Performance Analysis of MySQL vs Mongodb using Medicaid Drug Rebate Program data

Sanjayram Raja Srinivasan
Department of Data Science
The George Washington University
Washington DC, USA
sanjayram.rajasrinivasan@gwu.edu

Nadadhur Srivallabh Siddharth
Department of Data Science
The George Washington University
Washington DC, USA
srivallabhsiddharth.nadadhur@gwu.edu

Manoj Padala
Department of Data Science
The George Washington University
Washington DC, USA
manoj.padala@gwmail.gwu.edu

Abstract—**As the landscape of healthcare data expands, the necessity to efficiently manage and retrieve substantial volumes of information becomes paramount. This study delves into the comparative analysis of two prominent database management systems, MySQL and MongoDB, focusing on their performance using a dataset derived from the Medicaid Drug Rebate Program via US data.gov.**

Our investigation centers on a comprehensive dataset encompassing active drugs reported by various drug manufacturers. These pharmaceutical entries are identified by their National Drug Code (NDC), unit specifics, FDA approval and market entry dates, innovator classification, prescription availability, FDA therapeutic equivalency coding, DESI ratings, and termination dates. Each quarterly snapshot serves as a static representation within the system, unaltered by subsequent updates.Through rigorous experimentation and performance evaluations, this paper scrutinizes the efficiency and retrieval efficacy of MySQL and MongoDB when handling this intricate medical dataset.

Keywords*—*Mysql,Mongodb,Performance Analysis,Medical Drug Rebate Program, NDC, Big data.

I. Introduction

The primary objective of this study is to delve into a comprehensive performance comparison between MySQL and MongoDB, honing in on their retrieval efficiency and efficacy in managing the nuances of an intricate medical dataset. Our focus lies in understanding how these database systems navigate the complexities inherent in healthcare-related data, thereby shedding light on their operational strengths and weaknesses.

By subjecting MySQL and MongoDB to a battery of rigorous experimentation and meticulous performance evaluations, we endeavor to uncover the intricacies of their capabilities in handling the multifaceted nature of healthcare datasets. The specific dataset procured from the Medicaid Drug Rebate Program via US data.gov serves as an ideal testbed, comprising an extensive repository of active drug records meticulously cataloged by various parameters.In managing healthcare data, deciding between relational (MySQL) and non-relational (MongoDB) databases is crucial. Healthcare data often comes in complex structures that need careful handling. MySQL's organized structure suits well with established data connections and structured queries, reflecting the usual way healthcare systems handle data.

This research aims to yield valuable insights into the comparative strengths and limitations of MySQL and MongoDB within the realm of complex healthcare datasets. The comprehensive analysis of their performance using the Medicaid Drug Rebate Program dataset intends to offer actionable guidance to healthcare professionals and database administrators seeking optimal solutions for managing complex healthcare repositories.

II. Related Works

Sahatqija et al. [1] The goal of this study is to conduct a comparative analysis between relational and NoSQL databases, delving into the fundamental features and characteristics that define their application in data management.Focused on principles inherent in conventional DBMS, the research accentuates the importance of data integrity and transaction consistency upheld by relational databases.This study notably addresses the burgeoning challenge posed by the rapid expansion of data and the limited support from traditional databases in coping with this influx. It explores the emergence of NoSQL databases, offering an alternative paradigm to address the evolving data management landscape. By conducting a meticulous qualitative analysis, Sahatqija and colleagues delve into the nuanced advantages and limitations of both relational and NoSQL databases, offering insights into their respective capacities in data creation, retrieval, update, and management.The qualitative research methodology employed in this study involves an in-depth examination of recent publications, aiming to provide a comprehensive understanding of the distinctive features characterizing relational and NoSQL databases.

Filip and Čegan et al. [2] provides a comprehensive exploration of NoSQL databases, delineating the diverse data structures pivotal in influencing database performance and system dependencies. Their study encompasses an in-depth overview and comparison of four types of NoSQL databases, elucidating the nuanced differences that render each database more suited for specific application contexts. Additionally, the paper delves into a benchmarking analysis, focusing on the comparative performance evaluation between MongoDB and MySQL in various scenarios. By examining critical operations such as data insertion, updating, and deletion, with and without transactions, the study offers insights into the operational efficiencies of these databases. Notably, the research evaluates the added value of indexed fields in different scenarios, shedding light on MongoDB and MySQL's comparative advantages and limitations. Employing a rigorous methodology, this analysis aims to provide nuanced insights into the performance disparities between MongoDB and MySQL, contributing to a deeper understanding of their operational efficiencies and cost implications.

