

TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Nijat Salmanov 214136IASM

Selection Criteria of Database Management Systems for Large Datasets

Master's Thesis

Supervisor: Associate Professor Vladimir Viies, Ph.D.

Education of Prof.

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Nijat Salmanov 214136IASM

Suurte andmekogumite andmebaasihaldussüsteemide valikukriteeriumid

Magistritöö

Juhendaja: dotsent Vladimir Viies, Ph.D.

Professori haridus.

Tallinn 2023

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature, and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Nijat Salmanov 214136IASM.

Acknowledgments

The author of this thesis would like to express his kind appreciation to his supervisor Vladimir Viies for their encouragement, responsiveness, guidance, and supervision throughout the whole thesis work. This study was performed in Alpha Control Lab, which is within the Centre for Intelligent Systems, Department of Computer Systems, Tallinn University of Technology. My big appreciation goes to the people working in Alpha Control Lab for making things easier to learn.



MASTER'S THESIS TASK SPECIFICATION

Student name: Nijat Salmanov

Student code: 214136IASM

Topic: Selection Criteria of Database Management Systems for Large Datasets.

Topic

background:

In today's data-driven landscape, choosing the right database management system (DBMS) is essential for handling large datasets effectively and optimizing performance. This research offers insights into DBMS selection criteria within Java Spring Boot applications, particularly through a comparative analysis of ClickHouse and PostgreSQL. This research focuses on four critical aspects: Analytics on Big Data, where we analyze how ClickHouse and PostgreSQL handle extensive datasets in Java Spring Boot applications; Data Dumping and Insertion, evaluating the performance and efficiency of data dumping and insertion processes in ClickHouse and PostgreSQL; Scalability Assessment, where we assess scalability features in ClickHouse and PostgreSQL within Java Spring Boot, with a focus on enhancing scalability; and High Availability Evaluation, involving an investigation into high availability mechanisms in DBMS for Java Spring Boot, including performance metrics and reliability improvements. This research delves into Yelp's public data to tackle vital data analytics challenges, aiming to deliver actionable insights and solutions for improved decision-making in the realm of business and user reviews in our data-centric world.

Supervisor: Associate Professor Vladimir Viies, Ph.D.

Co-supervisor:

Additional specifications:

Issues to be resolved according to the following learning outcomes:

A) System aspects:

This research aims to address critical system challenges related to choosing the appropriate DBMS, specifically ClickHouse and PostgreSQL, for effective handling of extensive datasets within Java Spring Boot applications. It involves evaluating system performance, efficiency, scalability, and high availability mechanisms, with a focus on enhancing overall system capabilities.

B) Software aspects:

In the realm of software, this study endeavors to provide insights into the selection criteria for DBMS within Java Spring Boot applications. It entails a comparative analysis of ClickHouse and PostgreSQL, emphasizing their software-related attributes such as data dumping, insertion, query optimization, and software-driven scalability enhancements. Additionally, it explores high availability mechanisms and offers software-centric solutions for efficient large dataset management.

C) Hardware aspects:

Exceptional conditions:

Student's signature:

Nijat Salmanov

Abstract

In the dynamic landscape of data-driven decision-making, carefully selecting a Database Management System (DBMS) becomes paramount, especially within the intricate framework of Java Spring Boot applications handling extensive datasets. This research thoroughly explores critical facets spanning system, software, and hardware considerations, focusing on two prominent DBMS: ClickHouse and PostgreSQL. The primary objective is to furnish actionable insights for adept management of large datasets, encompassing an evaluation of system performance, software attributes, and underlying hardware dynamics. Utilizing real-world datasets extracted from Yelp, the research employs a methodical and rigorous approach involving benchmarking, experimental studies, and insightful case analyses. The comparative analysis unfolds across multiple dimensions, delivering a nuanced guide tailored for decision-makers and practitioners navigating the intricate realm of database technologies. The outcomes of this research shed light on the distinctive strengths and limitations inherent in ClickHouse and PostgreSQL. Notably, the findings transcend individual capabilities, providing a holistic understanding that is a compass for strategic decision-making in the ever-evolving landscape of data-centric applications.

List of Abbreviations and Terms

Words	Abbreviation
DBMS	Database Management System
ORM	Object-Relational Mapping
SQL	Structured Query Language
OLTP	Online Transaction Processing
OLAP	Online Analytical Processing
ERP	Enterprise Resource Planning
APIs	Interfaces for Application Programmers
DML	Data Manipulation Language
DDL	Data Definition Language
JPA	Java persistence API
REST API	Representational State Transfer Application Programming Interfaces
ACID	Attributes of Atomicity, Consistency, Isolation, Durability
YAML	Yet Another Markup Language
JSON	JavaScript Object Notation
JDBC	Java Database Connectivity
DCP	Database connection pool
HTTP	Hypertext Transfer Protocol
JDBC	Java Database Connectivity
CLI	Command Line Interface
R2DBC	Reactive Relational Database Connectivity Driver
CSV	Comma-Separated Values

Table of Contents

Introduction.....	12
1 Analysis of Database Management Systems for Large Datasets	13
1.1 Types of Database Management Systems (DBMS).....	13
1.2 Comparative Study of ClickHouse and PostgreSQL	14
1.3 Integration of DBMS with Java Spring Boot Applications	18
1.3.1 Integration of ClickHouse with Java Spring Boot Applications	18
1.3.2 Integration of PostgreSQL with Java Spring Boot Applications	19
1.4 Integration of PostgreSQL with Java Spring Boot Applications	19
2 Tools and Components of Database Management Systems	20
2.1 Database Management and Querying.....	22
2.1.1 Relational Database Management System (RDBMS).....	22
2.1.3 Database Engine	23
2.1.4 Query Processor/Optimizer	24
2.2 Database Development and Administration	24
2.2.1 Database Administrator (DBA) Tools	25
2.2.2 Database Design Tools	26
2.2.3 Middleware	27
2.3 Maintenance and Security	28
2.3.1 Backup and Recovery Tools.....	28
2.3.2 Data Security Components	29
2.3.3 Database Monitoring and Tuning Tools.....	31
2.3.4 Transaction Manager	32
2.3.5 Database Connectivity Drivers	33
2.4 Structured Query Language (SQL).....	34
2.4.1 Data Definition Language (DDL).....	35
2.4.2 Data Manipulation Language (DML).....	36
3 Analysis and Findings.....	36
3.1 Experimental System Setup.....	36
3.2 Results	54
3.3 Discussion.....	55
4 Conclusions.....	57

Table of Figures

Figure 1 Backend System Architecture	12
Figure 2 Docker run PostgreSQL	37
Figure 3 Docker run ClickHouse	37
Figure 4 ClickHouse-Client	39
Figure 5 Copying a File to a Docker Container	39
Figure 6 Business Table created	40
Figure 7 Create 'Users' Table in ClickHouse's YelpData	41
Figure 8 Dataset Command	41
Figure 9 JSON Files to Corresponding Tables	42
Figure 10 Uploading Records to PostgreSQL Database.....	43
Figure 11 JSON to CSV Conversion Script.....	44
Figure 12 Conversion Complete	45
Figure 13 JSON to CSV Conversion Script.....	46
Figure 14 JSON to CSV.....	47
Figure 15 Insertion Performance Comparison	51
Figure 16 Insert/Update/Delete Performance Comparison.....	53
Figure 17 Performance Comparison	54

List of Tables

Table 1 Comparison Table of Some Popular Databases..... 16
Table 2 Performance Comparison 51

Introduction

In the present environment dominated by data-driven decision-making, businesses must efficiently handle enormous amounts of information to draw significant insights and make smart decisions. The businesses operating now are navigating a complicated data maze and selecting a Database Management System (DBMS) is a crucial decision. This is especially the case when the DBMS is integrated faultlessly into the intricate architecture of Java Spring Boot applications [1]. The selection of a DBMS emerges as a critical factor, poised to have deep and far-reaching consequences as firms increasingly pivot towards data-centric insights for strategic decision-making. This is because selecting a DBMS may have profound and far-reaching repercussions. In the realm of big data, the efficiency of Database Management Systems (DBMS) directly influences the success of data-centric applications [2]. This thesis investigates two prevalent open-source DBMS, ClickHouse and PostgreSQL, within Java Spring Boot applications, to derive selection criteria based on their adeptness at handling large datasets. The core of this research leverages extensive Yelp datasets to benchmark the databases' capabilities, emphasizing their performance and resilience [3].

By staging a series of experiments, we examine each DBMS's handling of large data volumes across various operations—bulk data insertion, transaction processing, and analytical query execution. These experiments aim to paint a comprehensive picture of ClickHouse, noted for swift data analytics, and PostgreSQL, a stalwart in transactional reliability [4].

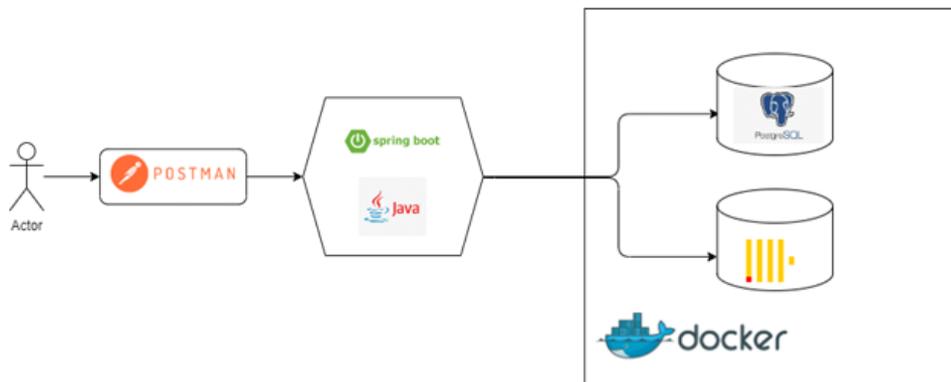


Figure 1 Backend System Architecture

Supplementing this empirical approach, we simulate an actor using Postman to interact with a Java Spring Boot backend—representing typical client-server communication [5]. The backends' responsibility is to facilitate data transactions with the databases, which are orchestrated within Docker containers. This reflects a modern development environment, leveraging Docker's ability to ensure consistency and ease of deployment for open-source databases [6].

The study proceeds with a structured methodology: initializing the databases in Docker, handling data manipulation, and analyzing performance benchmarks. We dissect each DBMS's efficacy in managing large datasets, intending to provide clear guidance for their selection and application in real-world scenarios.

Concisely, this thesis not only assesses the technical aspects of ClickHouse and PostgreSQL but also situates these findings within the broader context of backend development and data management. The outcome is a set of informed recommendations for database selection, poised to aid those architecting systems where data volume and handling efficiency are critical.

1 Analysis of Database Management Systems for Large Datasets

Large datasets, characterized by their size, variety, and the speed at which they are generated, pose unique challenges that traditional database systems may struggle to handle effectively. Yelp, a renowned platform for restaurant reviews and business recommendations, presents a compelling case study in the management of large datasets [7]. Yelp's dataset is extensive, encompassing millions of reviews, user accounts, and business listings. This data is not only massive in volume but also diverse, including text reviews, user ratings, geolocation data, and business attributes. The DBMS must ensure fast query processing, high data integrity, and the ability to scale with the growing data needs of the platform [8].

1.1 Types of Database Management Systems (DBMS)

Relational DBMS: Relational databases, the workhorse of structured data, use tables to store and organize information while also establishing the connections between the various pieces of data [9].

NoSQL DBMS: NoSQL databases, which can adapt to data that is either unstructured or semi-structured, give the flexibility and scalability required to go beyond conventional relational databases [10].

Object-Oriented DBMS: This approach allows for the portrayal of real-world things and the complex interactions between those entities inside the database by providing the data in the form of objects [11].

Graph DBMS: Designed specifically for managing complicated interactions, which are especially common in situations involving network infrastructure or social networks [12].

Columnar DBMS: ClickHouse, as a columnar DBMS, distinguishes itself with a set of features optimized for analytical processing and efficient data retrieval. Utilizing a columnar storage approach, ClickHouse organizes and stores data by columns rather than rows, leading to enhanced compression and accelerated query performance, especially in analytics-focused tasks [13]. The MergeTree engine, a specific storage component within ClickHouse, further contributes to the system's efficiency. Supporting standard SQL queries, ClickHouse ensures familiarity for users accustomed to relational database systems, making it accessible for a broader audience. Scalability is a key strength, enabling horizontal and vertical scaling through partitioning and sharding for efficient data distribution [14]. The DBMS employs robust replication mechanisms to ensure data availability and fault tolerance, while its high compression ratios contribute to reduced storage requirements. ClickHouse excels in scenarios with heavy read-loads, utilizing parallel processing capabilities and batch-oriented approaches for efficient data handling [15]. With materialized views and open-source accessibility, ClickHouse is well-suited for data warehousing environments, making it a compelling choice for analytical tasks and large-scale data analysis [16].

1.2 Comparative Study of ClickHouse and PostgreSQL

ClickHouse is an open-source, columnar-oriented relational database management system (DBMS) designed for high-performance analytical processing of large volumes of data. Developed by Yandex, ClickHouse is renowned for its exceptional speed and efficiency in handling complex analytical queries over extensive datasets [17]. The architecture of ClickHouse is optimized for read-heavy workloads, making it a powerful tool for scenarios where rapid data retrieval is crucial. One distinctive feature of ClickHouse is its columnar storage engine, known as MergeTree, which organizes and stores data in columns rather than rows [18]. This columnar storage approach enhances compression and accelerates query performance, especially for analytics-focused tasks. ClickHouse supports standard SQL queries, making it accessible to users familiar with traditional relational databases [19]. Scalability is a key strength of ClickHouse, supporting both horizontal and vertical scaling to handle growing data requirements. It facilitates partitioning and sharding,

enabling efficient data distribution across multiple nodes. ClickHouse also provides replication mechanisms for ensuring data availability and fault tolerance [20].

