, revenue: 3597.6 },
      { date: '2023-01-02', sales_volume: 90, revenue: 2699.1 }
    ]
  },
  {
    _id: ObjectId('6021138da4ffeb94b3cf3555'),
    product_id: 'P002',
    product_name: 'Women's Skinny Jeans',
    description: 'Stretch denim jeans for a comfortable fit',
    category: 'Apparel',
    price: 39.99,
    supplier_info: {
      supplier_id: 'S002',
      supplier_name: 'Denim Delights',
      contact_info: { email: 'contact@denimdelights.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 150, revenue: 5998.5 },
      { date: '2023-01-02', sales_volume: 110, revenue: 4399.9 }
    ]
  },
  {
    _id: ObjectId('6021138da4ffeb94b3cf3556'),
    product_id: 'P003',
    product_name: 'Men's Leather Belt',
    description: 'Genuine leather belt for a classic look',
    category: 'Apparel',
  }
}
```

Figure 3.12: Displays all the existing documents inserted inside the ProductCatalog collection in a JSON-like format

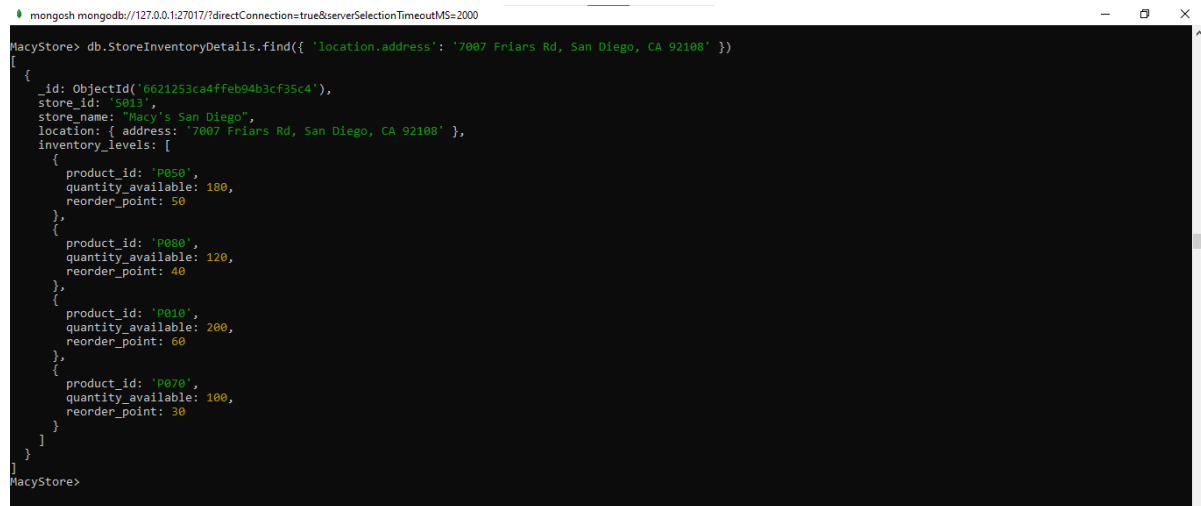
3.2.10 To Find specific documents

To find specific documents with special characteristics, find() command is used. However, to find specific documents with special characteristics, the field name must be given. The following is a command used to find documents with a specific field value inside the StoreInventoryDetails for analysis:

- **Command:** `db.StoreInventoryDetails.find({ 'location.address': '7007 Friars Rd, San Diego, CA 92108' })`

- **Result:** Finds and outputs the documents where the `location.address` field matches inside the `StoreInventoryDetails` collection having the specified address. In this use case, the result displays only one document which has this characteristic.

The Figure 3.13 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.StoreInventoryDetails.find({ 'location.address': '7007 Friars Rd, San Diego, CA 92108' })
[
  {
    _id: ObjectId('6621253ca4ffeb94b3cf35c4'),
    store_id: '5013',
    store_name: 'Macy's San Diego',
    location: { address: '7007 Friars Rd, San Diego, CA 92108' },
    inventory_levels: [
      {
        product_id: 'P0050',
        quantity_available: 100,
        reorder_point: 50
      },
      {
        product_id: 'P0080',
        quantity_available: 120,
        reorder_point: 40
      },
      {
        product_id: 'P010',
        quantity_available: 200,
        reorder_point: 60
      },
      {
        product_id: 'P070',
        quantity_available: 100,
        reorder_point: 30
      }
    ]
  }
]
```

Figure 3.13: Finds and outputs the documents where the `location.address` field

Another instance while doing analysis is, if there are “ ” or **special characters in specific field values** that must be searched, adding a “/” before the special character helps to find the correct result. The following is an example of a command used to find documents with a specific field value including a special character:

- **Command:** `db.StoreInventoryDetails.find({'store_name':'Macy's Seattle'})`
- **Result:** Finds and outputs the documents where the `store_name` field matches inside the `StoreInventoryDetails` collection having the specified name. In this use case, the result displays only one document which has this characteristic.

The Figure 3.14 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.StoreInventoryDetails.find({'store_name':'Macy's Seattle'})
[
  {
    _id: ObjectId('6621253ca4ffeb94b3cf35c8'),
    store_id: '5009',
    store_name: 'Macy's Seattle',
    location: { address: '1601 3rd Ave, Seattle, WA 98181' },
    inventory_levels: [
      {
        product_id: 'P030',
        quantity_available: 140,
        reorder_point: 50
      },
      {
        product_id: 'P060', quantity_available: 80, reorder_point: 30
      },
      {
        product_id: 'P090',
        quantity_available: 160,
        reorder_point: 40
      },
      {
        product_id: 'P070',
        quantity_available: 100,
        reorder_point: 40
      }
    ]
  }
]
MacyStore>
```

Figure 3.14: Finds and outputs the documents where the store_name field matches

3.3 To Analyse Whether The Database Is Functioning Properly

To analyse whether the database is functioning properly and identify issues in collections, the below methods and commands were used. Hence, it was able to identify loopholes in the collections through further analysis.

3.3.1 To Sort documents inside the specific collection

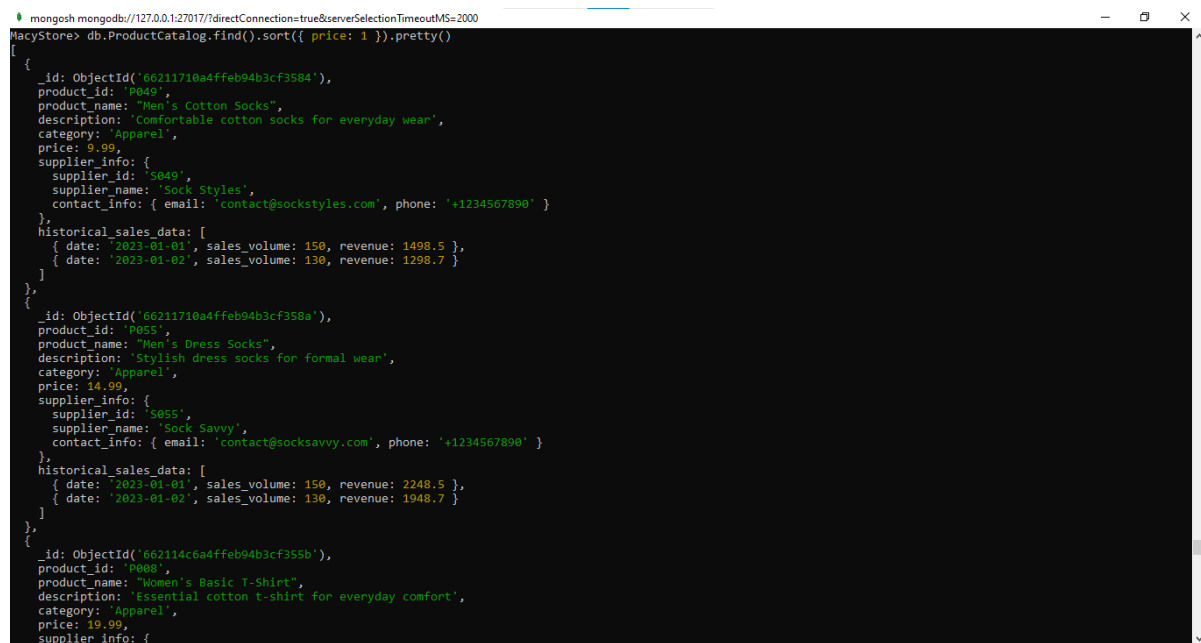
Documents inside the collection can be **sorted in both ascending and descending order** based on the specific field. In order to **understand the ProductCatalog collection better**, this step is used. Generally, if “1” is placed before the field name, based on the field name the values are sorted in ascending order.

Sort Documents In Ascending Order Inside The Specific Collection

Generally, if “1” is placed before the field name, based on the field name the values are **sorted in ascending order**. Use the below command to sort documents in ascending order by using a specific field:

- **Command:** `db.ProductCatalog.find().sort({ price: 1 }).pretty()`
- **Result:** Outputs the list of documents where the price field is sorted in ascending order after analyzing all the documents inside the ProductCatalog collection.

The Figure 3.15 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.ProductCatalog.find().sort({ price: 1 }).pretty()
{
  _id: ObjectId('66211710a4ffeb94b3cf3584'),
  product_id: 'P049',
  product_name: 'Men's Cotton Socks',
  description: 'Comfortable cotton socks for everyday wear',
  category: 'Apparel',
  price: 9.99,
  supplier_info: {
    supplier_id: 'S049',
    supplier_name: 'Sock Styles',
    contact_info: { email: 'contact@sockstyles.com', phone: '+1234567890' }
  },
  historical_sales_data: [
    { date: '2023-01-01', sales_volume: 150, revenue: 1498.5 },
    { date: '2023-01-02', sales_volume: 130, revenue: 1298.7 }
  ]
},
{
  _id: ObjectId('66211710a4ffeb94b3cf358a'),
  product_id: 'P055',
  product_name: 'Men's Dress Socks',
  description: 'Stylish dress socks for formal wear',
  category: 'Apparel',
  price: 14.99,
  supplier_info: {
    supplier_id: 'S055',
    supplier_name: 'Sock Savvy',
    contact_info: { email: 'contact@socksavvy.com', phone: '+1234567890' }
  },
  historical_sales_data: [
    { date: '2023-01-01', sales_volume: 150, revenue: 2248.5 },
    { date: '2023-01-02', sales_volume: 130, revenue: 1948.7 }
  ]
},
{
  _id: ObjectId('662114c6a4ffeb94b3cf355b'),
  product_id: 'P008',
  product_name: 'Women's Basic T-Shirt',
  description: 'Essential cotton t-shirt for everyday comfort',
  category: 'Apparel',
  price: 19.99,
  supplier_info: {
```

Figure 3.15: List of documents where the price field is sorted in ascending order

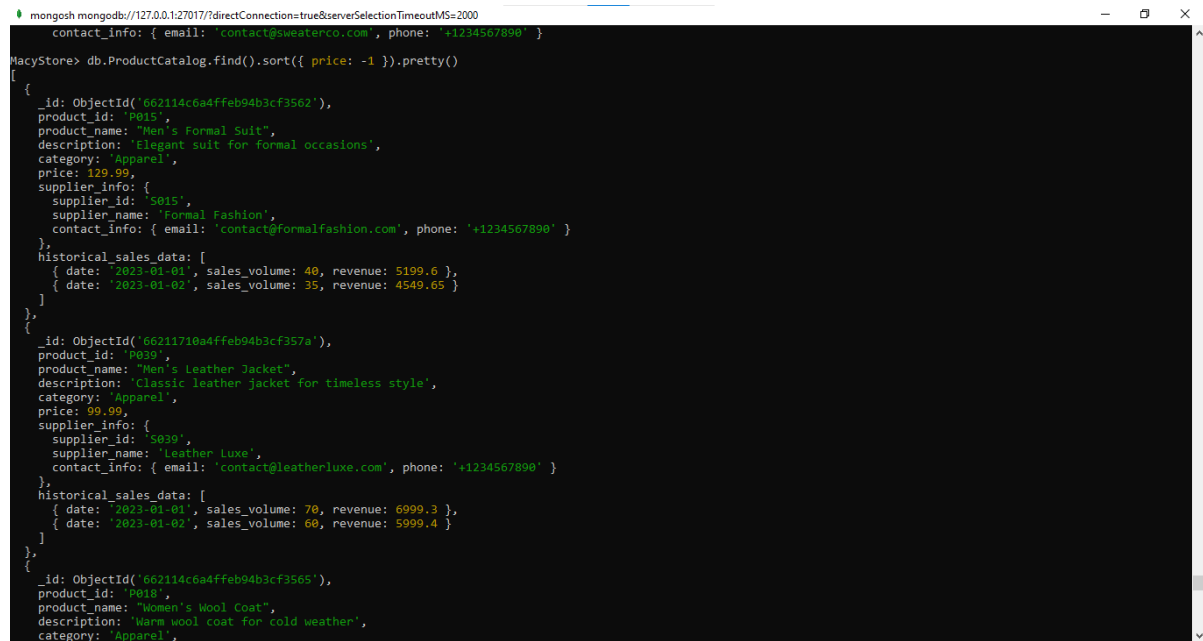
To Sort Documents In Descending Order Inside The Specific Collection

Generally, if “-1” is placed before the field name, based on the field name the values are **sorted in descending order**. Use the below command to sort documents in descending order by using a specific field in a pretty format:

- **Command:** `db.ProductCatalog.find().sort({ price: -1 }).pretty()`
- **Result:** Outputs the list of documents where the price field is sorted in descending order after analyzing all the documents inside the ProductCatalog collection in a JSON-like

format.