Yassine and Awad et al. [3] compare the complexities of transitioning from relational databases (SQL) to NoSQL structures, addressing the demands of big data and data analytics. Recognizing the inherent challenges—such as the absence of an automated transformation process and the imperative of ensuring optimal performance and accurate data representation, the paper focuses on evaluating common mapping strategies from SQL to NoSQL structures.Employing MySQL for SQL structures and MongoDB for NoSQL, the study rigorously compares these strategies, primarily gauging retrieval time to identify the most effective approaches. Notably, the research highlights promising outcomes from a hybrid method, amalgamating one-level embedded documents with reference relationships, showcasing.Through meticulous experimentation and analysis, the paper extends valuable insights into practical and effective mapping strategies for practitioners grappling with the complexities of database paradigm transitions.

Deari et al. [4] provide an extensive analysis and comparison between document-based and relational databases, spotlighting MongoDB and MySQL as the focal models for evaluation. Their study meticulously examines and appraises the fundamental principles underpinning data storage and management within each database type. Additionally, the research undertakes the evaluation of CRUD operations (Create, Read, Update, Delete) across various scenarios on both MongoDB and MySQL. Through this operational dissection, the study aims to extract valuable insights into the specific strengths and limitations inherent in each database model.Additionally,the investigation aims to outline the unique attributes characterizing each database type, particularly emphasizing scenarios wherein the document-based NoSQL approach emerges as a practical solution. Through this comparative exploration, Deari and team aim to furnish a nuanced comprehension of the operational contrasts distinguishing document-based from relational databases. Ultimately, this endeavor seeks to provide professionals with a clearer understanding of the contexts where adopting document-based NoSQL databases proves advantageous. Ultimately, this comparative analysis seeks to offer a comprehensive understanding of the operational intricacies between document-based and relational databases, serving as a guiding resource for practitioners navigating the complexities of database model selection and implementation in real-world scenarios.

Ha and Shichkina et al. [5] introduce a systematic approach to query translation from MySQL to MongoDB, emphasizing consideration for database structure across four distinct phases. Beginning with parsing MySQL queries to establish syntax grammar, the methodology proceeds to create a query parts dictionary aligned with MongoDB's aggregation structure. Notably, the authors meticulously define the MongoDB database structure by transforming collection data into a hierarchical structural tree, ensuring alignment between query parts and the target database structure. The final phase synthesizes MongoDB database queries from the assembled dictionary. Through empirical testing of diverse queries, this study serves as a robust validation of their approach, showcasing its practicality and efficacy in facilitating seamless query translation between MySQL and MongoDB databases. This methodological depth and empirical validation provide practitioners with a valuable framework for navigating database migration challenges, facilitating a smoother transition between disparate database structures.

III. Existing System

In the assessment of MySQL and MongoDB efficiency with complex medical datasets, the study highlighted their operational efficiency. However, notable oversights impacted the analysis. Specifically, there was no direct comparison of execution time between MySQLWorkbench and Python SQL connector. This omission restricts a holistic evaluation of the overall system's performance. Furthermore, the analysis didn't delve into the comparison of querying methods—direct MongoDB queries versus Python-based querying. These gaps hinder comprehensive insights into both system performance and querying efficiencies.

IV. Proposed System

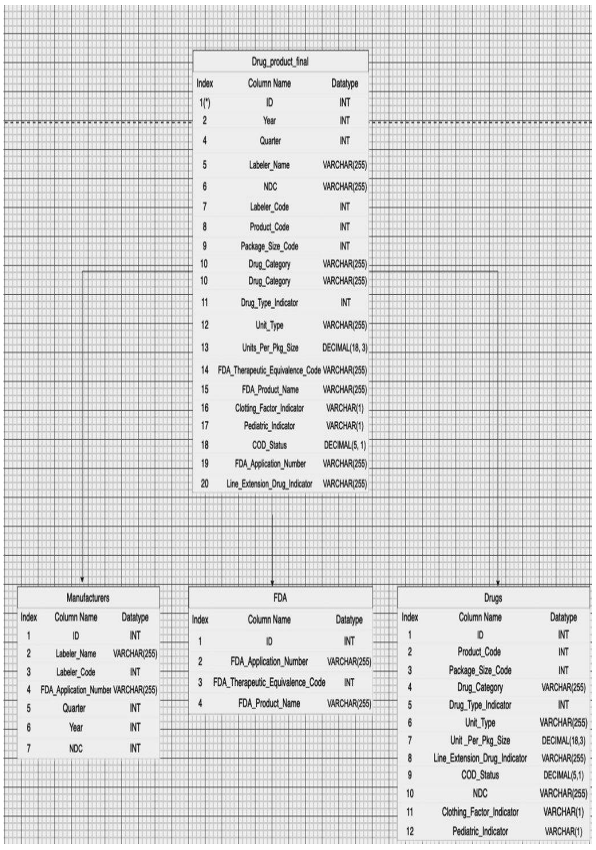
The proposed system targets refining hierarchical models in MongoDB and MySQL databases, specifically focusing on parent-child table relationships. Its goal is to improve data retrieval, storage, and query execution efficiency within these distinct database setups. The system's key objectives involve a thorough performance assessment, comparing operations between parent and child tables in both MongoDB and MySQL. This assessment aims to identify performance variations, highlight bottlenecks, and discover unique efficiencies in hierarchical relationships across these diverse database systems. Through methodical use of performance metrics, query executions, and controlled experiments, the proposed system aims to uncover optimization opportunities, streamlining hierarchical data management in MongoDB's document-based architecture and MySQL's structured relational model.

