Big Data MongoDB Supplementary Document

Eve McAleer

Big Data and Linked Data Management

28th March 20204

Table of Contents

[Domain Problem 4](#_Toc162531324)

[Technical Aspects 6](#_Toc162531325)

[Optimisation 11](#_Toc162531326)

[Results 12](#_Toc162531327)

[Appendix A 13](#_Toc162531328)

[Appendix B 14](#_Toc162531329)

[Appendix C 15](#_Toc162531330)

[Appendix D 16](#_Toc162531331)

[Appendix E 17](#_Toc162531332)

[Appendix F 18](#_Toc162531333)

[Appendix G 19](#_Toc162531334)

[Figure 1: Image of replacement orders collection produced by this code 1](#_Toc162436675)

[Figure 2: Image of details of each replacemnt order 2](#_Toc162436676)

[Figure 3: Main Code Function 9](#_Toc162436677)

[Figure 4: Function to query sales collection 10](#_Toc162436678)

[Figure 5: Function to query products collection 11](#_Toc162436679)

[Figure 6: Function to generate replacement orders 12](#_Toc162436680)

[Figure 7: Function to mark sales as processed 13](#_Toc162436681)

[Figure 8: Function to add indexes and optimise queries 14](#_Toc162436682)

[Figure 9: Function to bulk update the replacement orders 15](#_Toc162436683)

# Domain Problem

**Q. What is the best way to update existing replacement orders? Modify**

**existing ones, or merge them as a separate process?**

* The code I have provided bulk updates existing orders every 28 days based on latest sales data. This modifies existing replacement orders to reflect only the relevant sales i.e. those that are yet unprocessed within the last 28 days.
* This is efficient as the process is simplified by modifying rather than merging, as merging can involve additional steps (aggregation, comparison etc).
* Modifying also aids in consistency, as it reduces the error risk that comes with merging and potential overlap of orders.

*See Appendix G*

**Q. How is your process going to run? Is it going to be a singular process,**

**a series of concurrent equivalent workers, or are there different agents**

**with different tasks running simultaneously?**

* The process is singular – seen in the ‘main()’ function. This is easier to run and maintain than multiple concurrent operations. There is no need to manage communication/synchronization between processes. This also aids in simpler debugging.
* Could potentially be parallelized with multiple concurrent workers, this may improve performance. For example, generating replacement orders for each provider in parallel.

*See Appendices A, F, G*

**Q. What could go wrong with the way your process runs? What are the**

**consequences of it? Is it an acceptable risk? Why?**

* Process could face issues with concurrency if multiple instances tried to update the SAME orders simultaneously.
* Performance bottlenecks could arise if the volume of sales/replacement orders got too high. This can be mitigated by the process of optimisation through indexing and batch processing.

*See Appendices F and G*

# Technical Aspects

**Q. Which sales are you going to target each time the process is run? How**

**are you going to query the database to obtain such sales?**

* Targeting sales, grouped by provider, in the past 28 days. This means machines are restocked monthly.

*See Appendix D*

**Q. How does this choice relate to the problem being solved?**

* Targeting sales within the last 28 days ensures replacement orders are generated based on recent sales data.
* This solves the business problem as ALL providers will be restocking their drinks monthly.

**Q. How does this choice relate to effective horizontal scalability of**

**the process?**

* Bulk operations reduce the overhead of individual write operations.
* Distributing workload across multiple (monthly) instances of the process achieves horizontal scalability.

*See Appendix G*

**Q. How does this choice relate to the correctness of the results when**

**running in a concurrent manner?**

* Targeting specific sales data (i.e. from the last 28 days) , and those that haven’t been processed before, ensures the correctness of the replacement orders.
* By filtering out already processed sales, duplication is avoided, and only relevant sales contribute to the orders.

*See Appendix E*

**Q. How does this choice relate to the isolation of the process with**

**respect to other processes running concurrently?**

* This process operates within the boundaries of transactions with the ACID properties (atomicity, consistency, isolation, and durability).
* Atomicity – MongoDB’s support of atomic operations (updates, inserts etc) ensures that an operation will be completed in entirety or will have no effect on the data if it fails. This means other concurrent processes will not observe intermediate states during the operation, as partial updates which could lead to inconsistency are prevented.
* Consistency – This process maintains consistency by ensuring operations are performed with the defined business rules and constraints. E.g. only relevant data is considered.
* Isolation – MongoDB provides document-level isolation, meaning this process’s operations are isolated from other concurrent operations. This isolation ensures transactions occur independently.
* Durability – MongoDB ensures data durability by persisting changes to disk and replicating data across nodes in a sharded cluster or replica set.

**Q. How does this choice relate to the continuous insertion of new**

**sales in the database?**

* This process is unaffected by continuous insertion of new sales, as the process is repeated every 28 days, and targets only those sales input in the past 28 days.
* By processing via timestamps, the process remains responsive to additional sales.
* Any new sales will be processed and reordered in subsequent runs.