The Figure 3.16 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
contact_info: { email: 'contact@sweaterco.com', phone: '+1234567890' }
MacyStore> db.ProductCatalog.find().sort({ price: -1 }).pretty()
[
  {
    _id: ObjectId('662114c6a4ffeb94b3cf3562'),
    product_id: 'P015',
    product_name: 'Men's Formal Suit',
    description: 'Elegant suit for formal occasions',
    category: 'Apparel',
    price: 129.99,
    supplier_info: {
      supplier_id: 'S015',
      supplier_name: 'Formal Fashion',
      contact_info: { email: 'contact@formalfashion.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 40, revenue: 5199.6 },
      { date: '2023-01-02', sales_volume: 35, revenue: 4549.65 }
    ]
  },
  {
    _id: ObjectId('66211710a4ffeb94b3cf357a'),
    product_id: 'P039',
    product_name: 'Men's Leather Jacket',
    description: 'Classic leather jacket for timeless style',
    category: 'Apparel',
    price: 99.99,
    supplier_info: {
      supplier_id: 'S039',
      supplier_name: 'Leather Luxe',
      contact_info: { email: 'contact@leatherluxe.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 70, revenue: 6999.3 },
      { date: '2023-01-02', sales_volume: 60, revenue: 5999.4 }
    ]
  },
  {
    _id: ObjectId('662114c6a4ffeb94b3cf3565'),
    product_id: 'P018',
    product_name: 'Women's Wool Coat',
    description: 'Warm wool coat for cold weather',
    category: 'Apparel',
  }
]
```

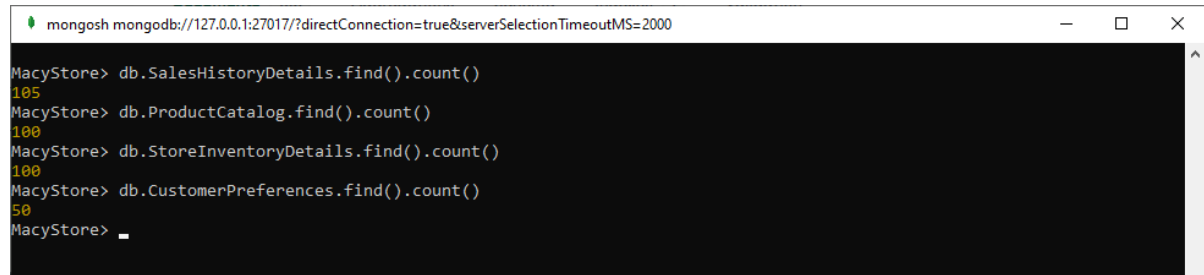
Figure 3.16: List of documents where the price field is sorted in descending order

3.3.2 To count the documents inside the collection

To count the documents inside the created collection, the function “find().count()” is used. The following is the command used to count all the documents inside the StoreInventoryDetails collection:

- **Command:** `db.StoreInventoryDetails.find().count()`
- **Result:** Outputs the number of documents inserted or available inside the StoreInventoryDetails collection. The figure displays the count of the documents inserted or available inside all the created collections such as SalesHistoryDetails, ProductCatalog, StoreInventoryDetails, and CustomerPreferences.

The Figure 3.17 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.SalesHistoryDetails.find().count()
105
MacyStore> db.ProductCatalog.find().count()
100
MacyStore> db.StoreInventoryDetails.find().count()
100
MacyStore> db.CustomerPreferences.find().count()
50
MacyStore> _
```

Figure 3.17: Displays the number of documents available inside the StoreInventoryDetails collection

3.3.3 To limit the number of documents to be displayed inside a specific collection

If the “find()” command is used, it will **display all the documents inside a specific collection**. Hence, you can use “find().limit(number)” to limit the number of documents displayed in the output. If “find().limit(2)” is used, it displays only 2 documents inside the specific collection. The following command is used to limit the number of documents to be displayed inside the StoreInventoryDetails collection:

- **Command:** `db.StoreInventoryDetails.find().limit(2).pretty()`
- **Result:** Displays only the first two documents stored inside the “StoreInventoryDetails” collection which is located inside the “MacyStore” database.

The Figure 3.18 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.StoreInventoryDetails.find().limit(2).pretty()
{
  _id: ObjectId('6621253ca4ffeb94b3cf35b8'),
  store_id: 'S001',
  store_name: "Macy's Herald Square",
  location: { address: '151 W 34th St, New York, NY 10001' },
  inventory_levels: [
    {
      product_id: 'P040',
      quantity_available: 150,
      reorder_point: 50
    },
    { product_id: 'P010', quantity_available: 80, reorder_point: 30 },
    {
      product_id: 'P080',
      quantity_available: 200,
      reorder_point: 60
    },
    {
      product_id: 'P025',
      quantity_available: 100,
      reorder_point: 40
    }
  ]
},
{
  _id: ObjectId('6621253ca4ffeb94b3cf35b9'),
  store_id: 'S002',
  store_name: "Macy's Chicago",
  location: { address: '111 N State St, Chicago, IL 60602' },
  inventory_levels: [
    {
      product_id: 'P005',
      quantity_available: 120,
      reorder_point: 40
    },
    { product_id: 'P035', quantity_available: 90, reorder_point: 30 },
    {
      product_id: 'P095',
      quantity_available: 180,
      reorder_point: 50
    }
  ]
}
```

Figure 3.18: Displays only the first two documents stored inside the “StoreInventoryDetails” collection

3.3.4 To sort and limit the number of documents displayed as output in a specific collection (Chaining)

As discussed above, documents inside the collection can be **sorted in both ascending and descending order based on the specific field**. Hence, the commands “sort()” and “limit()” can be used jointly to accomplish this function. This process is called “Chaining”


Sort Documents In Ascending Order And Limit The Documents Displayed Inside The Specific Collection

The following command is used to sort documents in ascending order and limit the output to display only 2 documents in ProductCatalog collection as outputs:

- **Command:** `db.ProductCatalog.find().limit(2).sort({ price: 1 }).pretty()`
- **Result:** Performs the sorting of the price field in ascending order, and outputs only 2

documents from the ProductCatalog collection in a pretty manner (JSON-like format).

The Figure 3.19 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.ProductCatalog.find().limit(2).sort({ price: 1 }).pretty()
[
  {
    _id: ObjectId('66211710a4ffeb94b3cf3584'),
    product_id: 'P049',
    product_name: 'Men's Cotton Socks',
    description: 'Comfortable cotton socks for everyday wear',
    category: 'Apparel',
    price: 9.99,
    supplier_info: {
      supplier_id: 'S049',
      supplier_name: 'Sock Styles',
      contact_info: { email: 'contact@sockstyles.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 150, revenue: 1498.5 },
      { date: '2023-01-02', sales_volume: 130, revenue: 1298.7 }
    ]
  },
  {
    _id: ObjectId('66211710a4ffeb94b3cf358a'),
    product_id: 'P055',
    product_name: 'Men's Dress Socks',
    description: 'Stylish dress socks for formal wear',
    category: 'Apparel',
    price: 14.99,
    supplier_info: {
      supplier_id: 'S055',
      supplier_name: 'Sock Savvy',
      contact_info: { email: 'contact@socksavvy.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 150, revenue: 2248.5 },
      { date: '2023-01-02', sales_volume: 130, revenue: 1948.7 }
    ]
  }
]
```

Figure 3.19: Sorting of price field in ascending order

Sort Documents In Descending Order And Limit The Documents Displayed Inside The Specific Collection

The following command is used to **sort documents in descending order and limit the output** to display only 2 documents as outputs in ProductCatalog collection:

- **Command:** `db.ProductCatalog.find().limit(2).sort({ price: -1 }).pretty()`
- **Result:** Performs the sorting of price field in descending order, and outputs only 2 documents from the ProductCatalog collection in a pretty manner (JSON-like format).

The Figure 3.20 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.ProductCatalog.find().limit(2).sort({ price: -1 }).pretty()
{
  _id: ObjectId('662114c6a4feb94b3cf3562'),
  product_id: 'P015',
  product_name: 'Men's Formal Suit',
  description: 'Elegant suit for formal occasions',
  category: 'Apparel',
  price: 129.99,
  supplier_info: {
    supplier_id: 'S015',
    supplier_name: 'Formal Fashion',
    contact_info: { email: 'contact@formalfashion.com', phone: '+1234567890' }
  },
  historical_sales_data: [
    { date: '2023-01-01', sales_volume: 40, revenue: 5199.6 },
    { date: '2023-01-02', sales_volume: 35, revenue: 4549.65 }
  ]
},
{
  _id: ObjectId('66211710a4feb94b3cf357a'),
  product_id: 'P039',
  product_name: 'Men's Leather Jacket',
  description: 'Classic leather jacket for timeless style',
  category: 'Apparel',
  price: 99.99,
  supplier_info: {
    supplier_id: 'S039',
    supplier_name: 'Leather Luxe',
    contact_info: { email: 'contact@leatherluxe.com', phone: '+1234567890' }
  },
  historical_sales_data: [
    { date: '2023-01-01', sales_volume: 70, revenue: 6999.3 },
    { date: '2023-01-02', sales_volume: 60, revenue: 5999.4 }
  ]
}
MacyStore>
```

Figure 3.20: Displays the sorting of price field in descending order

Usage of `forEach()` function

The `forEach()` function can be used to **iterate over each document** and perform a specific function. The following is an example of a command to use `forEach()` function in `StoreInventoryDetails` collection for analysis:

- **Command:** `db.StoreInventoryDetails.find().forEach(function(doc) {
 print("Store Name: " + doc.store_name)
})`
- **Results:** Outputs the list of `store_name` field along with the text “Store Name” preceding it in a simplified manner after iterating the function over each document and performing the defined `forEach()` function.

The Figure 3.21 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.StoreInventoryDetails.find().forEach(function (doc) { print("Store Name: " + doc.store_name); })
Store Name: Macy's Herald Square
Store Name: Macy's Chicago
Store Name: Macy's San Francisco
Store Name: Macy's Los Angeles
Store Name: Macy's Miami
Store Name: Macy's Houston
Store Name: Macy's Dallas
Store Name: Macy's Boston
Store Name: Macy's Seattle
Store Name: Macy's Philadelphia
Store Name: Macy's Atlanta
Store Name: Macy's Las Vegas
Store Name: Macy's San Diego
Store Name: Macy's Orlando
Store Name: Macy's Washington DC
Store Name: Macy's Minneapolis
Store Name: Macy's Phoenix
Store Name: Macy's Nashville
Store Name: Macy's Detroit
Store Name: Macy's Portland
Store Name: Macy's Tampa
Store Name: Macy's Charlotte
Store Name: Macy's Indianapolis
Store Name: Macy's New Orleans
Store Name: Macy's Pittsburgh
Store Name: Macy's Salt Lake City
Store Name: Macy's Cincinnati
Store Name: Macy's Austin
Store Name: Macy's San Antonio
Store Name: Macy's Sacramento
Store Name: Macy's Raleigh
Store Name: Macy's Las Vegas
Store Name: Macy's Denver
Store Name: Macy's Kansas City
Store Name: Macy's Oklahoma City
Store Name: Macy's Tucson
Store Name: Macy's Des Moines
Store Name: Macy's Anchorage
Store Name: Macy's Honolulu
Store Name: Macy's Raleigh
Store Name: Macy's Salt Lake City
Store Name: Macy's Atlanta
```

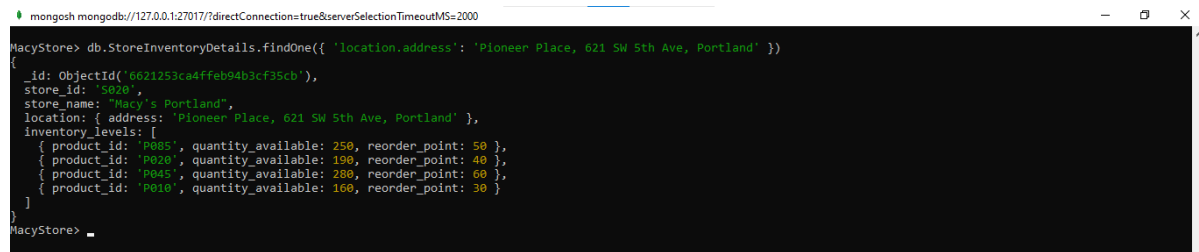
Figure 3.21: Displays the list of store_name field along with the text “Store Name”

3.3.5 To Find one document inside a specific collection

To find one document inside a specific collection, the “findOne()” command is used. The following is a command used to find one document in StoreInventoryDetails collection aligning with the described functionality:

- **Command:** `db.StoreInventoryDetails.findOne({ 'location.address': 'Pioneer Place, 621 SW 5th Ave, Portland' })`
- **Results:** Finds and returns the first document and outputs only one document which has the location address of “Pioneer Place, 621 SW 5th Ave, Portland” inside the “StoreInventoryDetails” collection.