V. Proposed Methodology

This section compares and evaluates the performance metrics of MySQL and MongoDB databases for a range of operations. Testing for data retrieval, insertion, update, deletion, and query executions in both MySQL and MongoDB installations will be part of the assessment. In order to determine the advantages and disadvantages of each database system in terms of speed, scalability, and efficiency, the performance metrics will be measured and compared. The objective is to extract knowledge about each database's appropriateness for various use cases and workloads.

A. Hierarchical Relations between Tables in SQL

Within the MySQL database schema provided, the tables showcase a hierarchical structure through parent-child relationships, establishing an interconnected framework of data. The central entity, the "Drug_product_final" table, acts as the cornerstone containing crucial information about various drug products. Its primary key, the "ID" column, serves as the linking factor with other associated tables.



Drug_product_final

Index	Column Name	Datatype
1(*)	ID	INT
2	Year	INT
4	Quarter	INT
5	Labeler_Name	VARCHAR(255)
6	NDC	VARCHAR(255)
7	Labeler_Code	INT
8	Product_Code	INT
9	Package_Size_Code	INT
10	Drug_Category	VARCHAR(255)
10	Drug_Category	VARCHAR(255)
11	Drug_Type_Indicator	INT
12	Unit_Type	VARCHAR(255)
13	Units_Per_Pkg_Size	DECIMAL(18, 3)
14	FDA_Therapeutic_Equivalence_Code	VARCHAR(255)
15	FDA_Product_Name	VARCHAR(255)
16	Clotting_Factor_Indicator	VARCHAR(1)
17	Pediatric_Indicator	VARCHAR(1)
18	COD_Status	DECIMAL(5, 1)
19	FDA_Application_Number	VARCHAR(255)
20	Line_Extension_Drug_Indicator	VARCHAR(255)

Manufacturers

Index	Column Name	Datatype
1	ID	INT
2	Labeler_Name	VARCHAR(255)
3	Labeler_Code	INT
4	FDA_Application_Number	VARCHAR(255)
5	Quarter	INT
6	Year	INT
7	NDC	INT

FDA

Index	Column Name	Datatype
1	ID	INT
2	FDA_Application_Number	VARCHAR(255)
3	FDA_Therapeutic_Equivalence_Code	INT
4	FDA_Product_Name	VARCHAR(255)

Drugs

Index	Column Name	Datatype
1	ID	INT
2	Product_Code	INT
3	Package_Size_Code	INT
4	Drug_Category	VARCHAR(255)
5	Drug_Type_Indicator	INT
6	Unit_Type	VARCHAR(255)
7	Unit _Per_Pkg_Size	DECIMAL(18,3)
8	Line_Extension_Drug_Indicator	VARCHAR(255)
9	COD_Status	DECIMAL(5,1)
10	NDC	VARCHAR(255)
11	Clothing_Factor_Indicator	VARCHAR(1)
12	Pediatric_Indicator	VARCHAR(1)

Visual Representation of Hierarchical Table Relationships

The "Manufacturers" table forms a parent-child relationship with the "Drug_product_final" table, establishing connections based on the "ID" foreign key. This table houses specific details about manufacturers associated with individual drug products. Each entry in the "Manufacturers" table corresponds to a manufacturer linked to a particular drug product within the central entity.And the visual representation gives us an clear indication of the hierarchy.Similarly, the "FDA" table maintains a parent-child association with the "Drug_product_final" table through the shared key "ID."It contains pertinent information regarding FDA applications, therapeutic equivalence codes, and product names linked directly to specific drug products listed in the central entity.Additionally, the "Drugs" table exhibits a parent-child relationship with the "Drug_product_final" table, utilizing the common key "ID." It encapsulates comprehensive drug-related data, including product codes, package sizes, drug categories, and indicators, intricately linked to individual drug products listed in the central entity.These tables interconnect via foreign key constraints that reference the "ID" column in the "Drug_product_final" table, ensuring data integrity and enforcing the hierarchical structure. These constraints enable only valid "ID" values from the central entity to be associated with respective entries in the child tables.The established parent-child relationships across these tables create a hierarchical structure, enabling comprehensive data organization and facilitating efficient retrieval of interconnected information. This hierarchy allows for in-depth analysis and seamless access to related data elements, such as manufacturers, FDA-related specifics, and drug-specific details, enriching the overall database schema.