While ClickHouse excels in analytical processing, it has limitations in certain aspects. For example, it may not be as well-suited for transactional workloads compared to traditional relational databases [21]. Additionally, ClickHouse lacks some advanced features commonly found in general-purpose databases [22].

PostgreSQL, often referred to as Postgres, is a powerful open-source relational database management system known for its robust features and standards compliance. Developed by the PostgreSQL Global Development Group, Postgres has a broad range of use cases, from small applications to large-scale enterprise systems [23]. As a relational database, PostgreSQL organizes data into structured tables with rows and columns. It supports the SQL language for defining and manipulating data, making it compatible with a wide range of applications [24]. PostgreSQL is recognized for its ACID compliance, ensuring data integrity and reliability in transactional operations. PostgreSQL is a versatile database that accommodates several types of workloads. It supports both online transaction processing (OLTP) and online analytical processing (OLAP), making it suitable for applications with diverse data handling requirements. Joins, indexing, and advanced query optimization capabilities contribute to efficient data retrieval and manipulation [25]. Also, PostgreSQL provides a range of features like support for spatial data, full-text search, and extensibility through stored procedures and triggers. It offers strong security features, including role-based access control. The PostgreSQL community is large and active, providing ongoing support, updates, and a wealth of extensions [1]. ClickHouse excels in analytical processing with its columnar storage, making it ideal for scenarios requiring rapid analytics over large datasets [26]. On the other hand, PostgreSQL offers a broader spectrum of capabilities, including robust support for transactions and diverse workloads, making it a versatile choice for various applications. The selection between ClickHouse and PostgreSQL depends on the specific requirements and nature of the data-processing tasks at hand [27].

Table 1 Comparison Table of Some Popular Databases

Reference	Feature/Aspect	MySQL	PostgreSQL	MongoDB	Oracle Database	Microsoft SQL Server	ClickHouse
[1]	Data Model	Relational	Relational	Document-oriented	Relational	Relational	Columnar
[2]	Primary Language	SQL	SQL	JSON-based queries	SQL	SQL	SQL
[3]	License	Open Source	Open Source	Server-Side Public	Proprietary	Proprietary	Open Source
[4]	ACID Compliance	Yes	Yes	Yes	Yes	Yes	Yes
[5]	Consistency Model	Strong	Strong	Eventual	Strong	Strong	Strong
[6]	Query Language	SQL	SQL	JSON Query Language	SQL	SQL	SQL
[7]	Joins	Yes	Yes	No	Yes	Yes	Yes
[8]	Indexing	Yes	Yes	Yes	Yes	Yes	Yes
[9]	Scalability	Horizontal	Horizontal	Horizontal & Vertical	Horizontal & Vertical	Horizontal & Vertical	Horizontal and Vertical
[10]	Partitioning	Yes	Yes	Sharding	Yes	Yes	Yes
[11]	Replication	Yes	Yes	Yes	Yes	Yes	Yes

[12]					Automatic Storage Management (ASM), Automatic Storage Management Cluster File System (ACFS)	SQL Server Storage Engine (various)	Merge Tree
	Storage Engine	InnoDB	PostgreSQL	WiredTiger, RocksDB			
[13]	JSON Support	Yes	Yes	Yes	Yes	Yes	Yes
[14]	Spatial Data Support	Yes	Yes	Yes	Yes	Yes	Yes
[15]	Full-Text Search	Yes	Yes	Yes	Yes	Yes	Yes
[16]	XML Support	Yes	Yes	No	Yes	Yes	No
[17]	Triggers	Yes	Yes	Yes	Yes	Yes	Yes
[18]	Stored Procedures	Yes	Yes	JavaScript-based	Yes	Yes	Yes
[19]	In-Memory Capabilities	Yes	Yes	No	Yes	Yes	Limited
[20]	Security Features	Yes	Yes	Role-Based Access Control	Advanced Security Options	Role-Based Access Control	Basic Role Based

							Access Control
[21]	Community Support	Large	Large	Large	Large	Large	Large
[22]	Use Cases	Web applications, Data Warehousing	Web applications, Geospatial applications	Document-oriented applications	Enterprise applications, Data Warehousing	Enterprise applications, Business Intelligence	Analytics, Processing, Time-Series Data, Logging, Monitoring

1.3 Integration of DBMS with Java Spring Boot Applications

In Java Spring Boot applications, efficient DBMS integration is crucial, especially for handling large datasets. The process involves establishing robust database connections, critical for performance but resource intensive [28]. To optimize this, DCP is used, caching reusable connections to reduce overhead. These pools enhance performance by minimizing latency and managing resources efficiently [29]. They support scalability and concurrency, essential for high-demand database access. Connection pools also offer flexibility, easily adapting to various database types and vendors in Java Spring Boot applications [30].

1.3.1 Integration of ClickHouse with Java Spring Boot Applications

Establishing a connection between a Java Spring Boot application and ClickHouse can be accomplished via three main approaches, each distinct in functionality:

Java Client: This client is well-suited for environments using OpenJDK version 8 or higher. It aligns with ClickHouse client version 0.5.0 and supports ClickHouse version 22.8+. Integrating new HTTP library dependencies is required, and it offers a broad array of data type support. Its

URL syntax allows for load balancing and failover parameters, contributing to a stable and effective connection [31].

JDBC Driver: Developed atop the ClickHouse client, the JDBC driver adheres to the standard JDBC interface and is beneficial for legacy systems, offering features like custom type mapping and transaction support. However, for performance-intensive applications or those requiring direct ClickHouse access, the Java Client is often the preferred choice [32].

R2DBC Driver: This driver, serving as a wrapper for the asynchronous Java client, caters to applications needing non-blocking database access. It complements the reactive programming paradigm, which is becoming increasingly relevant in the development of modern applications, especially those handling large data volumes or requiring scalability [33].

1.3.2 Integration of PostgreSQL with Java Spring Boot Applications

Integrating a Spring Boot application with a PostgreSQL database involves several steps to ensure a smooth and effective connection. Firstly, it is essential to add the PostgreSQL JDBC driver to the project's build file, such as Maven's pom.xml or Gradle's build.gradle. This step is crucial for enabling the Spring Boot application to communicate with the PostgreSQL database [34].

Next, configure the database connection settings in the application.properties file of the Spring Boot project. This includes specifying the database URL, username, and password, which are pivotal for establishing a successful connection to the PostgreSQL database [35].

Finally, leverage Spring Data JPA for convenient database operations. This involves creating entity classes that represent the database tables and are annotated with JPA annotations for mapping. These classes facilitate the Object-Relational Mapping (ORM) process, making it easier to perform CRUD operations within the Spring Boot application. Testing the entire setup is critical to validate the successful integration and ensure all database operations function as expected [36].

1.4 Integration of PostgreSQL with Java Spring Boot Applications

In my master's thesis, which focuses on comparing ClickHouse and PostgreSQL's integration with Java Spring Boot applications, Docker emerges as a pivotal component. Its deployment standardizes the environment for both databases, establishing a fair basis for comparison. Such uniformity is crucial for consistent performance evaluation and objective analysis [37].

Docker's ability to flexibly manage resources and configure environments aligns perfectly with my goal of easily modifying and optimizing database setups [38]. The containerization of these databases allows for swift scalability adjustments, facilitating their performance assessment under varied conditions. This flexibility is vital for a thorough comparison, particularly in examining scalability and resource efficiency [39].

Furthermore, Docker's consistent setup for ClickHouse and PostgreSQL validates my comparative analysis. By running both databases in an identical containerized environment, Docker removes potential biases in the comparison. This method reinforces the reliability of my findings and improves the reproducibility of my experimental setup, an essential aspect of scholarly research [40].

2 Tools and Components of Database Management Systems

In the fast-developing environment of database management systems (DBMS), several research studies have contributed to our knowledge of obstacles and breakthroughs, especially in the administration of massive datasets. These studies have been particularly helpful in managing enormous datasets. In the following paragraphs, we will go further into the work already done in this field and highlight significant discoveries and trends. The difficulties that come with the administration of massive datasets in the context of database management systems have been thoroughly investigated by researchers. Scalability appears as a primary problem, with studies highlighting the need to develop strategies that can expand without causing disruptions to meet increasing data quantities. Recurring topics include optimizing performance during read and write operations and ensuring high availability. Notable publications by [41] have provided an overview of the complexity involved in achieving economical scaling while maintaining the system's responsiveness. Recent research has emphasized head-to-head comparisons of well-known database management systems (DBMS), such as ClickHouse and PostgreSQL. The study conducted by [42] offered a comprehensive analysis of ClickHouse's capabilities in managing analytical workloads and highlighted the benefits associated with its columnar storage. On the other hand, authors in [43] conducted a comprehensive investigation of PostgreSQL's capabilities regarding transactional workloads. These studies offer the framework for understanding the complex performance characteristics of each system by providing the necessary background information.

The use of actual datasets significantly aids the formation of practical insights. For instance, the public data Yelp makes available has been used in several studies to recreate situations indicative of true data analytics difficulties. The works of [44] provide examples of how Yelp's datasets may be used to evaluate the effectiveness of DBMS in a variety of contexts, including user reviews, check-in analytics, and business engagement. The current body of research uses various methodological methods, including both quantitative and qualitative approaches. Benchmarking studies, like the ones done by [45], have concentrated on quantitative measures to evaluate system performance, while qualitative case studies, like the ones done by [46], have given in-depth analyses of particular use cases. Using such a wide variety of methodologies highlights the multidimensional nature of the issues posed by the administration of massive datasets.

Previous research has continuously been characterized by using real-world datasets and comparative studies as a defining characteristic. Researchers like [47] have used criteria such as response speed, scalability, and data size to make valid comparisons between database management systems (DBMS). These comparison assessments are helpful tools for companies that want to judge which database management system (DBMS) best suits their requirements [48].

An intricate picture emerges after doing an in-depth analysis of the available research. While relational database management systems (RDBMS) such as ClickHouse excel in analytical workloads, PostgreSQL remains reliable for transactional operations. Despite this, the study has clear holes, most notably in comprehending the most effective way to combine various systems to administer comprehensive massive datasets. The lack of a complete examination of cloud-based deployments, interaction with developing technologies, and a unified strategy that uses the benefits of both analytical and transactional databases are notable gaps. Other notable limitations include integration with existing technologies [49].

The corpus of work already in database management systems (DBMS) for managing massive datasets offers a rich tapestry of insights. Researchers have created the basis for a better understanding of the complexities of database systems by examining topics such as scalability issues and doing comparative evaluations of ClickHouse and PostgreSQL. Despite this, there is a clear need for research that tackles developing paradigms and provides a comprehensive and prospective viewpoint on the administration of massive datasets within the framework of DBMS. This demand arises as a direct result of the ongoing development of technology [50].

- **Tools and Components**

Database Management Systems, often DBMS, are essential to contemporary data management. These systems provide the infrastructure to store, organize, and effectively retrieve data. The usefulness, performance, and adaptability of a DBMS environment are all enhanced by the tools and components included inside it. The following is an in-depth examination of the primary tools and features that are essential to a dependable DBMS:

2.1 Database Management and Querying

Database management in the modern context is about structured data storage and retrieval with Relational Database Management Systems (RDBMS). Central to this process is the Structured Query Language (SQL), which, along with its components Data Definition Language (DDL) and Data Manipulation Language (DML), facilitates data structuring and handling [51]. Enhancing these operations are database engines and query optimizers, essential for streamlining data access. These elements together constitute the foundation of effective database management, highlighting the significance of organized data operations and adept querying in today's data-driven application landscape [52].

2.1.1 Relational Database Management System (RDBMS)

In database administration, a Relational Database Management System, often known as an RDBMS, is considered a cornerstone because of its stringent adherence to the relational model for the arrangement of data. At its heart, this method uses a structured approach by deploying tables defined by rows and columns to painstakingly portray the numerous connections between the many elements in the dataset. The usefulness of a relational database management system extends well beyond its primary function of storing data. RDBMSs provide a consistent and logical framework for organizing data, ensuring the links between various entities are retained and readily accessible [53]. The tables, comparable to linked matrices, serve as a graphical and logical representation of the underlying connections in the data. Not only does this hierarchical arrangement, founded on the principles of normalization, improve the data's quality, but it also makes searching simpler. Users can do sophisticated searches and get accurate results because of the structured format, which enables rapid information retrieval and optimization of such searches. Because it is based on the relational model, the relational database management system (RDBMS) provides a solid basis for data management. It does this by providing a standardized structure as well as an effective

mechanism for querying, which, in the end, makes it easier to glean insightful information from the data that has been stored [54].

2.1.3 Database Engine

When it comes to the execution of database operations that are specified by SQL queries, the database engine, which is a key component within the architecture of the database management system (DBMS), plays a crucial role. Its relevance stems from the ability to handle data storage, retrieval, and modification effectively, ensuring that the database system functions at its absolute best [55].

Users or applications may make SQL queries; the database engine will understand those questions and then carry them out. This central function of the database engine is known as the execution powerhouse. When a user makes a query, the database engine is responsible for executing the request. It identifies the method that will most effectively obtain or alter the required data. This includes accessing the proper data structures, using indexing methods, and optimizing the execution plan to deliver results as quickly as possible [56].

The database engine manages one of the most essential aspects: data storage efficiency. It is responsible for deciding on storage architectures, file management, and indexing algorithms, as well as overseeing the overall arrangement of data included inside the database. This careful management of the data storage leads to reduced access times, ensuring that information retrieval is carried out with the highest possible level of efficiency [57]. In addition, the database engine is exceptional for data retrieval since it can quickly get the required data depending on the requirements stated in the SQL queries. Its goal is to improve the responsiveness of the database system by using several different optimization strategies and algorithms to cut down on the latency and response time experienced [58].