The Figure 3.22 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.StoreInventoryDetails.findOne({ 'location.address': 'Pioneer Place, 621 SW 5th Ave, Portland' })
{
  _id: ObjectId('6621253ca4ffeb94b3cf35cb'),
  store_id: 'S020',
  store_name: 'Macy's Portland',
  location: { address: 'Pioneer Place, 621 SW 5th Ave, Portland' },
  inventory_levels: [
    { product_id: 'P085', quantity_available: 250, reorder_point: 50 },
    { product_id: 'P020', quantity_available: 190, reorder_point: 40 },
    { product_id: 'P045', quantity_available: 280, reorder_point: 60 },
    { product_id: 'P010', quantity_available: 160, reorder_point: 30 }
  ]
}
```

Figure 3.22: Displays the specific address specified in the command

3.3.6 To Find documents with specific fields

To display specific fields only, the `find()` command along with the field name can be used. The following is a command used to find and output documents with a specific field only in `StoreInventoryDetails` collection:

- **Command:** `db.StoreInventoryDetails.find({}, { store_name: 1, 'inventory_levels.product_id': 1 })`
- **Result:** Finds and outputs the documents including only the `store_name` and `inventory_levels.product_id` fields and values which are inside the `StoreInventoryDetails` collection.

The Figure 3.23 displays the usage of the above-mentioned command the expected result of using the specified command in the 'MacyStore' collection:


```
MacyStore> db.StoreInventoryDetails.find({}, { 'store_name': 1, 'inventory_levels.product_id': 1 })
[
  {
    _id: ObjectId('6621253ca4ffeb94b3cf35b8'),
    store_name: "Macy's Herald Square",
    inventory_levels: [
      { product_id: 'P040' },
      { product_id: 'P010' },
      { product_id: 'P080' },
      { product_id: 'P025' }
    ]
  },
  {
    _id: ObjectId('6621253ca4ffeb94b3cf35b9'),
    store_name: "Macy's Chicago",
    inventory_levels: [
      { product_id: 'P005' },
      { product_id: 'P035' },
      { product_id: 'P095' }
    ]
  },
  {
    _id: ObjectId('6621253ca4ffeb94b3cf35ba'),
    store_name: "Macy's San Francisco",
    inventory_levels: [
      { product_id: 'P025' },
      { product_id: 'P050' },
      { product_id: 'P090' },
      { product_id: 'P070' }
    ]
  },
  {
    _id: ObjectId('6621253ca4ffeb94b3cf35bb'),
    store_name: "Macy's Los Angeles",
    inventory_levels: [
      { product_id: 'P030' },
      { product_id: 'P080' },
      { product_id: 'P020' },
      { product_id: 'P060' }
    ]
  }
]
```

Figure 3.23: Displays the documents having the fields store_name and inventory_levels.product_id fields

3.3.7 Update one document

To **update one document** of a specific field, the **update()** command along with the '\$set' function is used. The following is a command used to update one specific field with a desired value in StoreInventoryDetails collection:

- **Command:** `db.StoreInventoryDetails.update({ store_id: 'S020' }, { $set: { 'location.address': 'Macy's Port Store' } })`
- **Result:** Updates the `location.address` field of the document consisting of the 'store_id' of 'S020' with the new value: 'Macy's Port Store'. The “ ” symbol is used in the command to signify the usage of “ ” in the value.

The Figure 3.24 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```
MacyStore> db.StoreInventoryDetails.update({ store_id: 'S020' }, { $set: { 'location.address': 'Macy's Port Store' } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
MacyStore> db.StoreInventoryDetails.find({"store_id": 'S020'})
[
  {
    _id: ObjectId('6621253ca4ffeb94b3cf35cb'),
    store_id: 'S020',
    store_name: 'Macy's Portland',
    location: { address: 'Macy's Port Store' },
    inventory_levels: [
      {
        product_id: 'P085',
        quantity_available: 250,
        reorder_point: 50
      },
      {
        product_id: 'P020',
        quantity_available: 190,
        reorder_point: 40
      },
      {
        product_id: 'P045',
        quantity_available: 280,
        reorder_point: 60
      },
      {
        product_id: 'P010',
        quantity_available: 160,
        reorder_point: 30
      }
    ]
  }
]
MacyStore>
```

Figure 3.24: Updates the document based on the specified command

After updating the above-mentioned field, to verify whether the update is performed, the following command was used:

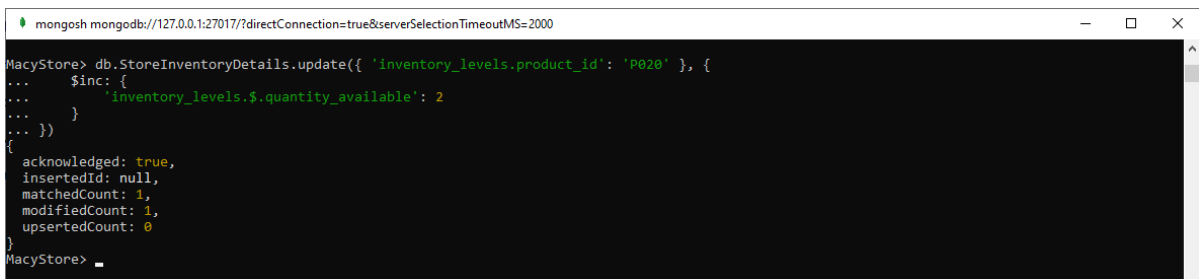
- **Command:** `db.StoreInventoryDetails.find({"store_id": 'S020'})`
- **Result:** The results of using the above-mentioned command display the document with the 'store_id' of 'S020'. After analyzing the document, it is noted that the update is successful as the changes were applied according to the predefined `update()` command.

3.3.8 Increment the field value of one or more than one document

To increment and update more than one document of a specific field, the `update()` command along with the '\$inc' function is used. When the '\$inc' function is used it would increment the value from the existing value which is already defined. For example, if the field of 'like' had a value of 5 and if it was incremented by 2, the final value would be 7 (5 + 2). The following is a command used to update and increment the value of specific fields with a desired value:

- **Command:** `db.StoreInventoryDetails.update({ 'inventory_levels.product_id': 'P020' }, { $inc: { 'inventory_levels.$.quantity_available': 2 } })`
- **Result:** Increments and updates the 'inventory_levels.\$.quantity_available' field by 2 values of all the documents consisting of the 'inventory_levels.product_id' as 'P020.' Hence, this command increments the quantity available for the product with ID 'P020' within the **inventory_levels** array by 2 from the existing value.

The Figure 3.25 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.StoreInventoryDetails.update({ 'inventory_levels.product_id': 'P020' }, {
...   $inc: {
...     'inventory_levels.$.quantity_available': 2
...   }
... })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
MacyStore>
```

Figure 3.25: Displays the result of incrementing the field values

3.3.9 Rename fields in documents

To **rename one or more than one document** matching specific criteria or fields, the **update()** command along with the '\$rename' function is used. It helps to rename all the fields matching the specified criteria. The following is a command used to rename and update more than one specific document with a desired value:

- **Command:** `db.StoreInventoryDetails.update({ 'inventory_levels.product_id': 'P045' }, { $rename: { 'store_name': 'store_place_name' } })`
- **Result:** Renames and updates the 'store_name' field with the new field value of 'store_place_name' field for all the documents consisting of the 'inventory_levels.product_id' as 'P045.' Hence, this command renames the 'store_name'

field to 'store_place_name' for the product with ID 'P045' within the `inventory_levels` array.

The Figure 3.26 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```
MacyStore> db.StoreInventoryDetails.update({ 'inventory_levels.product_id': 'P045' }, { $rename: { 'store_name': 'store_place_name' } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Figure 3.26: Rename fields in documents

3.3.10 Delete one or more documents

To delete one or more than one document matching a specific criteria or field, the `remove()` command along with the specific condition is used. It helps to delete the documents matching the specified criteria. The following is a command used to delete one or more than one specific document matching the given criteria:

- **Command:** `db.StoreInventoryDetails.remove({ 'store_id': 'S100' })`
- **Result:** Removes or deletes the documents with the 'store_id' field of value 'S100.' Hence, all the documents matching the `store_id` value of 'S100' are deleted in this scenario.

The Figure 3.27 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.StoreInventoryDetails.remove({ 'store_id': 'S100' })
DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDelete, or bulkWrite.
{ acknowledged: true, deletedCount: 1 }
MacyStore> db.StoreInventoryDetails.find({'store_id': 'S100'})
MacyStore> _
```

Figure 3.27: Delete one or more documents

3.3.11 Adding Index to the Collection

To **add an index to ease the search**, it can be added to specific fields by using the **createIndex()** command. The following is a command used to create a text index for a specific field:

- **Command:** `db.StoreInventoryDetails.createIndex({ 'location.address': 'text' })`
- **Result:** Creates a text index using the `createIndex()` command for the field `'location.address'`. Hence, text search operations can be performed on the address field.

The Figure 3.28 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

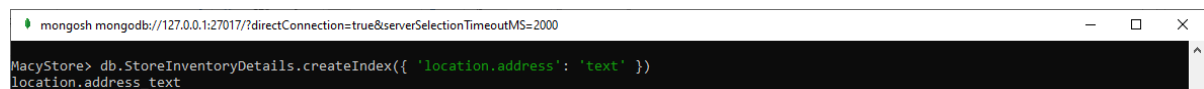


Figure 3.28: Displays the command and result of creating search index

3.3.12 Using text search for the created index

After creating the text index, you can **apply text search for the created index** using the **find()** command along with *'text' and 'search'* options. For example, if you created a text index for the address field, you can use this method to search for specific words in the address values. The following is a command used to use the text search option for the above-created Index:

- **Command:** `db.StoreInventoryDetails.find({ $text: { $search: 'Portland' } })`
- **Result:** Displays the results of documents consisting of the value "Portland" for the above-created index of `'location.address': 'text'`. Hence, this command performs a text search for documents containing the term 'Portland'.

The Figure 3.29 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```
MacyStore> db.StoreInventoryDetails.find({ $text: { $search: 'Portland' } })
[
  {
    _id: ObjectId('66212a43a7b8822edbf369a'),
    store_id: '5098',
    store_name: "Macy's Portland",
    location: { address: 'Pioneer Place, 621 SW 5th Ave, Portland' },
    inventory_levels: [
      {
        product_id: 'P080',
        quantity_available: 1030,
        reorder_point: 50
      },
      {
        product_id: 'P045',
        quantity_available: 970,
        reorder_point: 40
      },
      {
        product_id: 'P020',
        quantity_available: 1060,
        reorder_point: 60
      }
    ]
  },
  {
    _id: ObjectId('66212a43a7b8822edbf368f'),
    store_id: '5097',
    store_name: "Macy's Portland",
    location: { address: 'Pioneer Place, 621 SW 5th Ave, Portland' },
    inventory_levels: [
      {
        product_id: 'P025',
        quantity_available: 920,
        reorder_point: 50
      },
      {
        product_id: 'P090',
        quantity_available: 860,
        reorder_point: 40
      }
    ]
  }
]
```

Figure 3.29: Using text search for the created index

3.3.13 To Find by element in an Array in a collection

To find elements matching a specific condition, the '\$elemMatch' function is used. It helps to dive deep into the specific collections and find all the documents falling into the specific element match. The following is a command depicts the way the \$elemMatch is used to find specific documents:

- **Command:** `db.StoreInventoryDetails.find({ 'inventory_levels': { $elemMatch: { product_id: 'P045' } } })`
- **Result:** Finds and displays the documents having the `product_id` field value of 'P045' using the \$elemMatch function within the `inventory_levels` fields located inside the `StoreInventoryDetails` collection.

The Figure 3.30 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```
MacyStore> db.StoreInventoryDetails.find({ 'inventory_levels': { $elemMatch: { product_id: 'P045' } } })
[
  {
    id: ObjectId('6021253ca4ffeb94b3cf35bd'),
    store_id: '5006',
    location: { address: 'The Galleria, 5135 W Alabama St, Houston, TX 77056' },
    inventory_levels: [
      {
        product_id: 'P015',
        quantity_available: 140,
        reorder_point: 50
      },
      {
        product_id: 'P045', quantity_available: 80, reorder_point: 30
      },
      {
        product_id: 'P095',
        quantity_available: 160,
        reorder_point: 40
      },
      {
        product_id: 'P075',
        quantity_available: 100,
        reorder_point: 40
      }
    ],
    store_name: "Macy's Houston"
  },
  {
    id: ObjectId('6021253ca4ffeb94b3cf35c3'),
    store_id: '5012',
    store_name: "Macy's Las Vegas",
    location: {
      address: 'Fashion Show Mall, 3200 S Las Vegas Blvd, Las Vegas, NV 89109'
    },
    inventory_levels: [
      {
        product_id: 'P045',
        quantity_available: 170,
        reorder_point: 50
      },
      {
        product_id: 'P075',
        quantity_available: 110,