Understanding and leveraging these hierarchical relationships within the MySQL database schema is instrumental in executing complex queries, performing joins, and obtaining comprehensive insights into drug products and their associated details.

B. Analyzing Queries in SQL

1)Creating Parent Table (DDL)

This query is a directive to create a table named "Drug_product_final" within the database.
(We have used python using SQL connector to create tables and load the data)
Creating loading 1 million rows of data in mysql workbench was not working

2)Find Manufacturers with the Highest Number of Products

Parent Table Query: Counts the number of products per manufacturer from the "Drug_product_final" table.

Child Table Query: Joins the "Manufacturers" and "Drugs" tables to count the products per manufacturer, showcasing the flexibility of querying via parent-child relationships.

Query:
```
#Using parent table:
SELECT Labeler_Name, COUNT(Product_Code) AS
Total_Products
FROM Drug_product_final
GROUP BY Labeler_Name
ORDER BY Total_Products DESC;

#Using child table:
SELECT M.Labeler_Name, COUNT(D.Product_Code) AS
Total_Products
FROM Manufacturers M
LEFT JOIN Drugs D ON M.ID = D.ID
GROUP BY M.Labeler_Name
ORDER BY Total_Products DESC;
```

3)Retrieve the count of FDA application numbers associated with multiple product numbers. Display records where the count of associated product numbers for an FDA application number is greater than 1.

These queries count FDA application numbers associated with multiple products, utilizing both parent and child tables for comparison.

Query:
#Using parent table:
SELECT FDA_Application_Number, COUNT(DISTINCT Product_Code) AS Associated_Product_Count
FROM Drug_product_final
GROUP BY FDA_Application_Number
HAVING COUNT(DISTINCT Product_Code) > 1
Order by Associated_Product_Count DESC;

#Using child table:
SELECT f.FDA_Application_Number, COUNT(DISTINCT d.Product_Code) AS Associated_Product_Count
FROM FDA f
JOIN Drugs d ON f.ID = d.ID
GROUP BY f.FDA_Application_Number
HAVING COUNT(DISTINCT d.Product_Code) > 1
Order by Associated_Product_Count DESC;

4)Find FDA Application Numbers Associated with Multiple Labelers:

These queries find FDA application numbers associated with multiple labelers, employing both direct and joined table methods.

Query:
Using parent table:
SELECT dp.FDA_Application_Number,
COUNT(DISTINCT dp.Labeler_Code) AS Associated_Labelers
FROM Drug_product_final dp
GROUP BY dp.FDA_Application_Number
HAVING COUNT(DISTINCT dp.Labeler_Code) > 1
Order by Associated_Labelers DESC;

#Using child table:
SELECT f.FDA_Application_Number, COUNT(DISTINCT m.Labeler_Code) AS Associated_Labelers
FROM FDA f
JOIN Manufacturers m ON f.ID = m.ID
GROUP BY f.FDA_Application_Number
HAVING COUNT(DISTINCT m.Labeler_Code) > 1
Order by Associated_Labelers DESC;

5)Counting Products by Manufacturer for a Specific Drug Category

Counts the number of products by manufacturer for a specified drug category, showcasing the utilization of filters in both parent and child tables to retrieve specific subsets of data.

Query:
#Using parent table:
SELECT dp.Labeler_Name, COUNT(dp.Product_Code) AS Product_Count
FROM Drug_product_final dp
WHERE dp.Drug_Category = 'S'
GROUP BY dp.Labeler_Name
Order by Product_Count DESC;

Using child table:
SELECT m.Labeler_Name, COUNT(d.Product_Code) AS Product_Count
FROM Manufacturers m
JOIN Drugs d ON m.ID = d.ID
WHERE d.Drug_Category = 'S'
GROUP BY m.Labeler_Name
Order by Product_Count DESC;

6)Show only those manufacturers with more than 5 products and more than 3 FDA applications.

Filters manufacturers with specific product and FDA application counts, demonstrating the usage of conditions for data retrieval.