When manipulating data, the database engine makes operations like adding, updating, or removing entries possible. It guarantees these actions are done correctly by conforming to the restrictions and preserving the database's consistency. This meticulous supervision of data processing operations adds to the information that has been saved, giving it a high degree of dependability and precision. Concisely, the database engine is the component that acts as the pivot point for the execution of SQL queries. It is responsible for managing data storage, retrieval, and modification, emphasizing efficacy. Users are given a responsive and trustworthy platform to interact with their

data due to its role in performance optimization, which is essential to the system that manages databases and ensures its smooth operation [59].

2.1.4 Query Processor/Optimizer

The query processor is a fundamental part of a database management system (DBMS) that is responsible for optimizing SQL queries to achieve the highest possible level of productivity. Since SQL queries are the primary method of communication between users or applications and the database, the job of the query processor is essential to ensuring that the database operates at its optimum level. An in-depth examination of SQL queries is performed throughout the optimization phase to determine which execution plan will provide the best results. During this examination, several aspects are considered, with the primary focus being improving the speed at which queries are processed. The availability of indexes and the employment of those indexes, which function as structured guides to accelerate data retrieval, are key factors. Strategic join techniques, which decide how tables are linked to satisfy the query, are another key issue [60].

The goal of the query processor is to simplify the execution of SQL queries by deftly navigating these parameters, which will result in a reduction in response times and consumption of resources. The relevance of this optimization process extends to the entire efficiency of the database system. By ensuring that queries are processed in a way that optimizes throughput and reduces latency, this optimization process ensures that overall efficiency is maximized [61]. The query processor is responsible for converting SQL queries into executable plans and optimizing those plans depending on the structural components of the database. This is a significant responsibility. This optimization process adds to improved performance and demonstrates the DBMS's versatility in catering to various query types and data circumstances. This adaptability allows the DBMS to serve its customers better [62].

2.2 Database Development and Administration

Database development and administration involve diverse activities and tools, focusing on database structure and data handling, and design and management tools. Key aspects include languages for database structuring and data manipulation, along with administrative and middleware tools that facilitate efficient database system creation and interaction. This area is

geared towards crafting and maintaining robust and scalable database systems to meet various application requirements [63].

2.2.1 Database Administrator (DBA) Tools

Database Administrator (DBA) tools are an essential part of the toolset of persons entrusted with the complex role of administering, monitoring, and improving Database Management Systems (DBMS) (DBMS). These tools are essential because they enable database managers to handle the intricacies of database administration. As a result, database systems can run more smoothly and be optimized more effectively. The capability of DBA tools to simplify and improve a variety of facets of database administration is at the heart of the relevance that these tools provide. These tools make available specialized interfaces and utilities that make it easier to do various activities essential to the operation and maintenance of a database [64].

Tuning the database's performance is one of the most essential areas in which DBA tools demonstrate their value. Database administrators use these tools to monitor the database's performance to identify bottlenecks, wasteful queries, and other variables that affect response. These tools allow administrators to fine-tune setups and optimize query execution plans, improving the system's overall performance. This is made possible by the comprehensive analytics and diagnostic functions that the tools provide [65].

DBA technologies also have a significant impact in security management, another essential domain. These solutions allow administrators to implement stringent security protocols by providing functions such as user access restrictions and encryption techniques, amongst other things. They make it easier to create access controls, audit trails, and encryption methods, which help to protect the database from unwanted access, data breaches, and other security risks. DBA tools are invaluable companions when backing up and recovering data. They provide tools for automatically creating and managing database backups, ensuring the preservation of vital data [66]. These technologies help to speed the recovery process in the unfortunate event that data is lost, a system fails, or other interruptions occur. As a result, downtime, and the possibility of losing data are reduced. In addition, DBA tools improve their ability to cover a broader range of database management tasks than they could before [67]. These technologies make responsibilities like essential maintenance, keeping software up to date, and monitoring the system more productive. They often give administrators centralized dashboards, alerting systems, and reporting tools that

provide in-depth insights into the health and state of the database [68]. DBA tools are significant because they provide administrators with a set of features that contribute to a database's efficient operation, security, and optimization. This ability is at the heart of the relevance that DBA tools have. These tools become force multipliers because they provide customized interfaces and utilities. As a result, database administrators can negotiate the complex problems of database administration with more agility and efficiency [69].

2.2.2 Database Design Tools

Database design tools are an essential component of the complex procedure of developing a database schema, which entails defining the tables, connections, and constraints that serve as the structural basis of an effective and well-organized database management system. Their importance comes from the fact that they simplify the intricate and subtle database construction process. This ensures the design is well-structured and follows normalization standards, fostering optimum performance and data integrity [70]. The most important aspect of these tools is the capacity of these tools to give architects, database administrators, and developers working on the design phase of a database with a user-friendly and frequently graphical interface. Users are given the ability to graphically envision the database's structure via these tools, which include the organization of tables, the definition of fields contained within them, and the construction of links between them. The process of designing a database is made more transparent and easier to understand by using this visual depiction [64].

Support for creating connections and constraints is an essential feature of database design tools, making it one of the most critical components of these tools. Relationships between tables, such as one-to-one, one-to-many, or many-to-many, are essential to organizing data in a manner that faithfully represents the circumstances that may arise in the actual world. Constraints protect the database against inaccuracies and abnormalities, including primary keys, foreign keys, and other integrity constraints. Regulations guarantee that data is accurate and consistent.

The importance of these tools becomes clear when keeping a normalized database structure. Normalization is a strategy for designing databases that reduces duplication and reliance by logically structuring tables to store data. This is accomplished via the process of normalization. Database design tools often contain normalization principles, which guide users in developing a database schema that eliminates data abnormalities and assures effective data retrieval. In addition,

the simplified procedure available by these technologies contributes to the development life cycle's effectiveness. Database design tools streamline the process of developing a solid database schema by offering a user interface that is both visible and straightforward. This performance is necessary, particularly in agile development settings, characterized by frequent iterations and frequent modifications to the database structure [65].

The importance of database design tools may be summed up by referring to their function as facilitators of the database design process. These tools simplify the process of developing a well-structured and normalized database schema. They do this by providing a graphical representation of the data and by assisting in constructing connections and constraints. In database administration, the significance of their contributions to the data's integrity and efficiency and the entire development life cycle makes them vital.

2.2.3 Middleware

Middleware, which can be summed up as anything that makes it easier for applications and databases to communicate, plays a crucial part in this process since it adds another layer of abstraction. The relevance of it resides in the fact that it improves flexibility and interoperability, hence facilitating interaction between applications and various databases. Middleware's primary function is to act as an intermediate layer, abstracting the complexity of database interactions and protecting applications from the nuances of different database management systems (DBMS). This abstraction helps the general flexibility of the system. It makes it possible for applications to continue functioning without regard to the behind-the-scenes database technologies.

The capacity of middleware to facilitate interoperability is the primary factor that contributes to the relevance of this kind of software. Middleware is a necessary unifying factor since applications often need to connect with several databases, each of which has its own distinct protocols and communication mechanisms. Regardless of the technologies that are being used, it serves as a bridge that standardizes the communication that takes place between applications and databases. Because of this standardization, apps have a greater capacity for adaptation since they can interact with various databases smoothly and do not need significant alterations. Middleware is notable for the crucial quality of bringing flexibility to the table. Applications can continue to be dynamic and adaptive despite changes in the database architecture because of this technology, which works by abstracting the complexities of database connectivity. This is of the utmost significance in dynamic

settings, where databases may be improved, changed, or varied. Middleware serves as a protective buffer, ensuring that changes in the database design do not need large modifications in the application layer. This is because middleware intermediates between the database and the application layer [71].

Additionally, middleware aids scalability since it makes it easier for dispersed components to communicate with one another. Middleware facilitates efficient communication in situations in which applications and databases are dispersed over several servers or even in cloud settings, which increases the scalability of the entire system. Middleware is an essential component in the design of contemporary computers. It is a vital tool because of its capacity to simplify complex matters, streamline communication, and improve both flexibility and interoperability. Middleware helps to facilitate the development process by providing a standardized interface between applications and databases. This interface promotes adaptability and ensures that applications can interact seamlessly with various databases, contributing to the system's overall efficiency and nimbleness [72].

2.3 Maintenance and Security

In database management, two key elements are maintenance and security. Maintenance covers essential tasks like data backup and recovery, ensuring data is both preserved and restorable. Security focuses on protecting data from unauthorized access, incorporating strict access control and encryption. The use of monitoring and tuning tools is also essential, as they help maintain database efficiency and detect security risks.

2.3.1 Backup and Recovery Tools

The critical processes of creating database backups and facilitating recovery in the face of data loss or unexpected system failures are facilitated by backup and recovery tools, which play a crucial role in database management by automating these processes and saving time and effort. Their definition comprises a collection of capabilities meant to preserve the integrity and availability of data, providing a protection blanket against interruptions. The value of these technologies is made clear when one considers their capacity to protect essential data from being lost or corrupted by mistake, hardware failing unexpectedly, or any other disastrous occurrences. They guarantee that a copy of the database that is consistently made and kept in a safe place is protected by automating the backup process. This ensures that the database copy reflects its state

at a particular moment. This method removes the need for manual interventions, reducing the likelihood of the backup process suffering from supervision or other human error. If data is lost or there is a fault in the system, recovery solutions are necessary. These technologies make it possible to restore the database using copies made in the past, which helps reduce the time the company is down and any potential interruptions. It is impossible to emphasize the relevance of having this capacity, particularly in mission-critical systems where having continuous access to data is of the utmost importance. The mechanisms given by backup and recovery solutions go beyond the typical backup techniques used in the past. These mechanisms often include features such as incremental backups, differential backups, and backups based on snapshots. These sophisticated capabilities allow businesses to modify their backup procedures following the needs of their operations, improving storage consumption and recovery times. In addition, the value of these technologies is further emphasized by the function that they play in both compliance and data governance. In many fields of business, compliance with legal standards necessitates adopting comprehensive backup and recovery procedures to guarantee data availability, integrity, and confidentiality. The automated and methodical approach that backup and recovery systems use is helpful to enterprises in the process of conforming to these demanding compliance criteria [66].

When used in cloud-based contexts, these technologies become even more successful because of their seamless integration with cloud storage services, enabling them to provide scalable and cost-efficient solutions. This versatility guarantees that businesses can match their backup and recovery plans with the dynamic nature of current IT infrastructures. This is essential for maintaining data integrity. The relevance of backup and recovery tools rests in their capacity to automate and simplify the essential operations of producing database backups and restoring data in the case of unforeseen occurrences. This is the primary reason for the importance of backup and recovery programs. These technologies are essential to the execution of an all-encompassing data management strategy because they provide businesses the peace of mind that their information is secure, that it can be recovered, and that it can withstand the effects of unanticipated obstacles.

2.3.2 Data Security Components

A database management system's security components comprise a suite of features that center on access control, encryption, and authentication procedures. This suite of features is known as the database security architecture. Their definition underscores their function in further strengthening

the protection of sensitive data, highlighting the essential need for steps to maintain the data's confidentiality and integrity. There is a crucial need for measures to ensure the data's secrecy and integrity [67].

The power of security components to protect databases against unauthorized access and data breaches is at the heart of these components' relevance. Access control methods are essential since they manage user permissions and privileges to establish who among the users may access specific data or carry out actions inside the database. These components help defend against unauthorized users seeking to compromise sensitive information by enforcing the concept of least privilege. This principle states that users should be granted the fewest possible privileges. Encryption is essential to any data protection strategy during transmission and while the data is being stored. Encryption algorithms, part of security components, transform data into unreadable forms. These formats can only be decoded by using the necessary cryptographic keys. Because of this, even if illegal access is acquired, the data that is intercepted will still be incomprehensible, which will guarantee that its secrecy is maintained [68].

The identities of people accessing the database are checked and confirmed by authentication techniques included in the security components. This necessitates the usage of usernames and passwords, multi-factor authentication and any other authentication mechanisms. The value of these mechanisms rests in their capacity to prohibit malevolent players from adopting unauthorized roles inside the database system. This is where the significance of these mechanisms comes from. In addition to preventing illegal access, security components contribute to preserving data integrity. They integrate features like data validation tests, hash functions, and integrity requirements to guarantee that the data does not become inaccurate and does not get changed. This prohibits tampering with the data or corruption, which is essential for maintaining the reliability of the information that is kept in the database [69].

In the modern business environment, when data breaches provide a significant risk to companies, the importance of security components cannot be overstated. They not only secure sensitive data but also conform with legal and compliance standards that necessitate the protection of user information and privacy. This is because they take data security very seriously. One's reputation can be harmed, legal repercussions to be incurred, and financial damages to be incurred if effective security measures are not implemented. The importance of the many security components included inside a database management system may be broken down into several categories. To protect

users' privacy and maintain the reliability of stored information, they come together to build an all-encompassing security system that includes access control, encryption, and authentication functions. These components are essential for firms that want to strengthen their databases against threats and vulnerabilities today when data breaches are so commonplace [70].

2.3.3 Database Monitoring and Tuning Tools

Database Monitoring and Tuning Tools are specialized instruments that monitor database performance, detect bottlenecks, and offer improvements. These tools may also be used to tune database performance. Their value comes from the fact that they can improve database efficiency by resolving performance problems and maximizing the usage of system resources. The capacity of these tools to monitor several elements of database performance in real-time is at the core of their functionality. Metrics like query execution times, resource usage, transaction rates, and system health indicators are continually collected and analyzed by them. This monitoring tool offers administrators vital insights into the operational state of the database. This enables the administrators to rapidly discover any abnormalities or performance deterioration that may have occurred. These tools can perform an essential job, which is to identify bottlenecks. The term "bottleneck" refers to any point in a system where performance is restricted, which often results in slowing activities. Database Monitoring and Tuning Tools use complex algorithms to identify bottlenecks, which hardware limits, inefficient queries, or other systemic problems might cause. These bottlenecks can be located and resolved using these tools. When blockages are identified, administrators can more strategically target their efforts toward correcting issue areas, improving the database's overall efficiency [73].