```

Figure 3.30: Finding by element in Array in the collection

3.3.14 To Find documents greater or less than specific values of specific fields

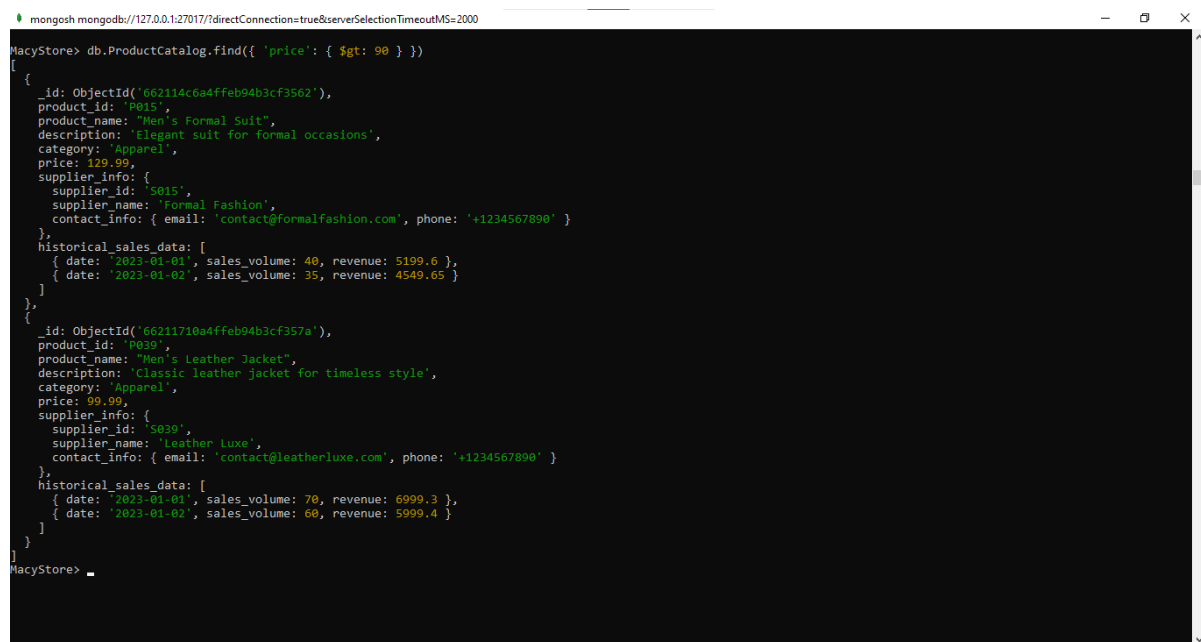
Finding documents **greater or less than specific values** of specific fields In this section, five scenarios are covered on different ways to find values greater or less than or equal to specific fields as follows:

- **\$gt**: Finds values greater than a specific value in a specific field.
- **\$gte**: Finds values greater than or equal a specific value in a specific field.
- **\$lt**: Finds values less than a specific value in a specific field.
- **\$lte**: Finds values less than or equal a specific value in a specific field.
- **Using the pairs of the above combination of commands** (example: using \$gt and \$lte together).

Using \$gt to find values greater than a specific value in a specific field The following command uses the command \$gt to find values greater than a specific value in a specific field:

- **Command:** `db.ProductCatalog.find({ 'price': { $gt: 90 } })`
- **Result:** Displays the documents having a `price` field greater than 90 in the `ProductCatalog` collection. Using \$gte finds values greater than or equal to a specific value in a specific field.

The Figure 3.31 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.ProductCatalog.find({ 'price': { $gt: 90 } })
[
  {
    _id: ObjectId('662114c6a4ffeb94b3cf3562'),
    product_id: 'P015',
    product_name: 'Men's Formal Suit',
    description: 'Elegant suit for formal occasions',
    category: 'Apparel',
    price: 129.99,
    supplier_info: {
      supplier_id: 'S015',
      supplier_name: 'Formal Fashion',
      contact_info: { email: 'contact@formalfashion.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 40, revenue: 5199.6 },
      { date: '2023-01-02', sales_volume: 35, revenue: 4549.65 }
    ]
  },
  {
    _id: ObjectId('66211710a4ffeb94b3cf357a'),
    product_id: 'P039',
    product_name: 'Men's Leather Jacket',
    description: 'Classic leather jacket for timeless style',
    category: 'Apparel',
    price: 99.99,
    supplier_info: {
      supplier_id: 'S039',
      supplier_name: 'Leather Luxe',
      contact_info: { email: 'contact@leatherluxe.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 70, revenue: 6999.3 },
      { date: '2023-01-02', sales_volume: 60, revenue: 5999.4 }
    ]
  }
]
```


Figure 3.31: Usage of \$gt command

The following command uses the \$gte to find values greater than or equal to a specific value in a specific field:

- **Command:** `db.ProductCatalog.find({ 'price': { $gte: 120 } })`

- **Result:** Displays the documents having a **price** field greater than or equal to 120 in the ProductCatalog collection. Using \$lt finds values less than a specific value in a specific field.

The Figure 3.32 displays the usage of the above-mentioned command and the expected result based in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.ProductCatalog.find({ 'price': { $gte: 120 } })
[
  {
    _id: ObjectId('602114c6a4ffeb94b3cf3562'),
    product_id: 'P015',
    product_name: 'Men's Formal Suit',
    description: 'Elegant suit for formal occasions',
    category: 'Apparel',
    price: 129.99,
    supplier_info: {
      supplier_id: 'S015',
      supplier_name: 'Formal Fashion',
      contact_info: { email: 'contact@formalfashion.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 40, revenue: 5199.6 },
      { date: '2023-01-02', sales_volume: 35, revenue: 4549.65 }
    ]
  }
]
```

Figure 3.32: Usage of \$gte command

The following command uses the command \$lt command which helps to find values which are less than a specific value:

- **Command:** `db.ProductCatalog.find({ 'price': { $lt: 10 } })`
- **Result:** Displays the documents which have the **price** field of less than 10 in the ProductCatalog collection. By using the \$lt it is able to find values which are less than a specific value in a specific field.

The Figure 3.33 displays the usage of the above-mentioned command in the 'MacyStore' collection:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.ProductCatalog.find({ 'price': { $lt: 10 } })
[
  {
    _id: ObjectId('66211710a4ffeb94b3cf3584'),
    product_id: 'P049',
    product_name: 'Men's Cotton Socks',
    description: 'Comfortable cotton socks for everyday wear',
    category: 'Apparel',
    price: 9.99,
    supplier_info: {
      supplier_id: 'S049',
      supplier_name: 'Sock Styles',
      contact_info: { email: 'contact@sockstyles.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 150, revenue: 1498.5 },
      { date: '2023-01-02', sales_volume: 130, revenue: 1298.7 }
    ]
  }
]
```

Figure 3.33: Usage of \$lt command

The following command uses the command \$lte which helps to find values less than or equal to a specific value in a given field:

- **Command:** `db.ProductCatalog.find({ 'price': { $lte: 20 } })`
- **Result:** Displays the documents having the `price` field less than or equal to 20 in the `ProductCatalog` collection.

The Figure 3.34 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```

mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.ProductCatalog.find({ 'price': { $lte: 20 } })
[
  {
    _id: ObjectId('662114c6a4ffeb94b3cf355b'),
    product_id: 'P008',
    product_name: 'Women's Basic T-Shirt',
    description: 'Essential cotton t-shirt for everyday comfort',
    category: 'Apparel',
    price: 19.99,
    supplier_info: {
      supplier_id: 'S008',
      supplier_name: 'Tee Trends',
      contact_info: { email: 'contact@teetrends.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 160, revenue: 3198.4 },
      { date: '2023-01-02', sales_volume: 120, revenue: 2398.8 }
    ]
  },
  {
    _id: ObjectId('662114c6a4ffeb94b3cf3564'),
    product_id: 'P017',
    product_name: 'Men's Athletic Shorts',
    description: 'Comfortable shorts for workouts and sports',
    category: 'Apparel',
    price: 19.99,
    supplier_info: {
      supplier_id: 'S017',
      supplier_name: 'Athletic Apparel Inc.',
      contact_info: { email: 'contact@athleticapparel.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 150, revenue: 2998.5 },
      { date: '2023-01-02', sales_volume: 130, revenue: 2598.7 }
    ]
  },
  {
    _id: ObjectId('66211710a4ffeb94b3cf357b'),
    product_id: 'P040',
    product_name: 'Women's Wool Scarf',
    description: 'Soft wool scarf for warmth and style',
    category: 'Apparel',
    price: 19.99,
  }
]

```

Figure 3.34: Usage of \$lte command

3.3.15 Usage of \$regex command in documents

The **\$regex** is the Regular expression command used to **find specific values in different ways**. The following are the different ways to use \$regex: \$regex: */^il/I* -> Indicates that the value *il* should be at the beginning of the string. \$regex: */ces/I* -i Indicates that the value “ces” should be in among the last three letters \$regex: */Reg/I* -i Indicates that the value “Reg” should be somewhere in the specific value or field.

The following is an example of using the \$regex command:

- **Command:** `db.ProductCatalog.find({ product_name: { $regex: /shirt/i } })`
- **Result:** This command uses the \$regex operator to search the value “shirt” somewhere in the ‘product_name’ field. Also, this command performs a case-insensitive regular expression search and returns the documents where the product_name contains the word ‘shirt’.

The Figure 3.35 displays the usage of the above-mentioned command and the expected result of using the specified command in the ‘MacyStore’ collection:

```
MacyStore> db.ProductCatalog.find({ product_name: { $regex: /shirt/i } });
{
  _id: ObjectId('6621125aa4ffeb94b3cf3554'),
  product_id: 'P001',
  product_name: 'Men's Casual Shirt',
  description: 'Soft cotton shirt for everyday wear',
  category: 'Apparel',
  price: 29.99,
  supplier_info: {
    supplier_id: 'S001',
    supplier_name: 'Fashion Suppliers Inc.',
    contact_info: { email: 'contact@fashionsuppliers.com', phone: '+1234567890' }
  },
  historical_sales_data: [
    { date: '2023-01-01', sales_volume: 120, revenue: 3597.6 },
    { date: '2023-01-02', sales_volume: 90, revenue: 2699.1 }
  ]
},
{
  _id: ObjectId('662114c6a4ffeb94b3cf355a'),
  product_id: 'P007',
  product_name: 'Men's Hooded Sweatshirt',
  description: 'Warm sweatshirt with a hood for chilly days',
  category: 'Apparel',
  price: 27.99,
  supplier_info: {
    supplier_id: 'S007',
    supplier_name: 'Sweater Co.',
    contact_info: { email: 'contact@sweaterco.com', phone: '+1234567890' }
  },
  historical_sales_data: [
    { date: '2023-01-01', sales_volume: 130, revenue: 3638.7 },
    { date: '2023-01-02', sales_volume: 100, revenue: 2799 }
  ]
},
{
  _id: ObjectId('662114c6a4ffeb94b3cf355b'),
  product_id: 'P008',
  product_name: 'Women's Basic T-Shirt',
  description: 'Essential cotton t-shirt for everyday comfort',

```

Figure 3.35: Using \$regex command in documents

3.3.16 Usage of \$in command in documents

You can use **\$in** the command to **look for specific values in specific fields**. The following is an example of using the **\$in** the command

- **Command:** `db.ProductCatalog.find({ product_name: { $in: ['Men's Casual Shirt', 'Women's Wool Coat'] } })`
- **Result:** The **\$in** command used in this example finds for 'product_name' field in the specified array. Thus, it returns documents with product names 'Men's Casual Shirt' or 'Women's Wool Coat' which are located within the **ProductCatalog** collection.