Query:
Using parent table:
SELECT Labeler_Name, COUNT(Product_Code) AS Product_Count, COUNT(FDA_Application_Number) AS FDA_Count
FROM Drug_product_final
GROUP BY Labeler_Name
HAVING Product_Count > 5 AND FDA_Count > 3
Order by Product_Count Desc;

Using child table:
SET SESSION profiling = 10;
SELECT m.Labeler_Name, COUNT(d.Product_Code) AS Product_Count, COUNT(f.FDA_Application_Number) AS FDA_Count
FROM Manufacturers m
LEFT JOIN Drugs d ON m.ID = d.ID
LEFT JOIN FDA f ON m.ID = f.ID
GROUP BY m.Labeler_Name
HAVING Product_Count > 5 AND FDA_Count > 3
Order by Product_Count DESC;

7)Generate a result set that combines the counts for specific drug categories, manufacturers for a drug type indicator, and product counts by year and quarter

Creates a combined result set with counts based on multiple criteria, leveraging groupings and counting across various dimensions.It delves into intricate relationships within the hierarchical structure, examining product distributions, FDA application associations,manufacturer-product linkages.This approach facilitates a overall perspective, enabling a deeper analysis of the interconnected entities and their distributions within the database, thereby supporting comprehensive data-driven decisions.
Query:
#Using parent table
```
SELECT
   CASE
        WHEN Drug_Category IN ('S', 'I') THEN
Drug_Category
       ELSE NULL
```

```
    END AS Drug_Category,
    COUNT(*) AS Category_Count,
    Drug_Type_Indicator,
    COUNT(DISTINCT Labeler_Name) AS
Manufacturer_Count,
    Year,
    Quarter,
    COUNT(*) AS Product_Count
FROM Drug_product_final
GROUP BY Drug_Category, Drug_Type_Indicator, Year,
Quarter
ORDER BY Drug_Category, Drug_Type_Indicator, Year,
Quarter;

#Using child Table
SELECT
    d.Drug_Category AS Drug_Category,
    COUNT(*) AS Category_Count,
    d.Drug_Type_Indicator AS Drug_Type_Indicator,
    COUNT(DISTINCT m.Labeler_Name) AS
Manufacturer_Count,
    m.Year AS Year,
    m.Quarter AS Quarter,
    COUNT(*) AS Product_Count
FROM Drugs d
JOIN Manufacturers m ON d.ID = m.ID
GROUP BY d.Drug_Category, d.Drug_Type_Indicator,
m.Year, m.Quarter
ORDER BY d.Drug_Category, d.Drug_Type_Indicator,
m.Year, m.Quarter;
```

8) Calculate the counts of drugs, manufacturers, and distinct FDA application numbers associated with each drug category based on the shared 'ID'

Computes counts for drugs, manufacturers, and distinct FDA applications categorized by drug type, illustrating an analysis based on common identifiers.Conversely, the child table query intertwines data from the 'Drugs', 'Manufacturers', and 'FDA' tables. It aligns the drug categories and computes the total drug count, distinct manufacturer count (based on unique identifiers), and distinct FDA application counts, offering a comprehensive analysis mirroring the parent table query but from a combined perspective.

Query:

```
#Using parent table:
SELECT
    Drug_Category,
    COUNT(*) AS Drug_Count,
    COUNT(DISTINCT Labeler_Code) AS
Manufacturer_Count,
    COUNT(DISTINCT FDA_Application_Number) AS
FDA_Application_Count
FROM Drug_product_final
GROUP BY Drug_Category;

#Using child table:
SELECT
    d.Drug_Category AS Drug_Category,
    COUNT(*) AS Drug_Count,
    COUNT(DISTINCT m.ID) AS Manufacturer_Count,
    COUNT(DISTINCT f.ID) AS FDA_Application_Count
FROM Drugs d
LEFT JOIN Manufacturers m ON d.ID = m.ID
LEFT JOIN FDA f ON d.ID = f.ID
GROUP BY d.Drug_Category;
```

9) This query filters the Manufacturers table to include only entries for the year 2023 and the 3rd quarter, then performs the counts for each drug category based on these filtered Manufacturers along with the Drugs and FDA tables.

Filters data based on specific time criteria and counts drug categories, demonstrating data retrieval and filtering with joins.

```
#Using parent table:
SELECT
    Drug_Category,
    COUNT(*) AS Drug_Count,
    COUNT(DISTINCT Labeler_Code) AS
Manufacturer_Count,
    COUNT(DISTINCT FDA_Application_Number) AS
FDA_Application_Count
FROM Drug_product_final
WHERE Year = 2023 AND Quarter = 3
GROUP BY Drug_Category;

#Using child table:
SELECT
    d.Drug_Category AS Drug_Category,
    COUNT(*) AS Drug_Count,
    COUNT(DISTINCT m.ID) AS Manufacturer_Count,
    COUNT(DISTINCT f.ID) AS FDA_Application_Count
FROM Drugs d
LEFT JOIN Manufacturers m ON d.ID = m.ID
LEFT JOIN FDA f ON d.ID = f.ID
WHERE m.Year = 2023 AND m.Quarter = 3
GROUP BY d.Drug_Category;
```

10) Update Labeler Name from 'ELI LILLY AND COMPANY' to 'Eli' (DDL)

Updates the labeler name in both the parent and child tables, showcasing the update operation's application. Employing the UPDATE statement, the query initiates modifications within the 'Labeler_Name' field,Hierarchical Relations between Tables in MySQL This adjustment directly influences the parent table, effecting changes uniformly across all entries that satisfy the condition stipulated by 'Labeler_Name = 'ELI LILLY AND COMPANY''.