The capacity of these tools to provide recommendations for improvements based on the data they have collected and analyzed is where their genuine value lies. After discovering bottlenecks, the devices can suggest changes to the database setup, query optimizations, or indexing methods. These suggestions are often based on best practices and adapted to the database's workload. It is possible that implementing these optimizations may result in observable gains in speed, which will ensure that the database functions to the best of its ability.

In addition, Database Monitoring and Tuning Tools are essential components of proactive database administration. These technologies can foresee prospective performance difficulties using trend analysis and predictive modeling, which allows them to do so before the issues become more

severe. Because of this proactive approach, administrators can take preventative steps, which helps ensure that the database maintains ideal performance levels even if the amount of work being done changes or increases over time. The relevance of Database Monitoring and Tuning Tools rests in their capacity to give real-time insights into database performance, detect bottlenecks, and recommend practical adjustments. This is where the importance of these tools comes into play. Administrators can maintain a high degree of operational efficiency by using these tools. They can also quickly fix performance concerns and proactively tune the database to adapt to changing workloads and needs [74].

2.3.4 Transaction Manager

Within a Database Management System (DBMS), the transaction manager is a pivotal component that sustains the underlying principles defined by the acronym ACID. These concepts include atomicity, consistency, isolation, and durability. When taken as a whole, these guiding principles establish the resilience and dependability of database transactions, which helps ensure that the database maintains its integrity despite various operating circumstances. Atomicity refers to how a transaction is handled as a single, undivided component of the overall task. The transaction manager is responsible for ensuring that every operation included in a transaction is completed. This avoids circumstances in which just a subset of operations is performed, hence preserving the state of integrity of the database.

The idea that a database may transition from one legitimate state to another via transactions is fundamental to consistency. The transaction manager ensures that the execution of a trade does not break any previously set rules or integrity requirements, ensuring that the database maintains its general consistency. The execution of one transaction is kept entirely separate from any other transactions that may be taking place simultaneously by isolating them. The transaction manager is responsible for preventing interference between different transactions and ensuring that the simultaneous execution of other transactions does not influence the results of one transaction.

The concept of durability highlights the dedication to ensuring that committed transactions remain permanent. Once a transaction has been appropriately committed, the transaction manager ensures that its modifications will remain even if there is a future failure, such as a crash in the system or a loss of power. The transaction manager's duty to protect these ACID features gives rise to the relevance of its function in the system. The transaction manager offers a solid foundation for

managing transactions by strictly ensuring atomicity, consistency, isolation, and durability. This gives the manager the capacity to handle transactions effectively. This protects the database against inconsistencies and assures users and applications that operations will either be completely successful or leave the database undisturbed, avoiding inconsistent or only partially effective modifications. The transaction manager fulfills the role of a sentinel by maintaining the dependability and trustworthiness of the database within the context of the ever-changing environment of transactional processes [59].

2.3.5 Database Connectivity Drivers

Connectivity drivers are critical channels via which databases and applications can successfully communicate. In doing so, they bridge the gap between these independent yet interdependent entities. Their relevance comes in the fact that they allow seamless integration, making it possible for applications and databases to communicate data resiliently and efficiently. Connection drivers' primary responsibility is to serve as mediators, converting applications' communication protocols and formats into a language the database can understand and vice versa. This is the fundamental function of connectivity drivers. This language translation is essential because databases often utilize specific protocols and data formats to store and manage information. At the same time, apps may use a variety of different standards for the representation of their data [62].

When viewed in the context of software applications in a varied and ever-changing environment, the relevance of connection drivers becomes more evident. Regardless of the underlying database management system, applications, which may be written using a variety of programming languages and frameworks, need a common approach to communicate with databases (DBMS). Connectivity drivers provide a standardized user experience to overcome this obstacle. This makes it possible for programs to connect fluidly with various database types. In addition, the drivers significantly impact the enhancement and optimization of the effectiveness of the data interchange. They contribute to the overall performance of applications that depend on database interactions by reducing communication and offering a standardized interface. This helps the apps run better. This optimization is essential, particularly when accessing data in real time or conducting data transactions quickly is of the utmost importance [63].

The seamless integration made possible by connection drivers has significant consequences for the operation of programs and the experience they provide to end users. It guarantees that programs

may interface with databases in a standard way, regardless of the specificities of the underlying database technology. This makes it possible for apps to work with several database technologies. This standardization streamlines the development process, lessens the complications associated with database connection, and increases compatibility across various applications and database systems. The relevance of connection drivers is founded on their function as indispensable bridges, making it possible for applications and databases to communicate effectively. Software applications that depend on database interactions tend to have higher efficiency, interoperability, and overall performance levels than those that do not. This is because of the capacity of database interactions to allow and optimize seamless integration.

2.4 Structured Query Language (SQL)

SQL, which stands for Structured Query Language, is a crucial domain-specific language developed expressly for administering and manipulating relational databases. It is impossible to overestimate its importance within database administration because it is the primary and universal interface via which users interact with the databases being managed. SQL is a lingua franca that enables users to carry out various operations. These duties include everything from searching the database for information to maintaining and changing the stored data. The language offers a standardized set of instructions and syntax, paving the way for users of relational databases to communicate on a level playing field. Users can execute complicated queries to extract certain subsets of data via SQL, which enables the retrieval of information efficiently and focused. Users can easily traverse massive datasets using SQL's querying capabilities, which allows for extracting relevant insights in a structured and organized way [55].

In addition, the capability of SQL is not limited to only accessing the database; it also includes essential actions such as adding new data, changing existing records, and maintaining database structures. It serves as a flexible tool that allows users to adjust the contents of a database in response to shifting needs, and it does this effectively. The dynamic flexibility of the database must be ensured by the capacity to accomplish operations such as adding new entries or modifying existing ones. This guarantees that the database can meet the ever-changing data demands [29]. Users are provided with a standardized and powerful means to interact with the underlying data, made possible by SQL's function as the major interface for relational databases. SQL's role as the primary interface for relational databases is essential. Because of its global applicability and

powerful expressive powers, it is an instrument that cannot be ignored by anybody engaged in the administration and exploitation of relational databases.

2.4.1 Data Definition Language (DDL)

Data Definition Language, often known as DDL, is a subset of Structured Query Language (SQL) exclusively devoted to establishing and maintaining the structural components inside a database. This contains fundamental elements such as tables, indexes, and constraints in the database. The ability of DDL to allow the creation, modification, and deletion of these essential database objects is what gives it its relevance. This capacity is vital to maintaining the overall structural integrity of the database [43]. At its heart, DDL comprises a collection of SQL commands that, when executed, provide users and database administrators to determine a database's structure and its contents' arrangement. To do this, the tables used to store the data must be specified, the columns included inside those tables must be defined along with the data types they will store, and relationships must be established using constraints. The facilitation of the construction of tables, which act as the fundamental frameworks for the organization of data, is one of the key responsibilities of DDL. Users can specify the properties of each table using DDL commands. This includes the data type stored in each column, the constraints used to ensure the data's integrity, and the indexes used to improve the speed of data retrieval [56].

In addition to this, DDL is an essential tool for altering preexisting database structures. This may entail modifying the definitions of the tables, adding, or deleting columns, and revising the constraints to meet changes in the data needs or upgrades to the system. A significant feature of DDL is that it allows the database structure to be modified while preserving its previous state. In addition, since it makes it possible to delete items from the database, DDL is essential in keeping its integrity intact [57]. DDL instructions make eliminating tables and indexes in a database easier when the tables or indexes are no longer required or when the database is being reorganized. This procedure must be carried out carefully to avoid having an effect that was not intended and to ensure that the database's structure remains coherent. Briefly, DDL is the language that acts as the medium for shaping and administering the fundamental components of a database. DDL guarantees that the database continues to be structurally sound and aligns with the ever-evolving data management demands by allowing for constructing, updating, and deleting structures such as tables, indexes, and constraints [42].

2.4.2 Data Manipulation Language (DML)

Data Manipulation Language, often known as DML, is a subset of Structured Query Language (SQL) developed expressly for dealing with the data saved in a database. DML is a fundamental part of SQL. The dynamic markup language, or DML, is essential because it gives users the tools they need to dynamically obtain and change data, contributing to the efficient administration of database material. The SELECT statement, an essential operation that enables users to get data from one or more tables in the database, is at the center of DML. It is this command that gives users the ability to manipulate data. This statement makes it possible to specify the columns that should be obtained, as well as requirements for the data selection, and it gives users the option to combine tables to conduct more complex searches [58].

Users can add new records or rows into a table inside the database using another essential DML component known as the INSERT statement. The dataset may be expanded and enriched because users can define the values for each column of the newly added row.

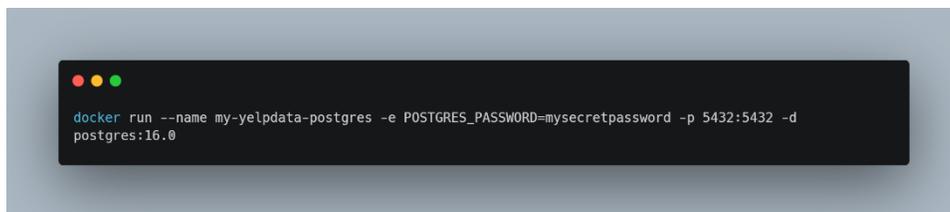
The UPDATE statement is used whenever a table needs to edit the data already present. Users can identify the columns that will be updated and the new values for them. This is usually done based on specific criteria that decide which rows should be modified. The DELETE statement may be used to delete records or rows from a database depending on the criteria given in the statement. This capability is essential for selectively eliminating data that is no longer required or satisfies specific criteria to save space. In its most basic form, DML in SQL offers a flexible collection of instructions that may be used to perform various data manipulation tasks. These actions ensure that users, apps, and administrators may dynamically interact with the database. They vary from basic data retrieval to complicated modifications. This kind of dynamic engagement is essential to keep data accurate, up-to-date, and in line with the constantly changing needs of the company [42].

3 Analysis and Findings

3.1 Experimental System Setup

The technical setup and setups that are required for the upcoming experiments are presented in this chapter, which serves as the basis for the presentation. The setup of both ClickHouse and PostgreSQL databases is broken down in great depth, with a particular focus on the essential role

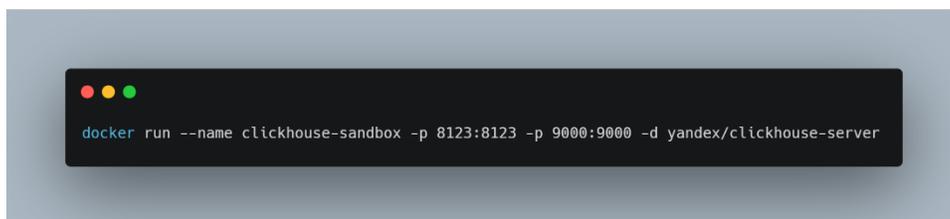
that Docker containers play in simplifying the process. Reproducibility and uniformity of the experimental setting are ensured by specific versions and figures. Prior to going into the protocols of the experiment, this chapter starts by elaborating on the significance of establishing the experimental systems. The focus is on clarity and openness, with the goal of ensuring that readers can recreate the experimental circumstances easily. In this chapter, the Docker instructions used to initialize both databases are provided. This offers a clear beginning point for the tests, which in turn creates a common ground for comprehending the remaining portions of the study. Setting up the ClickHouse and PostgreSQL databases to their default settings is an essential step that must be taken before beginning the experimental operations. Docker containers make this startup procedure easier to get started with. PostgreSQL version 16 and the Yandex image of ClickHouse were the specific tools that I employed for this project. While providing the PostgreSQL database with its initial configuration, a Docker container was used using the following command.

A terminal window with a dark background and light text. The command is: `docker run --name my-yelpdata-postgres -e POSTGRES_PASSWORD=mysecretpassword -p 5432:5432 -d postgres:16.0`. The terminal has three colored window control buttons (red, yellow, green) in the top left corner.

```
docker run --name my-yelpdata-postgres -e POSTGRES_PASSWORD=mysecretpassword -p 5432:5432 -d postgres:16.0
```

Figure 2 Docker run PostgreSQL

This command creates and runs a Docker container named "my-yelpdata-postgres" using the PostgreSQL version 16.0 image. The environment variable "POSTGRES_PASSWORD" is set to "mysecretpassword," and the container is mapped to the host machine's port 5432 for accessibility. A similar approach was adopted for ClickHouse initialization.

A terminal window with a dark background and light text. The command is: `docker run --name clickhouse-sandbox -p 8123:8123 -p 9000:9000 -d yandex/clickhouse-server`. The terminal has three colored window control buttons (red, yellow, green) in the top left corner.

```
docker run --name clickhouse-sandbox -p 8123:8123 -p 9000:9000 -d yandex/clickhouse-server
```

Figure 3 Docker run ClickHouse

Using the Yandex ClickHouse image, this command will start a Docker container that will be called "clickhouse-sandbox." For interaction purposes, the container is mapped to ports 8123 and 9000, and it operates in detached mode (also known as "-d") for execution in the background. The repeatability of the database setup procedure is ensured by these instructions, which adhere to the established protocols. This Docker command initializes a container named "clickhouse-sandbox" using the Yandex ClickHouse server image. It maps ports 8123 and 9000 for external access and runs in detached mode ("-d") to execute in the background. This standardized command ensures the consistent and replicable initialization of the ClickHouse database within a Docker environment.

Upon execution of these commands, both containers can be observed as active entities within the Docker Desktop environment. I will use DBeaver to visualize the databases. Connecting to databases running within Docker can be achieved through DBeaver.

The procedure of uploading massive datasets into both databases can be compared, which will allow us to begin our first experiment. To get things started, we have five comprehensive datasets, each of which is represented by a JSON file. Each JSON file corresponds to a different table in the Yelp architecture. Within the framework of the Yelp design, the goal is to develop five tables.