The Figure 3.36 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```

mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.ProductCatalog.find({ product_name: { $in: ['Men\'s Casual Shirt', 'Women\'s Wool Coat'] } })
[
  {
    _id: ObjectId('6621125aa4ffeb94b3cf3554'),
    product_id: 'P001',
    product_name: 'Men\'s Casual Shirt',
    description: 'Soft cotton shirt for everyday wear',
    category: 'Apparel',
    price: 29.99,
    supplier_info: {
      supplier_id: 'S001',
      supplier_name: 'Fashion Suppliers Inc.',
      contact_info: { email: 'contact@fashionsuppliers.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 120, revenue: 3597.6 },
      { date: '2023-01-02', sales_volume: 90, revenue: 2699.1 }
    ]
  },
  {
    _id: ObjectId('662114c6a4ffeb94b3cf3565'),
    product_id: 'P018',
    product_name: 'Women\'s Wool Coat',
    description: 'Warm wool coat for cold weather',
    category: 'Apparel',
    price: 89.99,
    supplier_info: {
      supplier_id: 'S018',
      supplier_name: 'Woolly Wonders',
      contact_info: { email: 'contact@woollywonders.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 60, revenue: 5399.4 },
      { date: '2023-01-02', sales_volume: 50, revenue: 4499.5 }
    ]
  }
]
MacyStore>

```

Figure 3.36: Using \$in command in documents

3.3.17 Usage of \$ne command in documents

You can use **\$ne** command to **eliminate specific values in specific fields**. The following is an example of using the \$ne command

- **Command:** `db.ProductCatalog.find({ price: { $ne: 29.99 } })`
- **Result:** The \$ne command used in this example eliminates the value of '29.99' in the 'price' field in the specified array. Thus, it returns documents that don't have a price value of 29.99 within the ProductCatalog collection.

The Figure 3.37 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```
MacyStore> db.ProductCatalog.find({ price: { $ne: 39.99 } })
[
  {
    _id: ObjectId('6621125aa4ffeb94b3cf3554'),
    product_id: 'P001',
    product_name: 'Men's Casual Shirt',
    description: 'Soft cotton shirt for everyday wear',
    category: 'Apparel',
    price: 29.99,
    supplier_info: {
      supplier_id: 'S001',
      supplier_name: 'Fashion Suppliers Inc.',
      contact_info: { email: 'contact@fashionsuppliers.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 120, revenue: 3597.6 },
      { date: '2023-01-02', sales_volume: 90, revenue: 2699.1 }
    ]
  },
  {
    _id: ObjectId('6621138da4ffeb94b3cf3556'),
    product_id: 'P003',
    product_name: 'Men's Leather Belt',
    description: 'Genuine leather belt for a classic look',
    category: 'Apparel',
    price: 24.99,
    supplier_info: {
      supplier_id: 'S003',
      supplier_name: 'Leather Works',
      contact_info: { email: 'contact@leatherworks.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 80, revenue: 1999.2 },
      { date: '2023-01-02', sales_volume: 70, revenue: 1749.3 }
    ]
  },
  {
    _id: ObjectId('6621138da4ffeb94b3cf3557'),
    product_id: 'P004',
    product_name: 'Women's Floral Dress',
    description: 'Elegant floral dress for special occasions',
    category: 'Apparel',
    price: 39.99,
    supplier_info: {
      supplier_id: 'S004',
      supplier_name: 'Fashion Suppliers Inc.',
      contact_info: { email: 'contact@fashionsuppliers.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 100, revenue: 3999.0 },
      { date: '2023-01-02', sales_volume: 80, revenue: 3199.2 }
    ]
  }
]
```

Figure 3.37: Using \$ne command in documents

3.3.18 Usage of \$nin command in documents

You can use \$nin command to find documents ‘not in’ or eliminate specific values in specific fields. The following is an example of using the \$nin command

- **Command:** `db.ProductCatalog.find({ product_name: { $nin: ['Men's Casual Shirt', 'Women's Wool Coat'] } })`
- **Result:** The \$nin command used in this example eliminates the value of the 'product_name' field that is not 'Men's Casual Shirt' or 'Women's Wool Coat'. Thus, it returns documents that don't have product names of 'Men's Casual Shirt' or 'Women's Wool Coat'.

The Figure 3.38 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

```
MacyStore> db.ProductCatalog.find({ product_name: { $nin: ['Men\'s Casual Shirt', 'Women\'s Wool Coat'] } })
[
  {
    _id: ObjectId('6621138da4ffeb94b3cf3555'),
    product_id: 'P002',
    product_name: 'Women\'s Skinny Jeans',
    description: 'Stretch denim jeans for a comfortable fit',
    category: 'Apparel',
    price: 39.99,
    supplier_info: {
      supplier_id: 'S002',
      supplier_name: 'Denim Delights',
      contact_info: { email: 'contact@denimdelights.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 150, revenue: 5998.5 },
      { date: '2023-01-02', sales_volume: 110, revenue: 4399.9 }
    ]
  },
  {
    _id: ObjectId('6621138da4ffeb94b3cf3556'),
    product_id: 'P003',
    product_name: 'Men\'s Leather Belt',
    description: 'Genuine leather belt for a classic look',
    category: 'Apparel',
    price: 24.99,
    supplier_info: {
      supplier_id: 'S003',
      supplier_name: 'Leather Works',
      contact_info: { email: 'contact@leatherworks.com', phone: '+1234567890' }
    },
    historical_sales_data: [
      { date: '2023-01-01', sales_volume: 80, revenue: 1999.2 },
      { date: '2023-01-02', sales_volume: 70, revenue: 1749.3 }
    ]
  },
  {
    _id: ObjectId('6621138da4ffeb94b3cf3557'),
    product_id: 'P004',
    product_name: 'Women\'s Floral Dress',
    description: 'Elegant floral dress for special occasions',
    category: 'Apparel',
  }
]
```

Figure 3.38: Using \$nin command in documents

Chapter 4

Using MongoDB Compass To Analyze the Key Questions To Solve The Problem

MongoDB compass can also be used to analyze documents and look for results. It helps to give an enhanced view including visuals. In this section, a couple of examples are discussed to analyze Macy's business-related documents stored in the MongoDB. Also, this section aims to analyze the key questions of Macy's store under each hypothesis and discuss in detail.

4.1 Analysis on Key Question 1

Key question 1: How to the analysis of product prices within a specific range help Macy's maintain optimal inventory levels?

As an example, by understanding which products are most commonly priced between \$30 and \$35, Macy's can identify popular items. Also, they can ensure these are adequately stocked to meet consumer demand, thereby avoiding both overstocking and stockouts.

To **analyze the prices of products with specific criteria**, the following query is used in MongoDB Compass search:

- **Command:** `price: { $gte: 30, $lt: 35 }`

- **Results:** Finds and displays the documents of products having the price field and falling between the value of greater than or equal to 30 and less than 35 of value in the Product-Catalog collection.

The Figure 4.1 displays the usage of the above-mentioned command and the expected result of using the specified command in the 'MacyStore' collection:

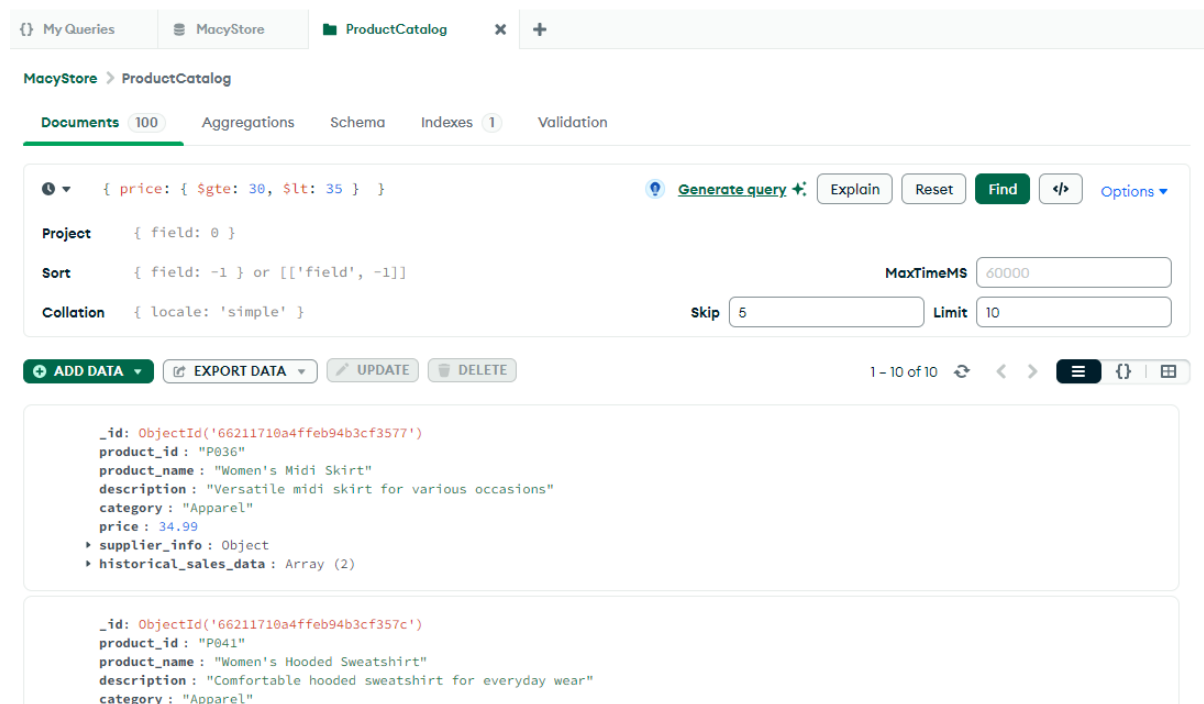


Figure 4.1: Analyzing the prices of Products

As per the above analysis and the Figure, about **14 products are falling into the price range of greater than or equal to 30** and less than 35 in value. Among them, 93% have a price value of 34.99 and the remaining 7% has 32.99.

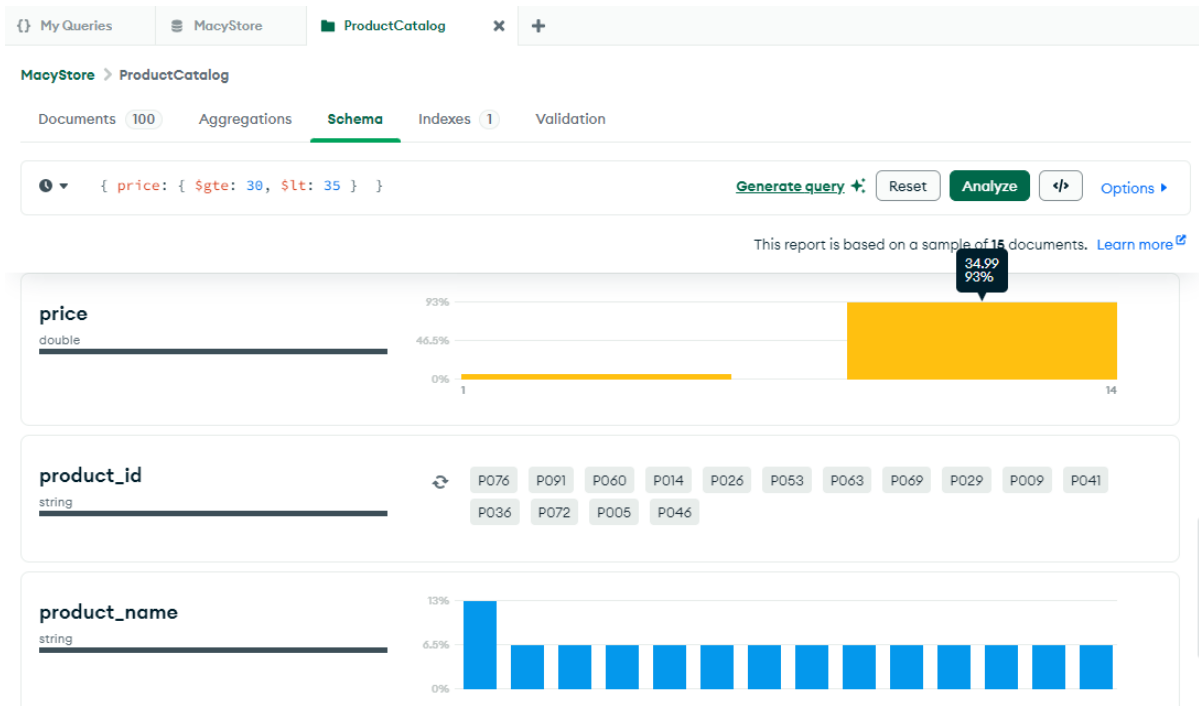


Figure 4.2: Result of analyzing the prices of Products

4.2 Analysis on Key Question 2

Key Question 2: What insights can be obtained by analyzing a specific supplier in managing inventory more effectively?