Query:

```
#Using parent table:
UPDATE Drug_product_final
SET Labeler_Name = 'Eli'
WHERE Labeler_Name = 'ELI LILLY AND COMPANY';
```

```
#Using child table:
UPDATE Manufacturers
SET Labeler_Name = 'Eli'
WHERE Labeler_Name = 'ELI LILLY AND COMPANY';
```

11)Drop tables (DML):

The provided sequence of 'DROP TABLE' commands facilitates the removal of specific tables from the database, effectively removing them from the schema. Each 'DROP TABLE' query is tailored to delete a distinct table within the database structure. The attempted deletion starts with the table named 'Drug_product_final1,' although the syntax seems to contain an error with the inclusion of 'table_name' before the actual table name. The subsequent queries then aim to eliminate the 'Drugs,' 'FDA,' and 'Manufacturers' tables.

Query:

```
DROP TABLE table_name Drug_product_final1;
Drop table Drugs;
Drop table FDA;
Drop table Manufacturers;
```

C. Executing Queries in MySQL Workbench vs. Connecting to SQL Server Using Python

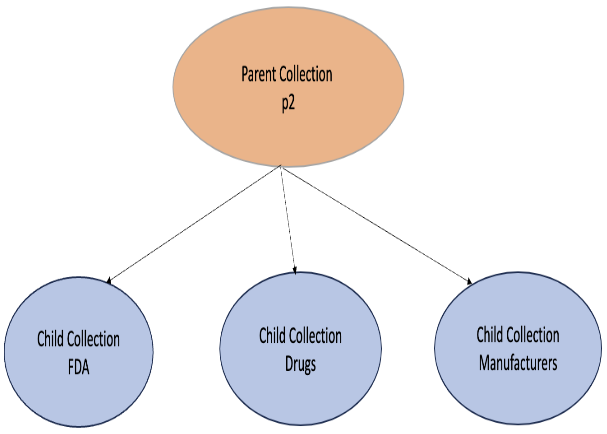
The paper incorporates a comprehensive comparison that scrutinizes the execution of queries in two distinct environments: MySQL Workbench and the execution of queries via Python connecting to a SQL Server. This analysis delves into performance metrics and efficiency measurements, aiming to showcase the disparities and operational nuances between these execution environments. The comparison between MySQL Workbench and Python-based query execution connecting to a SQL Server encompasses an exploration of diverse execution environments, each offering unique functionalities. Evaluating performance metrics and efficiency measurements allows for a comprehensive understanding of their operational disparities and nuances.This exploration sheds light on the varied strengths and operational nuances of these environments, highlighting their respective advantages and potential limitations. This analysis aims not only to highlight the differences in query execution speed or data retrieval but also to uncover the strengths and limitations inherent in each environment.By accessing these diverse execution environments, a nuanced understanding emerges, revealing the distinct operational nuances and highlighting areas of excellence or potential improvement within each system.This comparative examination intends to provide actionable guidance for professionals navigating the diverse landscape of database interaction, aiding in informed decision-making for efficient query execution and database management strategies. The forthcoming Performance Analysis section of the paper will detail findings and valuable insights. Looking beyond surface functionalities, it entails an in-depth analysis of performance metrics and operational efficiency. By scrutinizing these metrics closely, a holistic understanding emerges, revealing the nuanced differences and potential synergies within their execution environments

D. Analyzing the Queries in MongoDB

1)Difference in Terminologies between SQL and MongoDB

SQL Concept	MongoDB Equivalent
Database	Database
Table	Collection
Row	Document
Column	Field
Primary Key	_id Field
Joins	Embedded Documents or $lookup

E. Hierarchical Relations between Tables in MySQL



Visual Representation of Hierarchical Relationships in the Collections used in MongoDB

The architectural diagram presents a hierarchical relationship within MongoDB collections, showcasing a structured arrangement akin to parent-child relationships in traditional databases. The central focus is the parent collection, p2, visibly linked to multiple child collections: FDA, Drugs, and Manufacturers. This representation implies a hierarchical organization where the parent collection likely stores broader or generalized data, while the child collections likely contain specific and detailed information corresponding to entries in the parent collection.