**Q. How does this choice relate to the potential sharding and replication of the database?**

* Marking sales as processed for replacement, and targeting only those that haven’t been processed yet relates to the potential sharding/replication of the database by ensuring data consistency and integrity.
* Maintaining consistency across shards and replicas is crucial.
* By persisting the state of processed sales within each document, the process ensure consistency across shards and replicas, enhancing data reliability in a distributed environment.

**Q. How are you going to address the concerns with the correctness of the**

**result in the way your process is run?**

* Concerns regarding correctness are addressed by targeting sales every 28 days and ensuring that each sale is only processed ONCE for replacement.
* By querying the database with precise criteria and marking sales as processed after they are counted towards total sales, the process ensures replacement orders are accurate and void of duplications.

**Q. Do you need to make sure the same sale is not targeted more**

**than once? If so, how are you going to do it?**

* See marked\_as\_processed() function.
* Added ‘processed’ field to sales data to ensure no duplication when counting towards total sales.
* This is also aided in the check for last 28 days, meaning not ALL sales in the collection are being targeted every time.

*See Appendix E*

**Q. Do you need to persist a state in any of the existing documents**

**that is not persisted now? What level of transactionality do you**

**need for handling that state?**

* The state of whether a sale has been processed needs to be persisted, so that separate runs of the program don’t duplicate the number of sales, thus the number of replacement orders.
* A moderate level of transactionality is required to ensure data integrity and prevent race conditions when updating the state of multiple documents concurrently. Race conditions can lead to unexpected/incorrect results if multiple operations try to modify the same data at the same time.
* Referenced and explained in previous question, regarding atomic operations and transactionality.

*See Appendix E*

**Q. Is this process efficient?**

* Yes, this process is efficient ( as explained previously)
* Optimises database queries.
* Minimizes redundant operations.
* Leverages bulk write operations.
* Continuous monitoring and optimisation will be required to maintain optimal performance as data volume grows.

# Optimisation

**Bulk Updates:**

*See Appendix F*

* Reduces volume of individual write operations.
* Reduces Network traffic, particularly beneficial in distributed environments.
* Optimises disk I/O operations, leading to better disk utilisation, and overall improved performance.

**Indexing:**

*See Appendix F*

* Creates indexes on field such as ‘provider’ to improve query performance and enable faster retrieval.
* Also improves disk I/O performance as the indexes can be used to locate relevant documents more quickly.

**Query Optimisations:**

*See Appendix B*

* Filtering already processed sales and targeting sales in the last 28 days reduces data volume. This leads to faster query execution time, especially in databases with a high data volume.

# Results

Below are snippets from the current replacement orders collection, as generated by my code. Orders are grouped by provider and show the order period i.e. month in which these orders are being placed. The orders are then grouped by product name, and show the amount being ordered, which is the exact total of what has been sold in the last 28 days.

A screenshot of a computer

Description automatically generated

Figure 1: Image of replacement orders collection produced by this code

A screenshot of a computer

Description automatically generated

Figure 2: Image of details of each replacemnt order

# Appendix A

A screenshot of a computer program

Description automatically generated

Figure 3: Main Code Function

# Appendix B

A computer screen shot of a program code

Description automatically generated

Figure 4: Function to query sales collection

# Appendix C



Figure 5: Function to query products collection

# Appendix D

A screen shot of a computer program

Description automatically generated

Figure 6: Function to generate replacement orders

# Appendix E

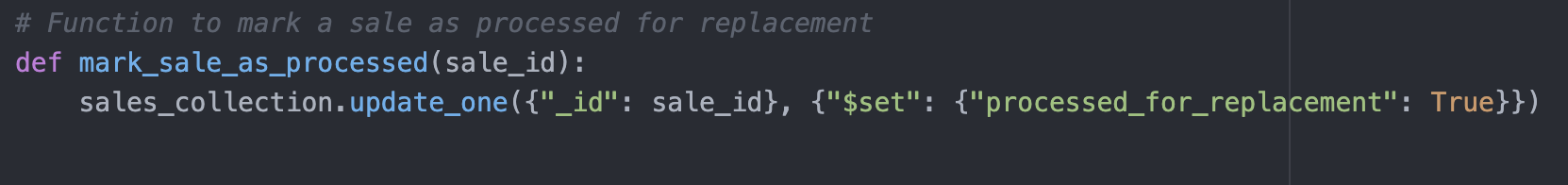


Figure 7: Function to mark sales as processed

# Appendix F

A screen shot of a computer code

Description automatically generated

Figure 8: Function to add indexes and optimise queries

# Appendix G

A screen shot of a computer program

Description automatically generated

Figure 9: Function to bulk update the replacement orders