Supplier specific analyses helps to understand the products supplied by a specific supplier by identifying the popular items. By recognizing which suppliers and products have consistent demand, Macy's can prioritize these items in their inventory to align with market demand and avoid shortages.

To analyze the **type of good provided by a specific supplier**, the following query is used in MongoDB Compass search:

- **Command:** `"supplier_info.supplier_name": "Slim Styles"`
- **Results:** Finds and displays the documents of products supplied by the 'Slim Styles'

supplier in the ProductCatalog collection according to Figure 4.3.

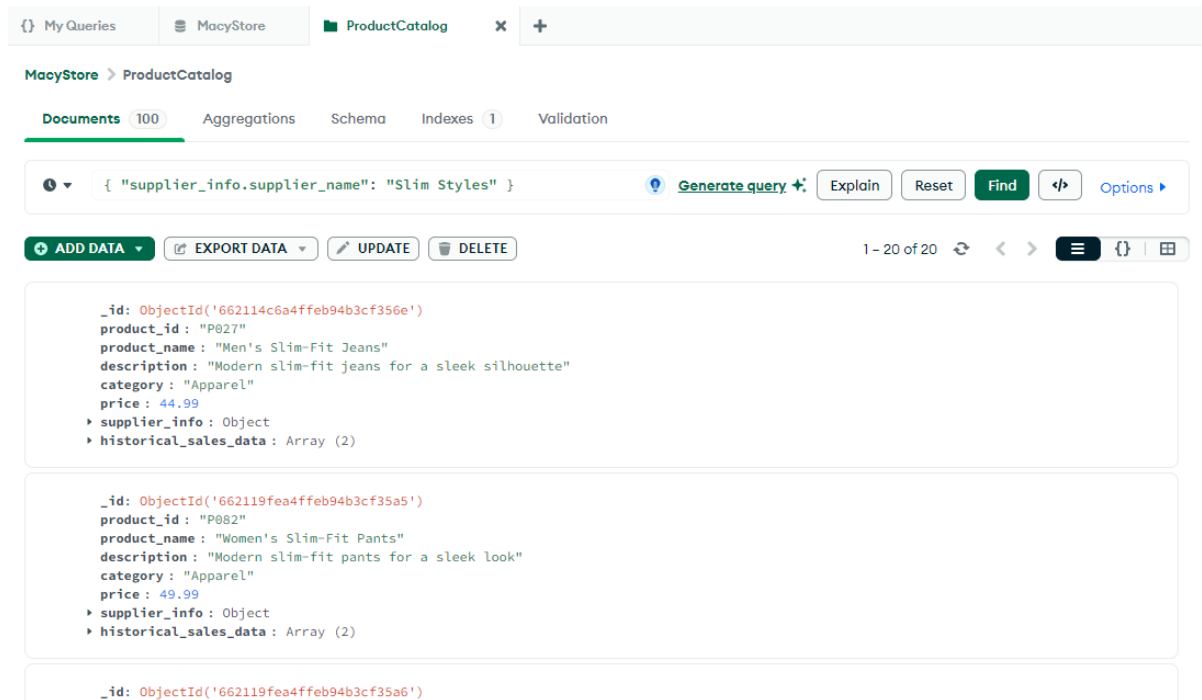


Figure 4.3: Analyzing a specific supplier's products

As per the above analysis and the Figure 4.4, there are about **20 products distributed** by the supplier 'Slim Styles' supplier for the Apparel category. Among them, **30%** of products fall into the price range of **39.99** which is the highest percentage.

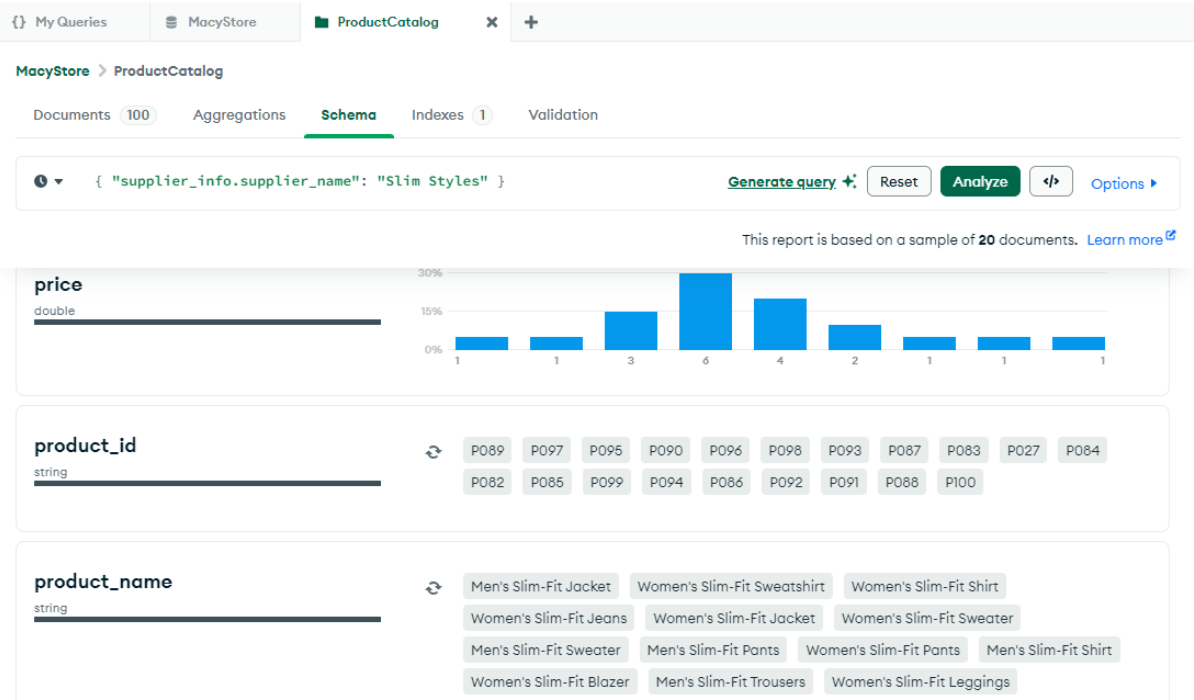


Figure 4.4: Results after analyzing a specific supplier’s products

4.3 Analysis on Key Question 3

Key Question 3: How to identify the Customer Preference to identify potential fluctuations in consumer demand?

Knowing that percentage of customers based on gender, Macy’s can tailor their inventory to match customer demographics. This data allows Macy’s to stock products purchased by their primary customer base, and reduce inventory waste and meeting consumer demand more accurately.

During the analysis of the Customer Preference collection, it was acknowledged that around **52% are male customers and 42% are female customers** as depicted in Figure 4.5.

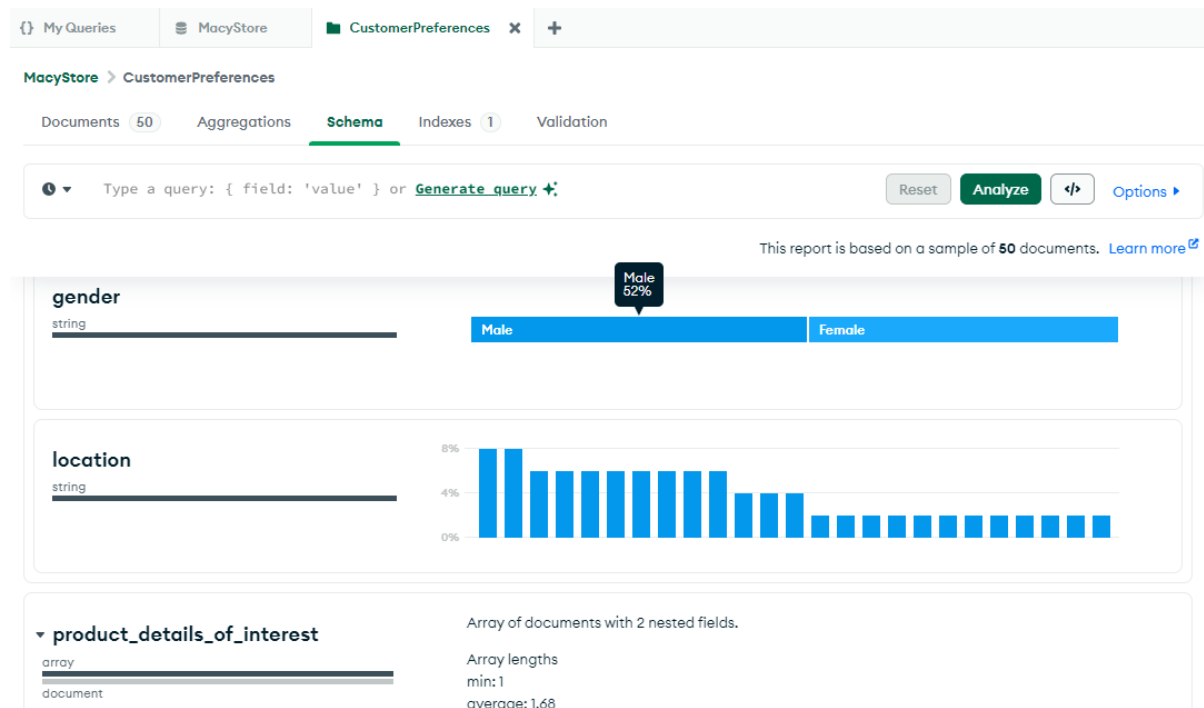


Figure 4.5: Analyzing the CustomerPreference collection

After further analysis using the command `{ "demographic_information.gender": "Male" }`, there were **15% male customers (highest range)** falling into the range of 30 age and depicted in Figure 4.6.

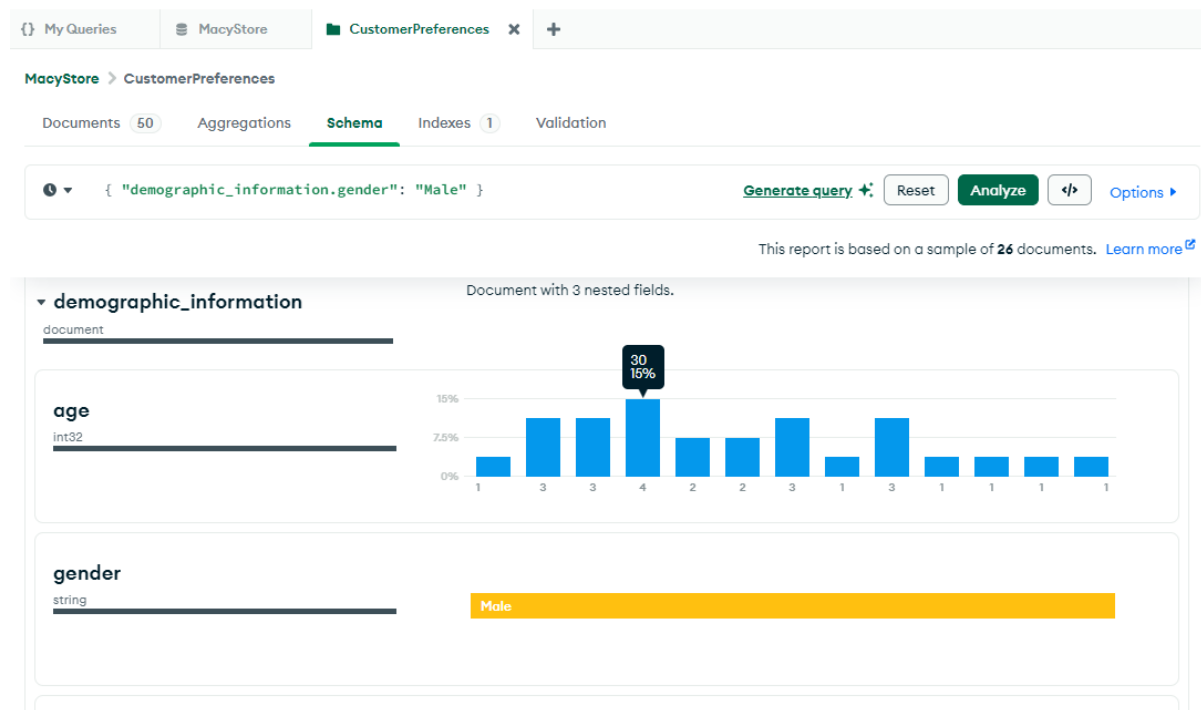


Figure 4.6: Results during the analysis of the CustomerPreference collection

Also, these **specific customers are from the area** of Houston, TXMiami, FLColumbus, OH and are highly interested in the following products displayed in Figure 4.7.