Each arrow connecting the parent and child collections signifies relationships and associations between the data. This interconnected structure suggests that entries within the child collections and the parent collection.. It showcases a relational structure where the data across these collections are interrelated and organized in a way that supports logical connections and contextual associations.

F.Comparison of the Performance by executing the same Queries in MongoDB

The central focus of this investigation rests on the parent collection, p2, within the MongoDB database, distinctly linked to multiple child collections: FDA, Drugs, and Manufacturers. This comprehensive comparative analysis aimed to evaluate MongoDB's performance against MySQL in handling diverse data extraction scenarios and intricate querying techniques.

Initiating the assessment involved the creation of two collections, p2 and project1, within the admin and config databases, respectively. The crucial dataset, final_dataset.csv, underwent initial upload into the p2 collection via MongoDB Compass in the admin database. To ensure a robust evaluation, this dataset was duplicated into the project1 collection within the config database. This duplication allowed for meticulous monitoring and timing of insertion, deletion, and update operations facilitated by Python scripts.

The dataset underwent further segregation into four subsets, each loaded into distinct collections: Drug_product_final, FDA, Drugs, and Manufacturers. This strategic division aimed to delve into and analyze MongoDB's aggregation pipeline performance, especially when leveraging the $lookup operator for merging data across multiple collections.

All queries were meticulously constructed and executed using MongoDB's aggregation pipelines, with rigorous testing conducted in both Compass and a Python editor. This process necessitated the establishment of a seamless connection between MongoDB and Python.Throughout the analysis phase, the execution times for each query were intricately recorded, highlighting the system's responsiveness to diverse data extraction methodologies. Owing to the dataset's substantial size, the actual query results weren't printed; only the execution times were showcased, offering a concise performance measure. However, the code segments to display the query results are available in the respective .py file comments (#).

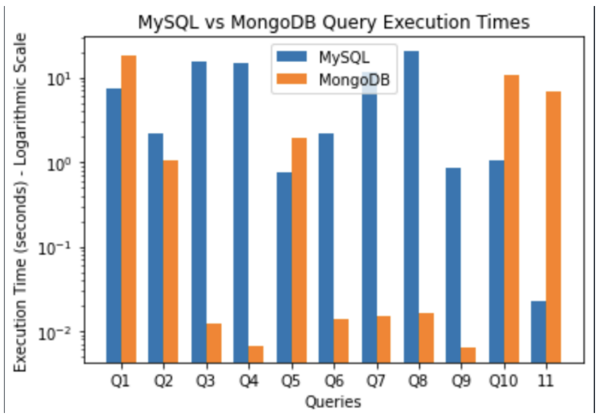
The comprehensive analysis of execution times provides invaluable insights into MongoDB's proficiency in managing extensive datasets and complex query operations. It offers a compelling comparative perspective against traditional SQL databases such as MySQL. The findings derived from this study serve as a pivotal reference for optimizing database performance and making informed choices regarding database selection.

In the forthcoming Performance Analysis section, the culmination of meticulous execution time recordings from diverse querying techniques and data extraction methodologies will be presented. The recorded execution times, meticulously captured and analyzed throughout the evaluation, will be the focal point of this analysis. This section will serve as the cornerstone for illustrating MongoDB's performance prowess, outlining its handling of complex querying scenarios and extensive datasets compared to conventional SQL databases like MySQL.

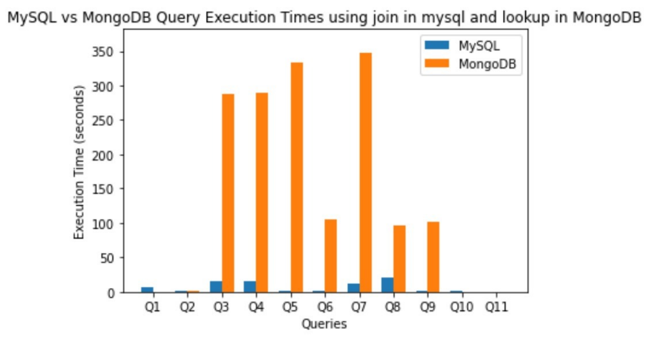
VI. Performance Analysis

	NOSQL (MongoDB)	
Query	Single Collections	Multiple Collections
1	2.4246811867 seconds	1.147345067 seconds
2	3.6467368347 seconds	286.9734880 seconds
3	3.3315622806 seconds	288.9863336 seconds
4	1.9538829326 seconds	332.9899108 seconds
5	3.3854403495 seconds	105.6316227 seconds
6	2.3259390576 seconds	347.2062229 seconds
7	3.9536590576 seconds	96.32964220 seconds
8	2.8050091266 seconds	101.5340096 seconds

	SQL (MySQL)			
	Python		**MySQL Workbench**	
Query	**Parent (in secs)**	**Child (in secs)**	**Parent (in secs)**	**Child (in secs)**
1	7.539	17.93	NA	NA
2	2.144	5.642	2.334	5.605
3	15.643	24.418	13.82	14.135
4	15.008	23.498	13.853	11.994
5	0.748	4.043	0.698	3.076s
6	2.225	9.392	2.377s	6.219
7	11.480	11.489	13.104	12.437
8	20.440	14.565	20.4s	15.27 sec
9	0.838	0.878	0.70s	0.836s
10	1.0416	0.567	0.0011	0.0006
11	0.023	0.79	0.0002	0.0039



The above chart shows the execution performance between parent table in MYSQL and Single Collection in MongoDB.