Primarily, we will be concentrating on the ClickHouse database. A Bash script that permits the parsing of JSON files, the creation of tables, and the insertion of all entries is made available by ClickHouse. We must run the clickhouse-client program to get this procedure started. Because both the PostgreSQL and ClickHouse containers are now operating inside Docker, we can determine the Docker container ID of ClickHouse by using the command `docker ps`. Once we have obtained the container ID, we are able to launch the clickhouse-client by executing the following command: "docker exec -it <ClickHouse Container ID> clickhouse-client."

Replace <ClickHouse_Container_ID> with the actual ID of your ClickHouse Docker container. This command opens an interactive terminal session (-it) within the specified ClickHouse container and launches the ClickHouse client, providing a direct interface for executing ClickHouse commands.

This command will grant us access to the ClickHouse client server. Subsequently, we can proceed with the necessary operations within the ClickHouse environment.

```
Command Prompt - docker exec -it 4bcf80fe4428 clickhouse-client
Microsoft Windows [Version 10.0.19045.3570]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nijat>docker ls
docker: 'ls' is not a docker command.
See 'docker --help'

C:\Users\nijat>docker ps
CONTAINER ID   IMAGE                                COMMAND                                CREATED        STATUS        PORTS
4c0d0f2c0870   postgres:16.0                       "docker-entrypoint.s..."           2 weeks ago   Up 18 hours   0.0.0.0:5432->5432/tcp
4bcf80fe4428   clickhouse/clickhouse-server:latest "/entrypoint.sh"                    5 weeks ago   Up 18 hours   9000/tcp, 0.0.0.0:8123->8123/tcp, 9009/tcp
docker_sandbox

C:\Users\nijat>docker exec -it 4bcf80fe4428 clickhouse-client
ClickHouse client version 23.9.1.1854 (official build).
Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 23.9.1 revision 54466.

Warnings:
* Linux transparent hugepages are set to "always". Check /sys/kernel/mm/transparent_hugepage/enabled

4bcf80fe4428 :)
```

Figure 4 ClickHouse-Client

We need to travel to the stated folder inside the container and copy the required JSON file from the host system to fulfil the requirement that the ClickHouse client access files that are located outside of the Docker container where it is being used. The command to copy the file might be done in the following manner, specifically for the user table.

```
docker cp C:\Users\nijat\Downloads\yelp_academic_dataset_user.json clickhouse-sandbox:/var/lib/clickhouse/user_files/
```

Figure 5 Copying a File to a Docker Container

This command instructs Docker to copy the specified JSON file from the host machine(located at C:\Users\nijat\Downloads\yelp_academic_dataset_user.json) to the /var/lib/clickhouse/user_files/

path within the ClickHouse container (clickhouse-sandbox). This step is crucial to ensure that the ClickHouse client can recognize and operate on the JSON file. Repeat similar commands for each JSON file corresponding to the different tables in the Yelp architecture, adapting the paths and filenames accordingly.

Then we can start executing the Bash script to create a table and insert a 3GB file into it.



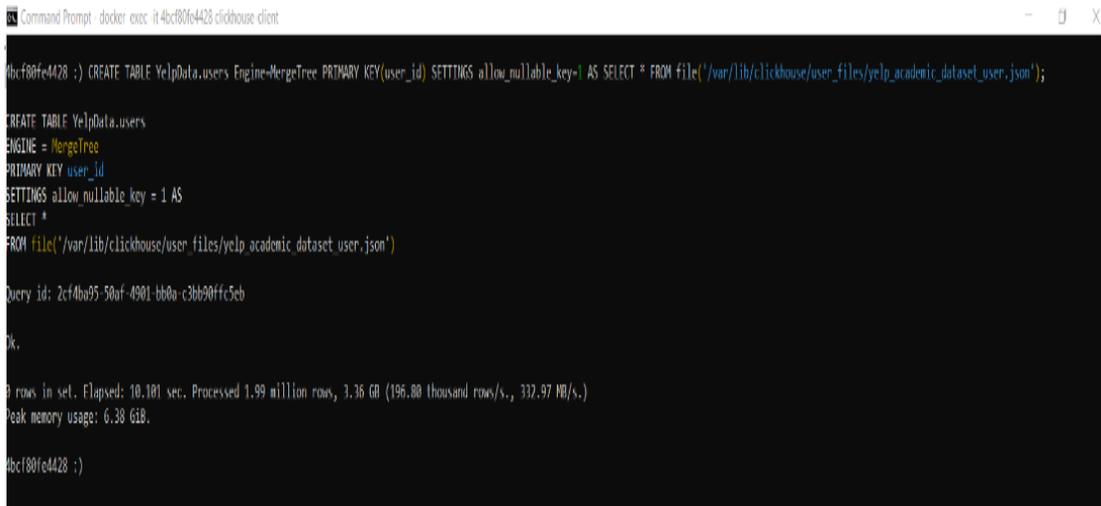
Figure 6 Business Table created

Creating a table named 'users' in the 'YelpData' database in ClickHouse

```
CREATE TABLE YelpData.users
(
    -- Define table columns based on the structure of the JSON file
    -- For example: user_id, name, age, etc.
)
ENGINE = MergeTree
-- Specify the primary key for the 'users' table
PRIMARY KEY user_id
-- Allowing nullable keys based on the specified setting
SETTINGS allow_nullable_key = 1
-- Populating the 'users' table by selecting all records from the specified JSON file
AS
SELECT * FROM file('/var/lib/clickhouse/user_files/yelp_academic_dataset_user.json');
```

Figure 7 Create 'Users' Table in ClickHouse's YelpData

The execution of this command is remarkably swift, requiring a mere 10.101 seconds. Notably, the process successfully processed 1.99 million rows from a file size of 3.36 GB, yielding an exceptionally favorable outcome.



```
Command Prompt - docker exec -it 4bcf80fe4428 clickhouse-client
4bcf80fe4428 :) CREATE TABLE YelpData.users Engine=MergeTree PRIMARY KEY(user_id) SETTINGS allow_nullable_key=1 AS SELECT * FROM file('/var/lib/clickhouse/user_files/yelp_academic_dataset_user.json');

CREATE TABLE YelpData.users
ENGINE = MergeTree
PRIMARY KEY user_id
SETTINGS allow_nullable_key = 1 AS
SELECT *
FROM file('/var/lib/clickhouse/user_files/yelp_academic_dataset_user.json')

Query id: 2cf4ba95-50af-4901-bb0a-c3bb90ffc5eb

Ok.

0 rows in set. Elapsed: 10.101 sec. Processed 1.99 million rows, 3.36 GB (196.80 thousand rows/s., 332.97 MB/s.)
Peak memory usage: 6.38 GiB.

4bcf80fe4428 :)
```

Figure 8 Dataset Command

The 'users' table has been successfully created, and the migration of all data from the JSON file to the table has been completed in a mere 10 seconds. Proceeding with analogous procedures, the migration of data from JSON files to corresponding tables within the ClickHouse database was accomplished in less than 5 minutes.

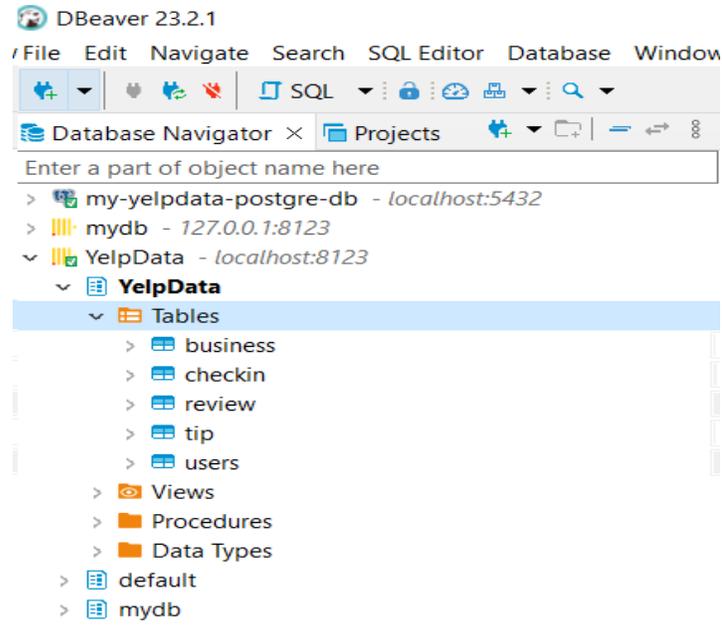


Figure 9 JSON Files to Corresponding Tables

It is difficult to establish a table in PostgreSQL based on a JSON file and migrate data without interruptions since there is no direct way. The COPY command is the primary command supported by the PostgreSQL Command Line Interface (CLI). However, this command cannot insert records into more than one column at a time, which is a restriction that is sadly present. This technique does not satisfy the process requirements. It is necessary to investigate other approaches to the migration of records into the PostgreSQL database in order to circumvent this constraint and reach a result that is comparable to that of ClickHouse.

There are many ways to upload massive amounts of data into PostgreSQL; however, one approach requires using a programming language such as Java, Python, or any of the other available options. The Java Spring Boot application was the one we decided to use for the experiment. While uploading a file that was 3.13 gigabytes in size to the PostgreSQL database, which was restricted to the 'user' table, a satisfactory result was achieved, and the procedure took around three minutes to complete. On the other hand, considering it is necessary to parse and upload each file that corresponds to different tables, this strategy can prove to be arduous and impractical when dealing with several files.

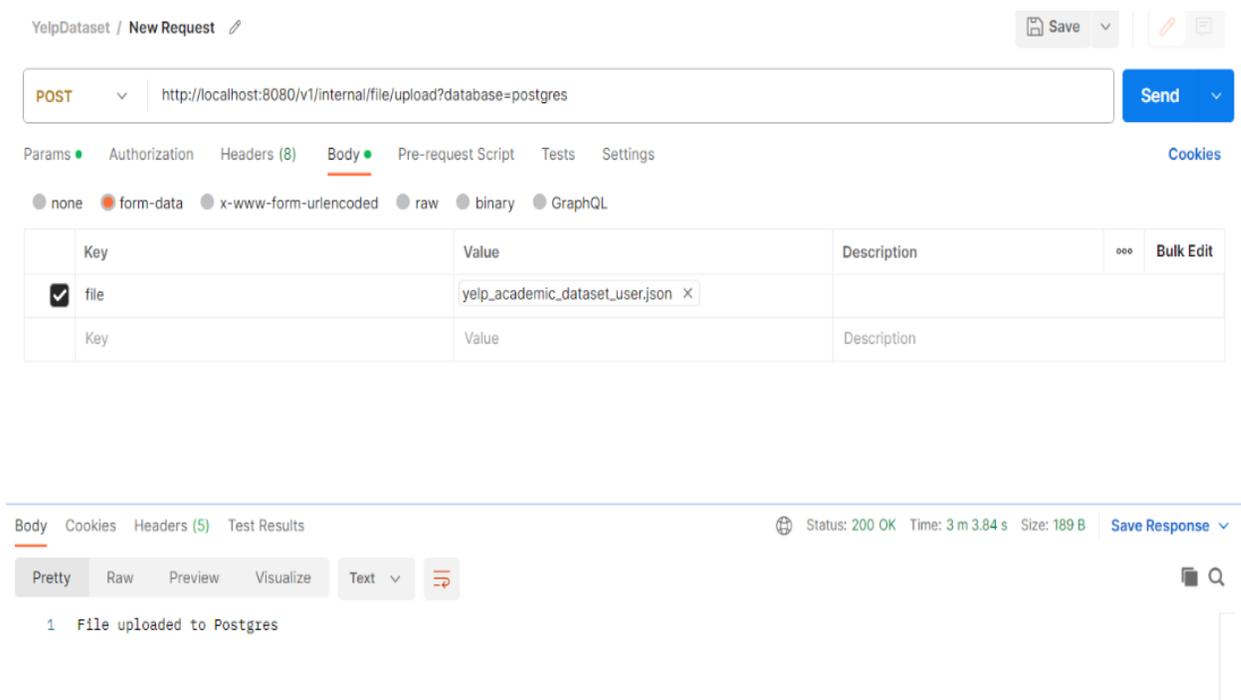
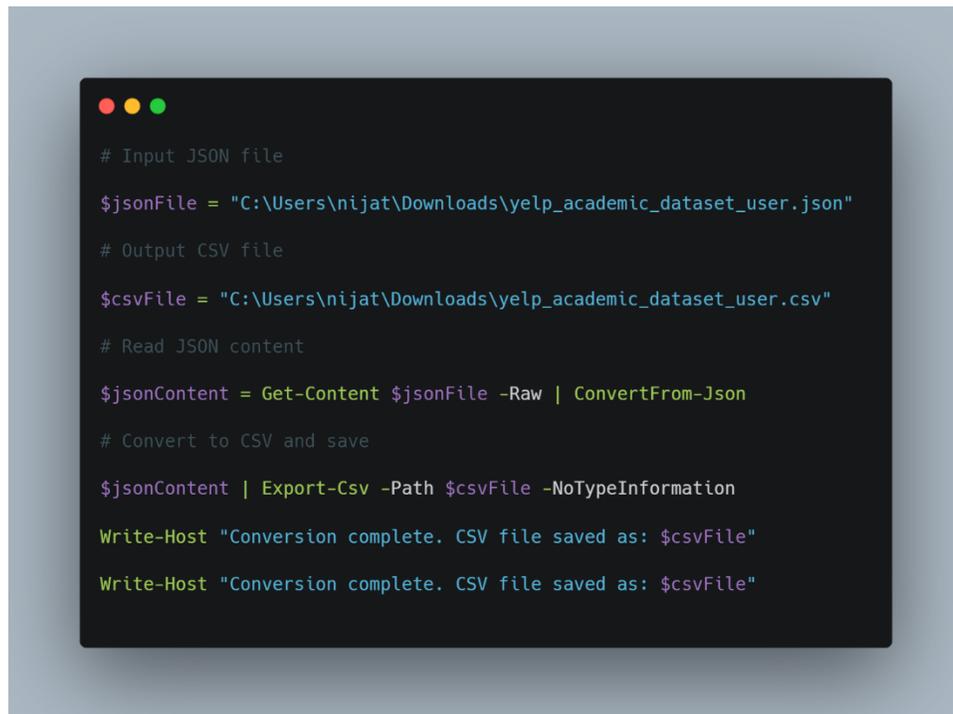


Figure 10 Uploading Records to PostgreSQL Database

The second approach entails converting the JSON file to the CSV format via manual translation. The process of uploading massive datasets into the PostgreSQL database may be made more accessible with the help of this approach, which provides a potential answer. The attempts to upload a huge quantity of data by means of a script, on the other hand, were faced with problems. The limits imposed by Windows PowerShell may be primarily responsible for these issues. The vast datasets that Windows PowerShell typically stores in memory before converting them to a CSV file are often large. It may be challenging to run the script fluidly due to this restriction that is inherent to Windows PowerShell. This is especially true when dealing with tremendous datasets. Because of this, the manual conversion of the JSON file to the CSV format looks more realistic for handling massive volumes of data uploaded to the PostgreSQL database. This is because the JSON file is in the CSV format.