- **Command:** `"demographic_information.gender": "Male" ,`
`"demographic_information.age": 30`

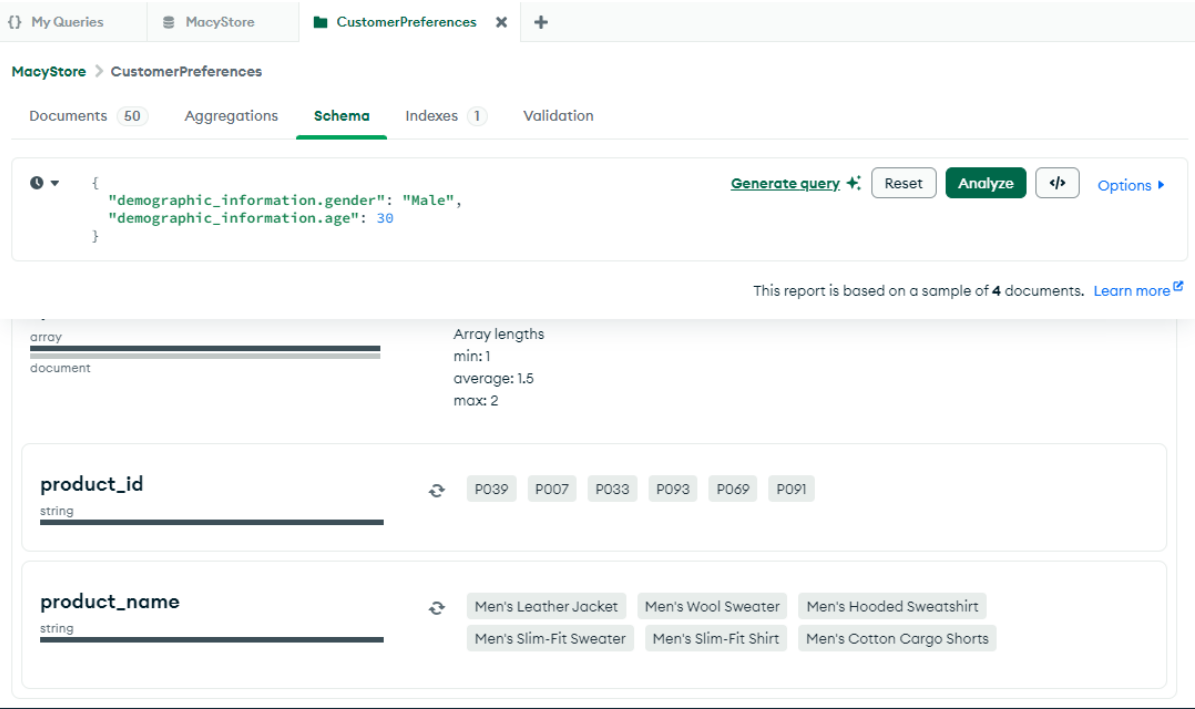


Figure 4.7: Products interested by the target demographics

4.4 Analysis on Key Question 4

Key Question 4: How to optimize inventory management during high sales periods?

Identifying the highest sales occurred for a product between seasons; times helps to identify peak sales periods. Thus, Macy can organize the timing of orders from suppliers and ensures that popular items are in stock during high-demand periods. After analyzing the historic sales data, the **highest sales** have occurred for the **product ID of ‘P014’** which is displayed in Figure 4.8.

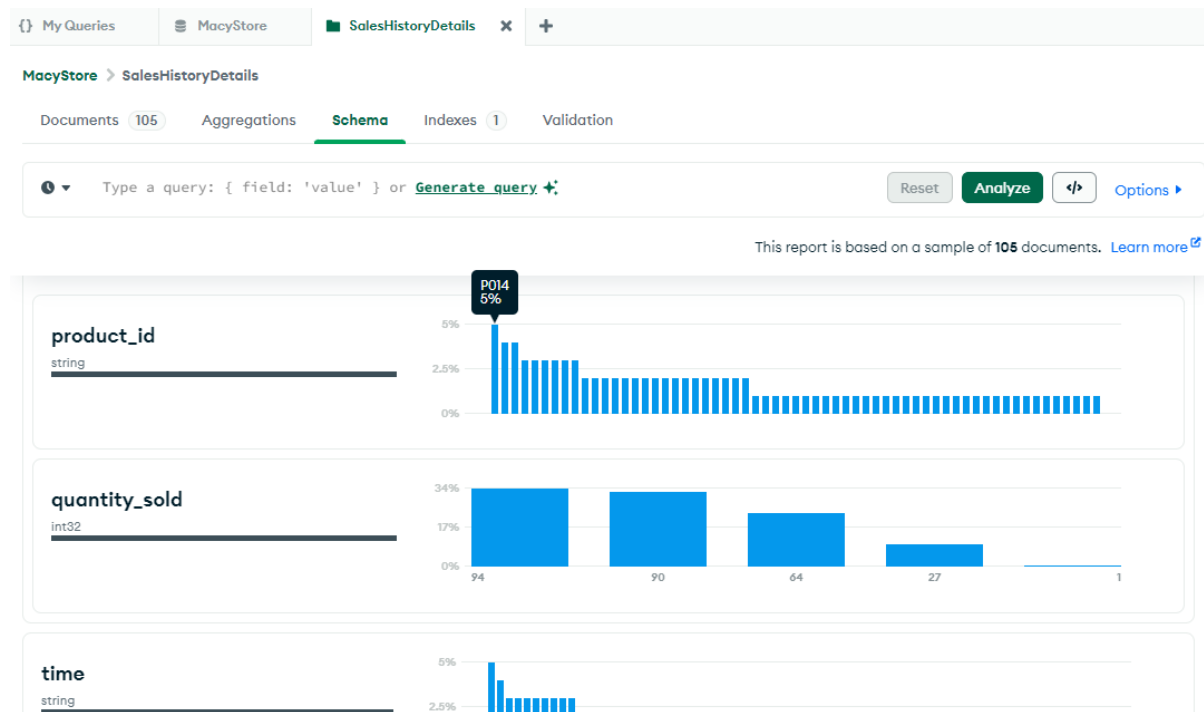


Figure 4.8: The amount of sales occurrence

After further analysis, it is concluded that this product was widely **sold between October 14 to December 26 2023**. However, more sales have occurred in December compared to other months.

Also, further details about this, the sales data can be gained through the **query** of `{ "product_details.product_id": "P014" }` from the MongoDB Compass and depicted in Figure 4.9.

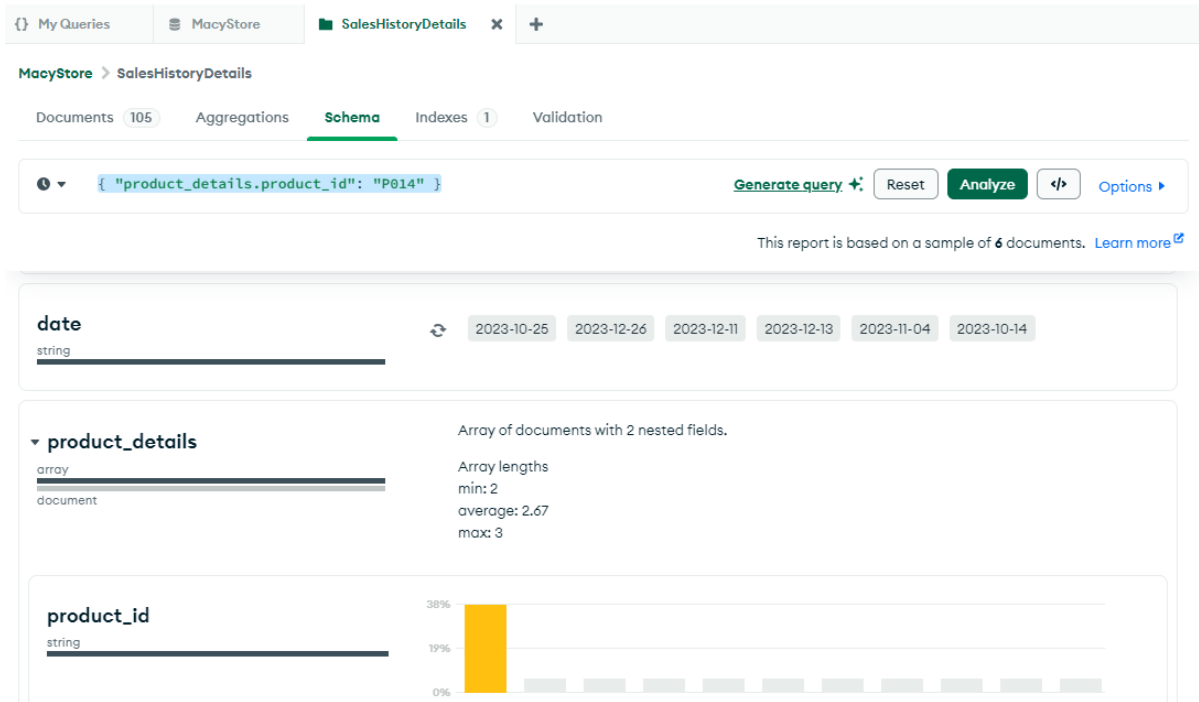


Figure 4.9: Usage of sales query in MongoDB Compass

4.5 Analysis on Key Question 5

Key Question 5: How to analyze the store inventory levels to prevent the inability to meet supplies and place on-time orders?

By analyzing inventory levels, such as the low availability of pertain products helps to proactively manage stock levels. It helps to know where inventory is low helps prevent shortages, ensuring that stores are restocked in a timely manner based on actual demand, thus maintaining optimal inventory levels.

To optimize the store inventory levels, the **product ID “P060”** is selected to analyze its data according to Figure 4.10. Based on the analysis, this product is **widely available in many stores**. However, the **lowest available level of this product is 750 to 800** quantities which is only 2% when compared to other store availability.

- **Command:** `"inventory_levels.product": "P060"`

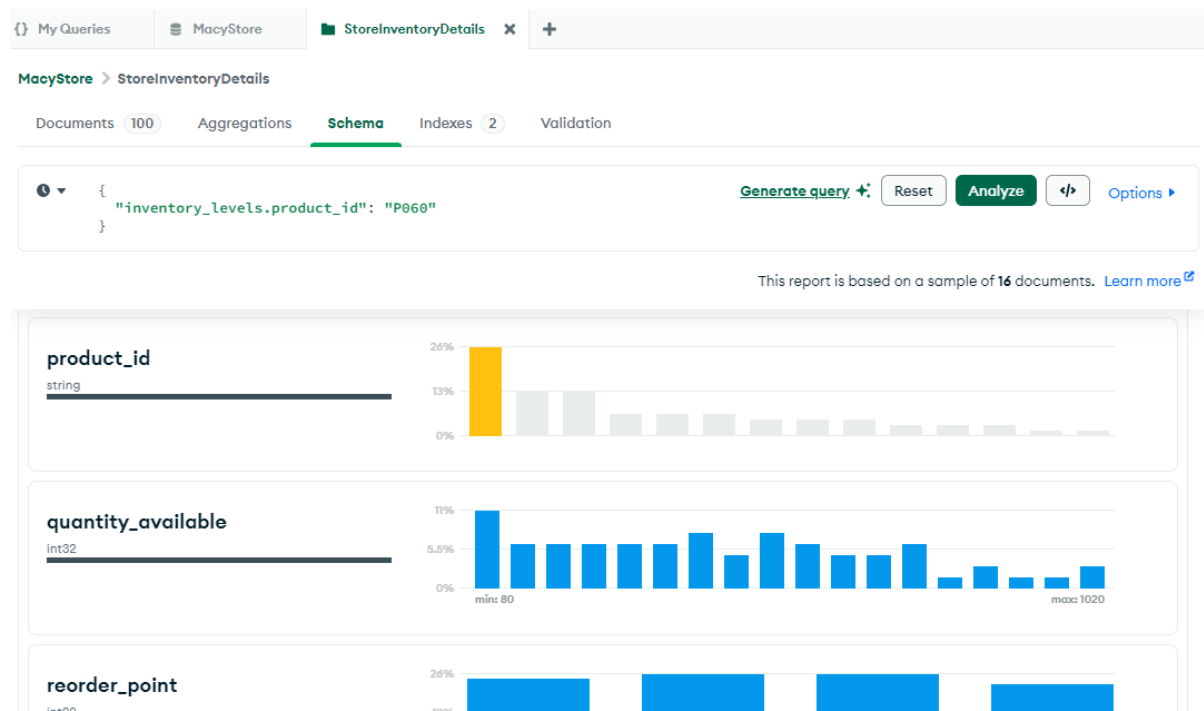


Figure 4.10: Analyzing the Store Inventory levels

Upon further analysis, it is derived that, this quantity level is available in the ‘Macy’s Indianapolis 02’ store which has the **store id of ‘S074’**. Hence, this store has to be closely watched and restocked considering the sales rate or the reorder point can be optimized and depicted in Figure 4.11.