The above chart shows the execution performance between child tables in MYSQL and Multiple Collection in MongoDB.

The diagrammatic form provides a comparison of MongoDB vs MySQL. These findings draw attention to the minute variations in performance between SQL and MongoDB environments, as well as the impact of collection architecture and tool choice on execution times. MongoDB's efficacy in single collections and the disparities in MySQL Workbench's and Python's performance in SQL contexts highlight the need for tailored approaches to optimize performance across various database systems and tools. These understandings are crucial for targeted optimization strategies that align with specific operational requirements and complexity as well as for making educated judgments.

VII. Conclusion

MySQL showcases exceptional proficiency in managing complex operations, excelling in intricate tasks like joins and aggregations across multiple tables. Its efficiency spans from data loading to maintaining data integrity in complex relational structures. In contrast, MongoDB, adept at simpler queries and NoSQL structures, struggles with intricate joins across collections, impacting its performance. MySQL's strength lies in complex relational tasks, while MongoDB excels in simpler NoSQL operations. The choice between them hinges on application needs, data complexity, and operational intricacies.Its efficiency from data loading to maintaining data integrity within complex setups is commendable. Conversely, MongoDB's proficiency in simpler queries and NoSQL structures highlights its suitability for straightforward operations. However, its limitations with intricate joins across collections affect its performance in such scenarios. Ultimately, the choice between these databases should align with specific application requirements, considering factors like data complexity and operational intricacies to ensure optimal performance and functionality.

VIII. Future Enhancement

Firstly, optimizing performance and functionality stands as a crucial aspect for future enhancements in the discussed database operations. This encompasses a detailed exploration into Performance Optimization, delving deeper into tuning strategies for both MySQL and MongoDB. Actions such as fine-tuning queries, optimizing indexes, and potentially restructuring schemas are essential for streamlining complex operations and reducing execution times.Another critical area for enhancement revolves around Scalability Strategies.

Techniques like sharding or partitioning data across multiple servers or nodes can notably reduce individual resource load, significantly enhancing overall performance. Simultaneously, fortifying data redundancy and backup mechanisms through robust replication strategies becomes imperative. These strategies ensure better fault tolerance, high availability, and expedited recovery processes in case of data loss.

Furthermore, embracing new features through regular updates in both MySQL and MongoDB could yield substantial improvements in overall database operations. Alongside these updates, implementing advanced Caching Mechanisms, fortifying Security Measures, exploring Machine Learning Integration, considering Cloud Integration, enhancing User Interface Improvements, and fostering Research and Collaboration avenues all play pivotal roles in advancing database operations and functionality. Prioritizing these enhancements requires a strategic alignment with the database system's specific goals, constraints, and anticipated changes in data volume and complexity.

References

[1] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi and F. Ismaili, "Comparison between relational and NOSQL databases," 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 2018, pp. 0216-0221, doi: 10.23919/MIPRO.2018.8400041.

[2] P. Filip and L. Čegan, "Comparison of MySQL and MongoDB with focus on performance," 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS), Jakarta, Indonesia, 2020, pp. 184-187, doi: 10.1109/ICIMCIS51567.2020.9354307.

[3] F. Yassine and M. A. Awad, "Migrating from SQL to NOSQL Database: Practices and Analysis," 2018 International Conference on Innovations in Information Technology (IIT), Al Ain, United Arab Emirates, 2018, pp. 58-62, doi: 10.1109/INNOVATIONS.2018.8606019.

[4] R. Deari, X. Zenuni, J. Ajdari, F. Ismaili and B. Raufi, "Analysis And Comparison of Document-Based Databases with Relational Databases: MongoDB vs MySQL," 2018 International Conference on Information Technologies (InfoTech), Varna, Bulgaria, 2018, pp. 1-4, doi: 10.1109/InfoTech.2018.8510719.

[5] M. Ha and Y. Shichkina, "The Query Translation from MySQL to MongoDB Taking into Account the Structure of the Database," 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus), St. Petersburg, Moscow, Russia, 2021, pp. 383-386, doi: 10.1109/ElConRus51938.2021.9396591.