We can go around the memory constraints that are imposed by the execution of the script in Windows PowerShell if we choose the option that allows for manual conversion. The use of this

tactic makes it feasible to create a process that is not only more efficient but also simpler to manage. It ensures that the conversion and subsequent uploading of each file may be completed without encountering any unneeded problems that are associated with memory limitations throughout the process. The trade-off in terms of reliability and practicality may prove to be favorable even though there is a step that must be taken manually. It is essential to keep this in mind when dealing with large datasets, which may sometimes place a strain on the capabilities of script execution under certain circumstances.



```
# Input JSON file
$jsonFile = "C:\Users\nijat\Downloads\yelp_academic_dataset_user.json"
# Output CSV file
$csvFile = "C:\Users\nijat\Downloads\yelp_academic_dataset_user.csv"
# Read JSON content
$jsonContent = Get-Content $jsonFile -Raw | ConvertFrom-Json
# Convert to CSV and save
$jsonContent | Export-Csv -Path $csvFile -NoTypeInformation
Write-Host "Conversion complete. CSV file saved as: $csvFile"
Write-Host "Conversion complete. CSV file saved as: $csvFile"
```

Figure 11 JSON to CSV Conversion Script

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\nijat> cd C:\Users\nijat\Downloads\
PS C:\Users\nijat\Downloads> .\converter.ps1
Get-Content : Insufficient memory to continue the execution of the program.
At C:\Users\nijat\Downloads\converter.ps1:8 char:16
+ $jsonContent = Get-Content $jsonFile -Raw | ConvertFrom-Json
+ ~~~~~
+ CategoryInfo          : InvalidOperation: (:) [Get-Content], OutOfMemoryException
+ FullyQualifiedErrorId : ProviderContentReadError,Microsoft.PowerShell.Commands.GetContentCommand

Conversion complete. CSV file saved as: C:\Users\nijat\Downloads\yelp_academic_dataset_user.csv
PS C:\Users\nijat\Downloads>
```

Figure 12 Conversion Complete

It might be beneficial to investigate an alternate approach that involves stream processing. To do this, a script that has been given the name converter with stream.sh has been developed, and the execution of this script is started from inside Windows PowerShell. This approach was chosen to study the effectiveness of stream processing because it can provide benefits in situations where memory restrictions play a crucial role throughout the process of data conversion and subsequent uploading.

```
# Input JSON file
$jsonFile = "C:\Users\nijat\Downloads\yelp_academic_dataset_user.json"
# Output CSV file
$csvFile = "C:\Users\nijat\Downloads\yelp_academic_dataset_user.csv"
# Open the JSON file for reading
$jsonReader = [System.IO.StreamReader]::new($jsonFile)
# Open the CSV file for writing
$csvWriter = [System.IO.StreamWriter]::new($csvFile, $false, [System.Text.Encoding]::UTF8)
# Try-finally block to ensure the streams are closed
try {
    # Read the first line (header) from the JSON file
    $jsonHeader = $jsonReader.ReadLine()
    $csvWriter.WriteLine($jsonHeader)
    # Process each subsequent line from the JSON file
    while ($jsonReader.Peek() -ge 0) {
        $jsonLine = $jsonReader.ReadLine()
        $jsonObject = $jsonLine | ConvertFrom-Json
        # Convert JSON object to CSV line and write to the CSV file
        $csvLine = ($jsonObject.PSObject.Properties.Value -join ',')
        $csvWriter.WriteLine($csvLine)
    }
}
finally {
    # Close the streams
    $jsonReader.Close()
    $csvWriter.Close()
}
Write-Host "Conversion complete. CSV file saved as: $csvFile"
```

Figure 13 JSON to CSV Conversion Script

This script reads a JSON file, converts its contents to CSV, and writes the CSV data to a specified file. It ensures proper handling of streams with a try-final block to guarantee closure. The final line of the script outputs a message indicating the completion of the conversion process and the path where the CSV file is saved.

Write-Host "Conversion complete. CSV file saved as: \$csvFile"

The execution of the script was successful, enabling the conversion of JSON to CSV through the Bash script. The subsequent step involves utilizing a Database Viewer to import the CSV file into the PostgreSQL database.

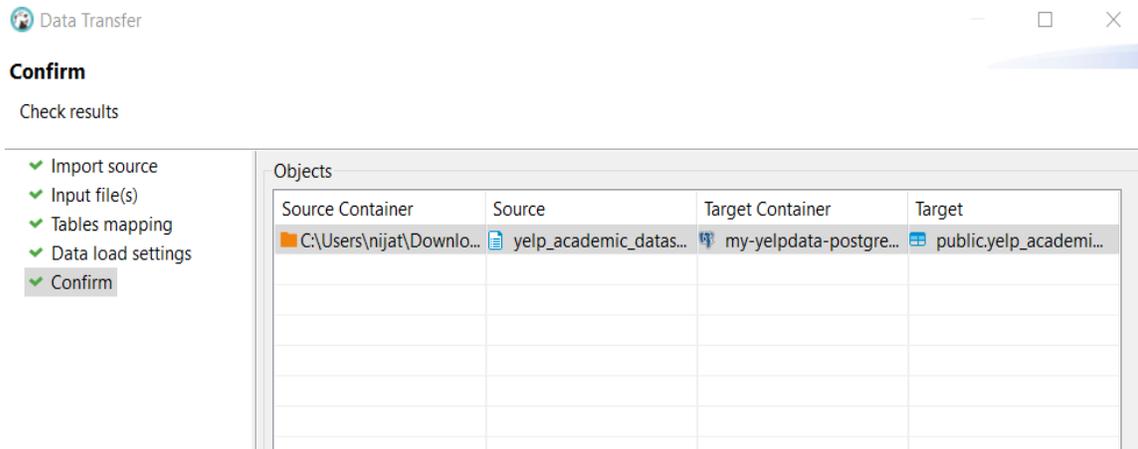


Figure 14 JSON to CSV

The result of this experiment reveals that there are a variety of approaches that may be used to upload files to the database effectively. Having said that, it is essential to point out that ClickHouse offers a highly effective solution, which is especially visible in the extremely short uploading times that it makes possible. It is important to note that ClickHouse simplifies the process by automatically generating and moving data to the proper tables. This eliminates the necessity for manually constructing tables for each JSON format. During the course of this trial, ClickHouse was able to do this job in a fraction of a second. One of the benefits that ClickHouse provides is the ability to handle enormous datasets with low overhead, and this efficiency highlights that advantage.

Let us take into consideration the hypothetical situation in which the company needs to do data analytics based on the big dataset that Yelp provides.

Scenario 1: Identifying the Top Reviewed Businesses

Requirement

The specific requirement for Scenario 1 involves finding the top businesses with the highest average review ratings, ordered from highest to lowest.

Results

PostgreSQL Database:

Response Time: 2.40 seconds

Response Size: 14.31MB

ClickHouse Database:

Response Time: 495 milliseconds

Response Size: 14.31MB

Scenario 2: Weekly Review Distribution

Requirement

The analytical task for Scenario 2 entails analyzing the distribution of reviews across different days of the week.

Results

PostgreSQL Database:

Response Time: 1878 milliseconds (approximately 2 seconds)

ClickHouse Database:

Response Time: 266 milliseconds

Scenario 3: Check-in Analysis

Requirement

The analytical task for Scenario 3 involves analyzing the frequency of check-ins for different businesses.

Results

PostgreSQL Database:

Response Time: 2.24 seconds

Records Size: 11.16MB

ClickHouse Database:

Response Time: 716 milliseconds (about half second)

Records Size: 11.16MB

Scenario 4: User Engagement Analysis

Requirement

The analytical task for Scenario 4 involves analyzing user engagement by identifying users who write reviews frequently and have many fans.

Results

PostgreSQL Database:

Response Time: 1890 milliseconds (about 2 seconds)

ClickHouse Database:

Response Time: 347 milliseconds

Main outcome

It is widely acknowledged that the use of relational database management systems (RDBMS), such as PostgreSQL, is indispensable in the arena of operational databases. The higher performance of PostgreSQL in operations involving insertions, deletions, and updates makes it the best option for transactional workloads. This is the reason it is recommended. Integration of the ClickHouse database is recommended, however, when one is presented with an environment that is significantly dependent on analytical queries.

The system's performance may deteriorate when analytical operations are performed on it, despite PostgreSQL being effective when dealing with transactional workloads. Utilizing PostgreSQL as the operational database for CRUD operations and transactions is a suggested method in this circumstance. This strategy takes use of the features that PostgreSQL has. Concurrently, it may be leveraged as a fundamental data source for future analytical queries that may be conducted. Incorporating a columnar database structure, such as ClickHouse, is recommended as a means of further enhancing the effectiveness of analytical queries.

This approach to dual databases takes use of the characteristics of both systems, ensuring that both transactional and analytical workloads are handled with the highest possible level of performance. ClickHouse, with its columnar storage architecture, specializes in handling analytical queries, and

as a result, it provides a well-rounded and efficient solution for a variety of operational database needs. PostgreSQL, however, acts as a dependable workhorse for daily operations.

Experiment 3

We have experimented to test the feasibility of putting entries into both databases using the Java spring boot program. As part of this experiment, I will do bulk insertions of one million, ten thousand, one thousand, and one million records. For determining the average number of insertions for each sample, each experiment is repeated five times.

- **Postgres DB:**

1 record: 38.12 milliseconds

100 records: 193.81 milliseconds

10000 records: 3.2 seconds

100000 records: 28 seconds

1000000 records: it took some time, and it failed. Considering I am trying to insert bulk insertion it failed. If I divide it into small numbers of bulk, then it will work but it is not needed for qualification.

- **ClickHouse DB:**

1 record: 35.1 milliseconds

100 records: 174.6 milliseconds

10000 records: 2.95 seconds

100000 records: 26.9 seconds

1000000 records: it took some time, and it failed. Seems we have some connection problems. Both databases failed during insertion 1million records at once. We can reduce it using batch size to 100000 records then it will help in managing memory efficiently and prevents issues like running out of memory for large datasets. If we follow it changes are synchronized with the database, and the commit makes the changes permanent.

This graph is represented in logarithmic scale due to date range so it can illustrate time complexity for different number of records. Table 2 shows exact numbers in detail.

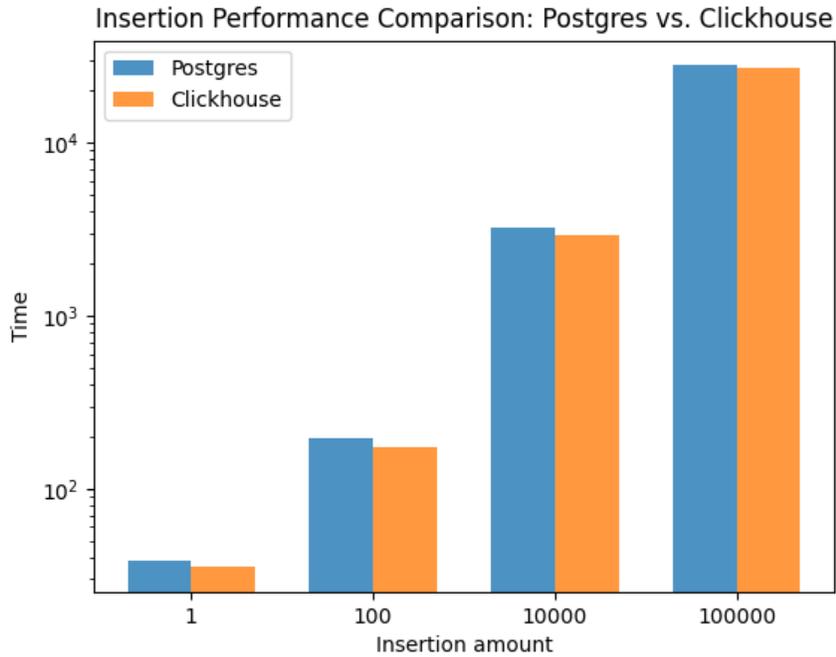


Figure 15 Insertion Performance Comparison

Table 2 Performance Comparison

	Record	Time
Postgres DB	1	38.12 milliseconds
	100	193.81 milliseconds
	10000	3.2 seconds
	100000	28 seconds
	1000000	Took Time and Failed
ClickHouse DB	1	35.1 milliseconds
	100	174.6 milliseconds
	10000	2.95 seconds

	100000	26.9 seconds
	1000000	Took Time and Failed

Experiment 4

The first three experiments demonstrate that ClickHouse is superior to Postgres DB in terms of performance. In the final experiment, we may consider a business case in the following manner: We want to evaluate which database works better in a case where numerous operations are being carried out concurrently. Imagine this scenario. Creating an endpoint that allows us to input results, update them in parallel, and then remove the results from both databases for testing purposes is one way to do this.