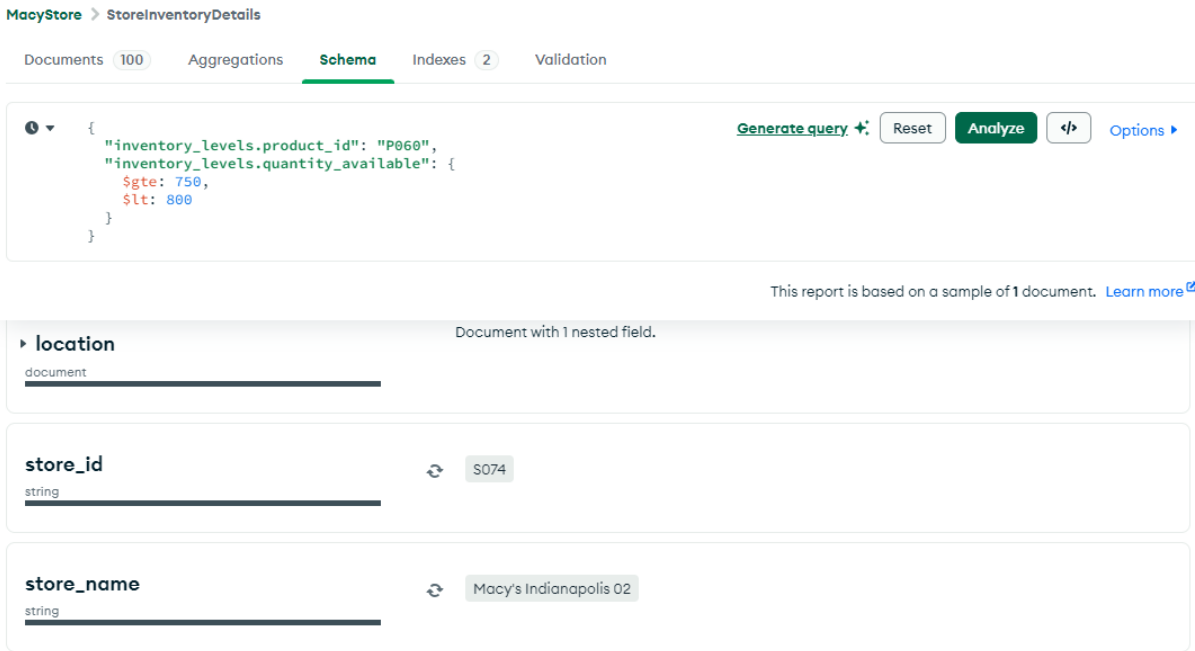


Figure 4.11: Results after analyzing the Store Inventory levels

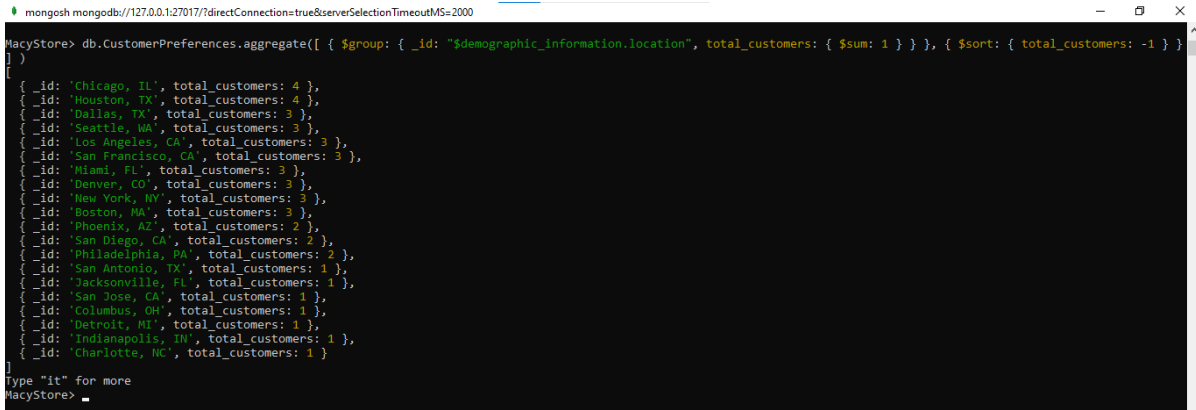
Chapter 5

Discussion on Hypothesis related to Macy's Store

In this section, a couple of hypothesis related to the problem context of Macy's store is discussed in detail. It includes the hypothesis and the discussion on whether the hypothesis is true or false correspondingly.

5.1 Hypothesis 1: Customers of 'Chicago, IL' are richer than 'Detroit, MI.'

- **Hypothesis:** Customers of 'Chicago, IL' are richer than 'Detroit, MI.'
- **Command to check hypothesis:** `db.CustomerPreferences.aggregate([{ $group: { _id: "$demographic_information.location", total_customers: { $sum: 1 } } }, { $sort: { total_customers: -1 } }])`
- **Result:** True
- **Explanation:** Customers of 'Chicago, IL' tend to make more purchases than 'Detroit, MI' as there are 4 customers located in Chicago, IL and only 1 customer located in Detroit, MI based on the available documents of data as shown in Figure 5.1.



```

mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.CustomerPreferences.aggregate([ { $group: { _id: "$demographic_information.location", total_customers: { $sum: 1 } } }, { $sort: { total_customers: -1 } } ] )
[
  { _id: 'Chicago, IL', total_customers: 4 },
  { _id: 'Houston, TX', total_customers: 4 },
  { _id: 'Dallas, TX', total_customers: 3 },
  { _id: 'Seattle, WA', total_customers: 3 },
  { _id: 'Los Angeles, CA', total_customers: 3 },
  { _id: 'San Francisco, CA', total_customers: 3 },
  { _id: 'Miami, FL', total_customers: 3 },
  { _id: 'Denver, CO', total_customers: 3 },
  { _id: 'New York, NY', total_customers: 3 },
  { _id: 'Boston, MA', total_customers: 3 },
  { _id: 'Phoenix, AZ', total_customers: 2 },
  { _id: 'San Diego, CA', total_customers: 2 },
  { _id: 'Philadelphia, PA', total_customers: 2 },
  { _id: 'San Antonio, TX', total_customers: 1 },
  { _id: 'Jacksonville, FL', total_customers: 1 },
  { _id: 'San Jose, CA', total_customers: 1 },
  { _id: 'Columbus, OH', total_customers: 1 },
  { _id: 'Detroit, MI', total_customers: 1 },
  { _id: 'Indianapolis, IN', total_customers: 1 },
  { _id: 'Charlotte, NC', total_customers: 1 }
]
Type "it" for more
MacyStore>

```

Figure 5.1: Results of Hypothesis 1

5.2 Hypothesis 2: Seasonal months have more Sales

- Hypothesis: There are more sales in the seasonal months.
- Command to check hypothesis during the seasonal months such as October and December:

```

db.SalesHistoryDetails.aggregate([ { $match: { $or: [ { date: { $regex: /2023-10/ } }, { date: { $regex: /2023-12/ } } ] } }, { $group: { _id: { $substr: ["$date", 0, 7] }, total_products_sold: { $sum: { $size: "$product_details" } } } } ] )

```

- Result: False
- Explanation: Only 86 products of sales amount in October but in December 104 products were sold as shown in Figure 5.2. Hence, the hypothesis that more number of products are sold in October due to Halloween than in December is incorrect.

```

mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.SalesHistoryDetails.aggregate([
...   { $match: { $or: [{ date: { $regex: /^2023-10/ } }, { date: { $regex: /^2023-12/ } } ] } },
...   { $group: { _id: { $substr: ["$date", 0, 7] }, total_products_sold: { $sum: { $size: "$product_details" } } } }
... ])
[
  { _id: '2023-10', total_products_sold: 86 },
  { _id: '2023-12', total_products_sold: 164 }
]
MacyStore>

```

Figure 5.2: Results of Hypothesis 3

5.3 Hypothesis 3: The top 5 products bring more sales

- **Hypothesis:** The top 5 products bring more sales.
- **Command to check hypothesis of the top 5 products having more sales:**

```

db.SalesHistoryDetails.aggregate([
  { $unwind: "$product_details" },
  { $group: { _id: "$product_details.product_id", total_quantity_sold: { $sum:
"$product_details.quantity_sold" } } },
  { $sort: { total_quantity_sold: -1 } },
  { $limit: 5 }])

```
- **Result:** True
- **Explanation:** Among the top 5 high-sales products, the product 'P091' is in second place with a sales count of 14 products as shown in Figure 5.3.

```
MacyStore> db.SalesHistoryDetails.aggregate([
...   {
...     $unwind: "$product_details"
...   },
...   {
...     $group: {
...       _id: "$product_details.product_id",
...       total_quantity_sold: {
...         $sum: "$product_details.quantity_sold"
...       }
...     }
...   },
...   {
...     $sort: { total_quantity_sold: -1 }
...   },
...   {
...     $limit: 5
...   }
... ])
[
  { _id: 'P032', total_quantity_sold: 16 },
  { _id: 'P091', total_quantity_sold: 14 },
  { _id: 'P096', total_quantity_sold: 14 },
  { _id: 'P079', total_quantity_sold: 14 },
  { _id: 'P087', total_quantity_sold: 13 }
]
```

Figure 5.3: Results of Hypothesis 3

5.4 Hypothesis 4: Younger customers make more purchases than older customers

- **Hypothesis:** Younger customers make more purchases than older customers.

- **Command used to check hypothesis:** `db.CustomerPreferences.aggregate([`
`{ $facet: { "age_25_29": [{ $match: {`
`"demographic_information.age": { $gte: 25, $lte: 29 } } }, { $count: "count"`
`}] ,`
`"age_30_and_above": [{ $match: { "demographic_information.age": {`
`$gte: 30 } } }, { $count: "count" }] } } ,`
`{ $project: { _id: 0, age_25_29: { $ifNull: [{ $arrayElemAt:`
`"$age_25_29.count", 0] }, 0] },`
`age_30_and_above: { $ifNull: [{ $arrayElemAt: [`
`"$age_30_and_above.count", 0] }, 0] } }]])`
- **Result:** False
- **Explanation:** Customers between the ages of 25 to 29 are only 19 in the customer preference collection, while customers aged 30 or more are 31 as shown in Figure 5.4.

```

mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
MacyStore> db.CustomerPreferences.aggregate([ { $facet: { "age_25_29": [ { $match: { "demographic_information.age": { $gte: 25, $lte: 29 } } }, { $count: "count" } ], "age_30_and_above": [ { $match: { "demographic_information.age": { $gte: 30 } } }, { $count: "count" } ] } }, { $project: { _id: 0, age_25_29: { $ifNull: [ { $arrayElemAt: [ "$age_25_29.count", 0 ] }, 0 ] }, age_30_and_above: { $ifNull: [ { $arrayElemAt: [ "$age_30_and_above.count", 0 ] }, 0 ] } } } ] ])
MacyStore>
{ 'age_25_29': 19, 'age_30_and_above': 31 }

```

Figure 5.4: Results of Hypothesis 4

Chapter 6

Conclusion

In conclusion, this report briefly discussed the problem context of optimizing inventory allocation for Macy's stores for the apparel section. To analyse the problem, key collections such as CustomerPreferences, StoreInventoryDetails, SalesHistoryDetails, and ProductCatalog were created using MongoDB. Even though there are many challenges as discussed, these challenges can be overcome through a strategic and data-driven approach.

Hence, the MongoDB family of databases including the application of NoSQL, Mongoosh shell, and MongoDB Compass was used to handle the problem effectively and help Macy's store enhance its ability to forecast customer demand accurately, optimize inventory levels, and improve overall operational efficiency.

By addressing the key challenges and using innovative solutions, Macy's stores can achieve greater agility, responsiveness, and competitiveness in the retail landscape and deliver excellent service to customers and stakeholders.

Bibliography

Alvin.Magestore (2018). Macy's: An omnichannel case study for retailers 2018.

URL: <https://medium.com/@alvindang.magestore/macys-an-omnichannel-case-study-for-retailers-2018-f4934bf03da4>

Dataiku (n.d.). Sql pipelines in dss.

URL: <https://doc.dataiku.com/dss/latest/sql/pipelines/index.html>

DB, M. (n.d.). What is nosql?

URL: <https://www.mongodb.com/resources/basics/databases/nosql-explained>

Geeksforgeeks (2021). What is mongodb – working and features.

URL: <https://www.geeksforgeeks.org/what-is-mongodb-working-and-features/>

Geeksforgeeks (2024). Mongodb compass.

URL: <https://www.geeksforgeeks.org/mongodb-compass/>

IBM (n.d.). What is a data pipeline?

URL: <https://www.ibm.com/topics/data-pipeline>

Macy (2023). Macy's store.

URL: <https://www.macys.com/>

Services, A. W. (n.d.). What is mysql?

URL: <https://aws.amazon.com/what-is/sql/>

Simplilearn (2023). Top 5 mongodb tools for 2024.

URL: <https://www.simplilearn.com/top-mangodb-tools-article>