The implementation of a synchronized thread that a transaction will follow must be done before we can proceed with the experiment. It is quite easy for me to implement the ACID transaction principles in an application by using Spring Boot connection with the PostgreSQL database. PostgreSQL provides comprehensive support for these principles.

The result will be processing one hundred records concurrently, including the operations of inserting, updating, and deleting records simultaneously. Note that this is not a bulk insertion because of the transactional process, and that it will take place synchronously. This is a key point to keep in mind.

Result was Postgres: 1.32 Seconds

Following the experiment, it appears that ClickHouse is not proficient in handling transactions, particularly when adhering to ACID principles.

The fact that ClickHouse does not comply with ACID standards indicates that any data manipulations or updates made in the background do not ensure that they will be finished. Because there is no transaction isolation, every SELECT query that accesses data during an UPDATE or DELETE update will get the most recent data in each component. If a delete operation has only modified fifty percent of the components for a column, for example, queries would get obsolete information from the remaining components that have not been completely processed.

This restriction applies to all of the data that is kept in ClickHouse, including not just big tables with an analytical emphasis, such as those that deal with time-series data, but also the metadata that is linked with those tables. Business-centric metadata tables, on the other hand, are subject to

alterations and updates over time, in contrast to time-series data, which is typically insert-only and seldom modified.

MergeTree tables are used to store linked business data in ClickHouse for the purpose of performing elaborate joins and doing more in-depth analyses. This is true regardless of the data type (or its variant). As a result, whenever there is a revision, updates or deletions need a full rewrite (via the ALTER TABLE command).

Within ClickHouse, I came into a certain situation that was facilitated by a Java program. Without the implementation of a way to examine each process on its own, the system ran the danger of crashing. Data that was incorrect was deleted from the database, and the corrected prices did not correspond. After these difficulties were resolved from the Java application side, the system operated as expected. Spring's capabilities for transactions and synchrony were included in the system.

However, **it is noteworthy that the result in ClickHouse took 3.87 seconds.**

PostgreSQL is the database management system that should be used for operational activities, according to this experience. When it comes to ClickHouse, operational chores are time-consuming and provide issues in terms of creating resilience across numerous operations and preserving transactional integrity.

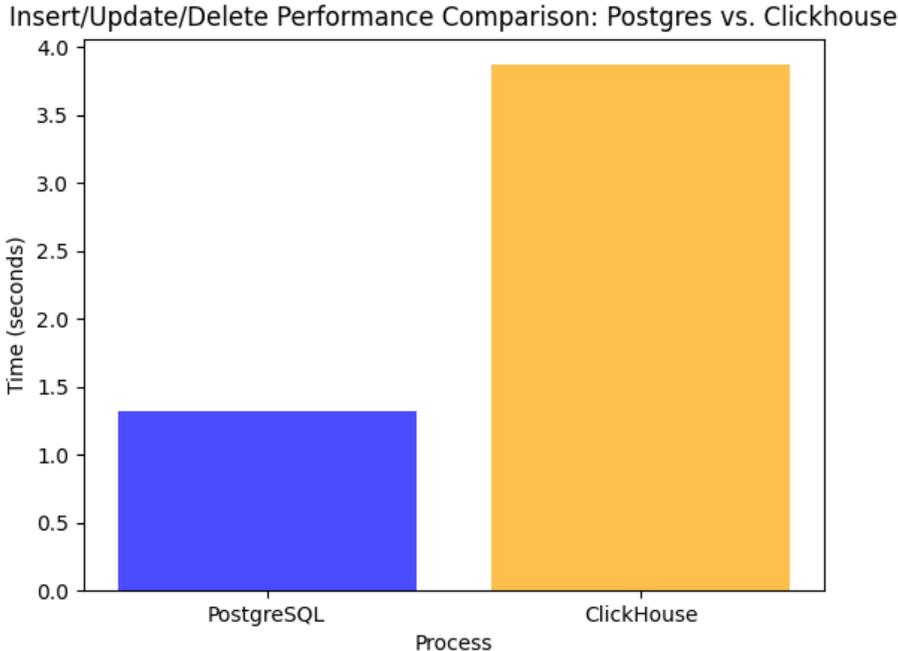


Figure 16 Insert/Update/Delete Performance Comparison

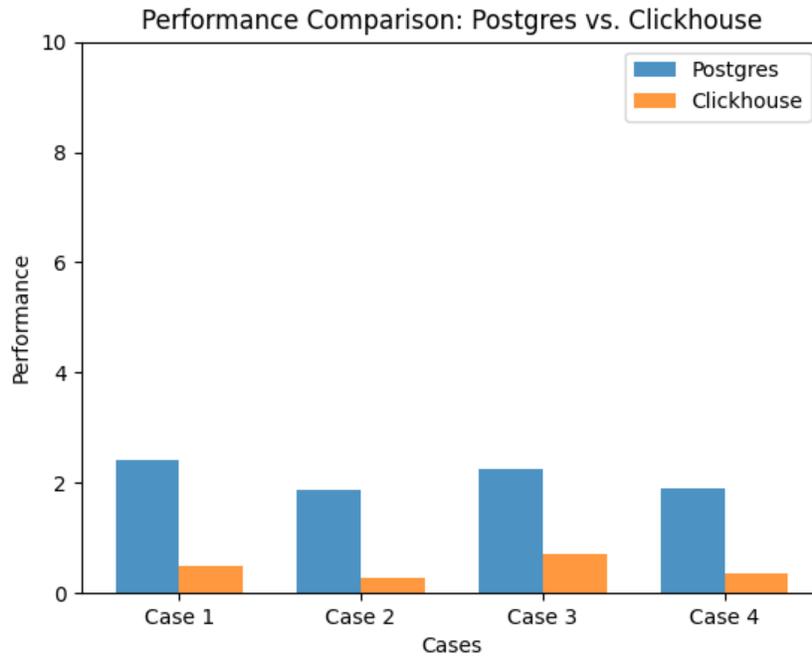


Figure 17 Performance Comparison

3.2 Results

During the experiment involving bulk insertion, both PostgreSQL and ClickHouse encountered difficulties with varied scales of record insertions. These difficulties were made possible by an application written in Java to leverage Spring Boot. It is important to note that both databases had challenges when confronted with the enormous magnitude of one million entries. ClickHouse demonstrated a need to modify batch sizes to maximize memory use. The graphical depiction of the findings on a logarithmic scale illustrated the temporal complexity for various numbers of records. This highlighted the difficulties connected with managing enormous datasets.

By simulating simultaneous insert, update, and delete actions on one hundred records, experiment 4 aims to evaluate the transactional capabilities of both PostgreSQL and ClickHouse. Within 1.32 seconds, PostgreSQL, well-known for its extensive support for ACID transactions, demonstrated its efficiency by successfully finishing the operation. ClickHouse, which did not comply with ACID standards, required 3.87 seconds to complete comparable processes. The results light the striking differences between the transactional behaviors of PostgreSQL and ClickHouse, which is essential for decision-making when the integrity of transactions is of the utmost importance.

Performance Analysis:

A comprehensive study of several different situations was carried out during the performance analysis phase. This investigation considered ACID compliance, transactional integrity, and managing huge datasets effectively. PostgreSQL demonstrated its strength regarding operational activities, notably in situations that included insertions, removals, and updates. In scenarios that required a lot of analytical queries, ClickHouse performed quite well, proving its ability to manage massive datasets with a low amount of overhead.

Use Case Scenarios:

Several hypothetical use case situations were investigated to determine which database would be most suitable for certain applications. Due to its superior performance in transactional workloads, PostgreSQL has become the database of choice for operational procedures required to comply with ACID standards. It was advised that analytical searches and bulk insertions be performed using ClickHouse because of its improved performance in situations involving huge datasets.

Analytical Queries Performance:

Several scenarios were used to assess the performance of analytical queries. These scenarios included determining which companies received the highest number of reviews, studying the distribution of reviews, analyzing the frequency of check-ins, and analyzing user interaction. Compared to PostgreSQL, ClickHouse regularly displayed much quicker response times in the above instances. The findings highlighted the benefits that ClickHouse offers in managing complicated analytical queries over excessively big datasets, therefore establishing it as an option suited for use in applications related to data analytics and business intelligence. A comprehensive picture of how PostgreSQL and ClickHouse work in various settings is provided by carefully evaluating the findings across all of these studies. When making educated judgments regarding selecting a database system based on the requirements of an application, these insights are helpful sources of information. Next, the following part digs into the documenting and reporting of the whole experimental procedure, with the goal of assuring transparency and making it easier to replicate the experiment.

3.3 Discussion

The analysis of the findings and the consequences of those results sheds light on the performance of PostgreSQL and ClickHouse in various operational and analytical settings, as well as their

strengths and limits. When it comes to managing massive volumes of data, both databases' difficulties during the bulk insertion experiment shed insight on the situation's complexity. PostgreSQL and ClickHouse had problems while working with a million records, prompting them to speculate on the best batch sizes to utilize to make the most effective use of memory. Because of its sensitivity to memory management during large-scale data processing, ClickHouse requires batch size modifications. This highlights the need for memory management. This understanding is essential for database administrators attempting to fine-tune performance settings for certain workloads.

During the fourth experiment, it was discovered that PostgreSQL and ClickHouse exhibit different transactional behaviors. PostgreSQL, which complies with the ACID standard, has shown its effectiveness in managing simultaneous insert, update, and delete operations. In contrast, ClickHouse had difficulties, notably in preserving transactional integrity across several different processes, since it did not comply with ACID standards. The significance of considering the characteristics of transactions when selecting a database system is highlighted by this discovery. This is particularly true when maintaining transactional consistency is of the utmost importance. The comprehensive performance analysis shows how each database suits job responsibilities. The power that PostgreSQL has in operational activities, shown by its ability to perform insertions, deletions, and updates effectively, places it as a reliable option for transactional workloads. ClickHouse's exceptional performance in managing large datasets and complex analytical queries aligns with its role as a specialized database for data analytics and business intelligence. It becomes clear that there is a trade-off between the efficiency of analytical queries and the consistency of transactions, which helps practitioners align their choice with the application's requirements.

Through investigating hypothetical use case situations, practitioners are given suggestions that may be implemented. The advice that PostgreSQL is used for operational procedures that need ACID compliance aligns with the database management system's historical strength in transactional workloads. ClickHouse's advice for analytical queries and bulk insertions reflects the company's design emphasis on analytics over big datasets. Having a detailed grasp of each database's strengths enables strategic decision-making based on the application's unique requirements, hence avoiding universally applicable techniques.

The fact that ClickHouse has consistently shown faster response times in analytical settings proves the company's ability to handle complicated queries across vast datasets. ClickHouse beat

PostgreSQL in various tasks, including determining which companies had the highest ratings and measuring user interaction. As a result, it provides an appealing alternative for applications that emphasize analytics and data-driven decision-making.

PostgreSQL and ClickHouse have their own unique performance characteristics, which are encapsulated in this debate. The application's individual requirements should be considered when selecting a database. This includes whether the application prioritizes transactional consistency or excels in analytics across huge datasets. Next, the following part digs into the documenting and reporting of the experimental process, with the goal of maintaining transparency and making it easier to replicate the procedures.

4 Conclusions

This comprehensive study delves into the comparative analysis of two prominent database management systems, PostgreSQL and ClickHouse, with a focus on their performance in handling large datasets and diverse operational scenarios. The experimentation process involved a meticulous exploration of various aspects, ranging from database initialization and dataset creation to bulk insertions, simultaneous processes, and analytical query performance. The obtained results provide valuable insights into the strengths and limitations of each database, aiding in informed decision-making for specific application requirements.

The initialization phase highlighted the ease of setting up both databases using Docker containers, ensuring a standardized environment for fair evaluation. The creation of datasets, particularly the challenges faced in PostgreSQL due to limitations in handling JSON files, emphasized the practical considerations necessary for efficient data management.

Bulk insertion experiments uncovered the intricacies of managing substantial data volumes. Both PostgreSQL and ClickHouse faced challenges with a million-record scale, necessitating thoughtful adjustments to batch sizes for optimal memory usage. ClickHouse's sensitivity to memory management became evident, guiding administrators in fine-tuning performance parameters for large-scale operations.

Experiment 4, focusing on simultaneous processes, elucidated the divergent transactional behaviors between the two databases. PostgreSQL, with its robust ACID compliance, excelled in handling concurrent insert, update, and delete operations, providing a reliable option for transactional workloads. In contrast, ClickHouse, lacking ACID compliance, faced difficulties in

maintaining transactional integrity across multiple processes, indicating its specialization in analytics rather than transactional consistency.

The performance analysis underscored the suitability of each database for specific tasks. PostgreSQL's efficiency in operational processes, exemplified in swift insertions, deletions, and updates, positions it as a robust choice for transactional workloads. ClickHouse's exceptional performance in analytical scenarios, consistently outperforming PostgreSQL in response times, solidifies its role as a specialized database for data analytics and business intelligence.

Hypothetical use case scenarios offered practical guidance for practitioners, recommending PostgreSQL for ACID-compliant operational tasks and ClickHouse for analytics and bulk insertions over extensive datasets. The nuanced understanding of each database's strengths enables strategic decision-making aligned with the unique demands of diverse applications.

The documentation and reporting processes ensure transparency and replicability, contributing to the study's credibility. ER diagrams visually represent the database structures, and visual representations of performance metrics enhance the comprehensibility of the findings.

This research provides a holistic view of the performance characteristics of PostgreSQL and ClickHouse, facilitating informed decisions in selecting a database system tailored to specific application needs. The nuanced understanding of transactional consistency versus analytic query efficiency enables practitioners to make strategic choices aligned with their unique requirements.

References

- [1] Z. M. Jiang, J. J. Bai, and Z. Su, “DynSQL: Stateful Fuzzing for Database Management Systems with Complex and Valid SQL Query Generation,” *32nd USENIX Secur. Symp. USENIX Secur. 2023*, vol. 7, pp. 4949–4966, 2023.
- [2] S. Kanani, “0110 A method to evaluate database management systems for Big Data focus on spatial data,” 2019.
- [3] O. Dmitrijev, “Selection Criteria of the Database Management System for Estonian Small and Medium Sized Enterprises,” 2015.
- [4] S. Haque, Z. Eberhart, A. Bansal, and C. McMillan, “Semantic Similarity Metrics for Evaluating Source Code Summarization,” *IEEE Int. Conf. Progr. Compr.*, vol. 2022-March, pp. 36–47, 2022, doi: 10.1145/nnnnnnn.nnnnnnn.
- [5] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, “Automatic database management system tuning through large-scale machine learning,” *Proc. ACM SIGMOD Int. Conf. Manag. Data*, vol. Part F127746, pp. 1009–1024, 2017, doi: 10.1145/3035918.3064029.
- [6] Y. Hajjaji, W. Boulila, I. R. Farah, I. Romdhani, and A. Hussain, “Big data and IoT-based applications in smart environments: A systematic review,” *Comput. Sci. Rev.*, vol. 39, p. 100318, 2021, doi: 10.1016/j.cosrev.2020.100318.
- [7] S. S. Kamble and A. Gunasekaran, “Big data-driven supply chain performance measurement system: a review and framework for implementation,” *Int. J. Prod. Res.*, vol. 58, no. 1, pp. 65–86, 2020, doi: 10.1080/00207543.2019.1630770.
- [8] P. Kraft *et al.*, “Apiary: A DBMS-Integrated Transactional Function-as-a-Service Framework,” 2022, [Online]. Available: <http://arxiv.org/abs/2208.13068>.
- [9] A. Agrawal *et al.*, “Cloudy with High Chance of DBMS: A 10-year Prediction for Enterprise-Grade ML,” *10th Annu. Conf. Innov. Data Syst. Res. CIDR 2020*, 2020.
- [10] B. W. Yogatama, W. Gong, and X. Yu, “Orchestrating Data Placement and Query Execution in Heterogeneous CPU-GPU DBMS,” *Proc. VLDB Endow.*, vol. 15, no. 11, pp. 2491–2503, 2022, doi: 10.14778/3551793.3551809.
- [11] J. Tan *et al.*, “Choosing a cloud DBMS: Architectures and tradeoffs,” *Proc. VLDB Endow.*, vol. 12, no. 12, pp. 2170–2182, 2018, doi: 10.14778/3352063.3352133.
- [12] M. Zymbler, Y. Kraeva, A. Grents, A. Perkova, and S. Kumar, “An approach to fuzzy

- clustering of big data inside a parallel relational DBMS,” *Commun. Comput. Inf. Sci.*, vol. 1223 CCIS, no. July, pp. 211–223, 2020, doi: 10.1007/978-3-030-51913-1_14.
- [13] Y. Han *et al.*, “Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation,” *Proc. VLDB Endow.*, vol. 15, no. 4, pp. 752–765, 2021, doi: 10.14778/3503585.3503586.
- [14] J. J. Pan, J. Wang, and G. Li, “Survey of Vector Database Management Systems,” 2023, [Online]. Available: <http://arxiv.org/abs/2310.14021>.
- [15] A. Bernhardt *et al.*, “neoDBMS: In-situ Snapshots for Multi-Version DBMS on Native Computational Storage,” *Proc. - Int. Conf. Data Eng.*, vol. 2022-May, no. iii, pp. 3170–3173, 2022, doi: 10.1109/ICDE53745.2022.00290.
- [16] M. Thesis, “Architecting SQL / PGQ support in DuckDB,” 2022.
- [17] F. Gerlinghoff, “Master ’ s Thesis A Testing Strategy for Hybrid Query Execution in Database Management Systems Author :,” 2023.
- [18] D. F. Systems, “Confrencea – Virtual International Conferences Confrencea – Virtual International Conferences,” no. November, pp. 126–131, 2023.
- [19] D. Francisco and P. Junior, “Defining Metrics for the Identification of Microservices in Code Repositories Defining Metrics for the Identification of Microservices in,” 2023.
- [20] T. Zois, “Hermes : Query-driven adaptive replication Master Thesis Hermes : Query-driven adaptive replication Author : Theodoros Zois,” no. September, 2021, doi: 10.13140/RG.2.2.13449.98404.
- [21] A. Kohn, D. Moritz, and T. Neumann, “DashQL -- Complete Analysis Workflows with SQL,” 2023, [Online]. Available: <http://arxiv.org/abs/2306.03714>.
- [22] M. Wawrzoniak, R. Bruno, A. Klimovic, and G. Alonso, “Ephemeral Per-query Engines for Serverless Analytics,” *CEUR Workshop Proc.*, vol. 3462, 2023.
- [23] D. Miguel and D. A. Silveira, “BSc in Business Administration LEAN DATA ENGINEERING,” 2022.
- [24] P. K. Erdelt and J. Jestel, “DBMS-Benchmarker: Benchmark and Evaluate DBMS in Python,” *J. Open Source Softw.*, vol. 7, no. 79, p. 4628, 2022, doi: 10.21105/joss.04628.
- [25] O. Falg en, “Cardinality estimation with a machine learning approach,” 2020, [Online]. Available: www.kth.se/sci.
- [26] A. Dragoni, “A model-based methodology to de- fine data-intensive architectures Author :

- Arianna Dragoni,” 2022.
- [27] E. Zeydan and J. Mangues-Bafalluy, “Recent Advances in Data Engineering for Networking,” *IEEE Access*, vol. 10, no. D1, pp. 34449–34496, 2022, doi: 10.1109/ACCESS.2022.3162863.
- [28] O. Kolbina, E. Istomin, N. Yagotinceva, T. Safonova, and M. Kalambet, “Peculiarities of creating a database for the IoT system of urban forest management in the city of St. Petersburg,” *IOP Conf. Ser. Earth Environ. Sci.*, vol. 876, no. 1, 2021, doi: 10.1088/1755-1315/876/1/012039.
- [29] R. Li *et al.*, “Apache ShardingSphere: A Holistic and Pluggable Platform for Data Sharding,” *Proc. - Int. Conf. Data Eng.*, vol. 2022-May, pp. 2468–2480, 2022, doi: 10.1109/ICDE53745.2022.00230.
- [30] P. A. Astanin, S. E. Rauzina, and T. V. Zarubina, “Digital Tools in UMLS Metathesaurus Knowledge Processing,” *Stud. Health Technol. Inform.*, vol. 305, pp. 186–189, 2023, doi: 10.3233/SHTI230458.
- [31] E. Guliyev, “Comparative Analysis of Multi-model Databases,” 2022.
- [32] B. McBride and D. Reynolds, “Survey of Time Series Database Technology,” 2020.
- [33] M. Fernanda, M. Helena, P. De Jesus, S. Marlene, and M. Oliveira, “Instituto Superior de Gestão e Administração de Santarém,” 2022.
- [34] J. Huang, D. Miao, and X. Liu, “QUEST: An Efficient Query Evaluation Scheme Towards Scan-Intensive Cross-Model Analysis,” 2023, [Online]. Available: <http://arxiv.org/abs/2309.11860>.
- [35] B. S. C. Science, C. Science, and C. Science, “Increasing DoS-Resilience for Cross-Protocol Proxies,” no. 2020, 2022.
- [36] J. Luettgau, G. Scorzelli, V. Pascucci, G. Tarcea, C. R. Kirkpatrick, and M. Taufer, “NSDF-Catalog: Lightweight Indexing Service for Democratizing Data Delivery,” *Proc. - 2022 IEEE/ACM 15th Int. Conf. Util. Cloud Comput. UCC 2022*, pp. 1–10, 2022, doi: 10.1109/UCC56403.2022.00011.
- [37] D. Patra, “High-Performance Database Management System Design for Efficient Query Scheduling,” no. December, 2022.
- [38] M. Thesis and C. C. Felius, “Assessing the performance of distributed PostgreSQL,” 2022.
- [39] H. Gavriilidis, L. Rose, J. Ziegler, K. Beedkar, J. A. Quiané-Ruiz, and V. Markl, “XDB in

- Action: Decentralized Cross-Database Query Processing for Black-Box DBMSes,” *Proc. VLDB Endow.*, vol. 16, no. 12, pp. 4078–4081, 2023, doi: 10.14778/3611540.3611625.
- [40] A. Egorov, D. Alexandrov, and N. Butakov, “Towards a Toolbox for Mining QA-pairs and QAT-triplets from Conversational Data of Public Chats,” *Conf. Open Innov. Assoc. Fruct.*, vol. 2021-May, pp. 94–101, 2021, doi: 10.23919/FRUCT52173.2021.9435511.
- [41] J. Mostafa, S. Wehbi, S. Chilingaryan, and A. Kopmann, *SciTS: A Benchmark for Time-Series Databases in Scientific Experiments and Industrial Internet of Things*, vol. 1, no. 1. Association for Computing Machinery, 2022.
- [42] M. J. Awan *et al.*, “Real-time ddos attack detection system using big data approach,” *Sustain.*, vol. 13, no. 19, pp. 1–19, 2021, doi: 10.3390/su131910743.
- [43] S. Rani, M. Chauhan, A. Kataria, and A. Khang, “IoT Equipped Intelligent Distributed Framework for Smart Healthcare Systems,” *Stud. Big Data*, vol. 137, pp. 97–114, 2023, doi: 10.1007/978-981-99-6034-7_6.
- [44] A. Bendimerad, Y. Remil, R. Mathonat, and M. Kaytoue, “On-Premise AIOps Infrastructure for a Software Editor SME: An Experience Report,” 2023, [Online]. Available: <http://arxiv.org/abs/2308.11225>.
- [45] K. Malanchev *et al.*, “The SNAD Viewer: Everything You Want to Know about Your Favorite ZTF Object,” *Publ. Astron. Soc. Pacific*, vol. 135, no. 1044, p. 0, 2023, doi: 10.1088/1538-3873/acb292.
- [46] D. Tang and T. H. Wang, “BitUP: Efficient Bitmap Data Storage Solution For User Profile.”
- [47] A. A. Visheratin *et al.*, “Peregreen - Modular database for efficient storage of historical time series in cloud environments,” *Proc. 2020 USENIX Annu. Tech. Conf. ATC 2020*, pp. 589–601, 2020.
- [48] F. Barez, P. Bilokon, and R. Xiong, “Benchmarking Specialized Databases for High-frequency Data,” *SSRN Electron. J.*, pp. 1–29, 2023, doi: 10.2139/ssrn.4342004.
- [49] M. Pasumansky and B. Wagner, “Assembling a Query Engine From Spare Parts,” *Others*, [Online]. Available: <https://github.com/google/zetasql>.
- [50] S. Chen *et al.*, “Large Vector Spatial Data Storage and Query Processing Using Clickhouse,” *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. - ISPRS Arch.*, vol. 48, no. M-1–2023, pp. 65–72, 2023, doi: 10.5194/isprs-archives-XLVIII-M-1-2023-65-2023.

- [51] A. Farhat, S. Delaughter, and K. Sollins, “Measuring and Analyzing DoS Flooding Experiments,” *ACM Int. Conf. Proceeding Ser.*, vol. i, pp. 81–90, 2022, doi: 10.1145/3546096.3546105.
- [52] M. Gouel *et al.*, “Towards a Publicly Available Framework to Process Traceroutes with MetaTrace To cite this version : HAL Id : hal-04198068 Towards a Publicly Available Framework to Process Traceroutes with MetaTrace,” 2023.
- [53] M. Besta *et al.*, “Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries,” *ACM Comput. Surv.*, vol. 56, no. 2, pp. 1–46, 2023, doi: 10.1145/3604932.
- [54] R. Rialti, L. Zollo, A. Ferraris, and I. Alon, “Big data analytics capabilities and performance: Evidence from a moderated multi-mediation model,” *Technol. Forecast. Soc. Change*, vol. 149, no. November, 2019, doi: 10.1016/j.techfore.2019.119781.
- [55] R. Sahal, J. G. Breslin, and M. I. Ali, “Big data and stream processing platforms for Industry 4.0 requirements mapping for a predictive maintenance use case,” *J. Manuf. Syst.*, vol. 54, no. January, pp. 138–151, 2020, doi: 10.1016/j.jmsy.2019.11.004.
- [56] L. Sun *et al.*, “Review of Bridge Structural Health Monitoring Aided by Big Data and Artificial Intelligence : From Condition Assessment to Damage Detection,” vol. 146, no. 5, 2020, doi: 10.1061/(ASCE)ST.1943-541X.0002535.
- [57] K. M. Jablonka, D. Ongari, S. M. Moosavi, and B. Smit, “Big-Data Science in Porous Materials: Materials Genomics and Machine Learning,” *Chem. Rev.*, vol. 120, no. 16, pp. 8066–8129, 2020, doi: 10.1021/acs.chemrev.0c00004.
- [58] G. Dicuonzo, G. Galeone, E. Zappimulso, and V. Dell’Atti, “Risk Management 4.0: the Role of Big Data Analytics in the Bank Sector,” *Int. J. Econ. Financ. Issues*, vol. 9, no. 6, pp. 40–47, 2019, doi: 10.32479/ijefi.8556.
- [59] L. Thostrup, D. Failing, T. Ziegler, and C. Binnig, “A DBMS-centric Evaluation of BlueField DPUs on Fast Networks.”
- [60] N. Boeschen and C. Binnig, *GaccO-A GPU-accelerated OLTP DBMS*, vol. 1, no. 1. Association for Computing Machinery, 2022.
- [61] Q. Ma and P. Triantafillou, “Query-centric regression for in-DBMS analytics,” *CEUR Workshop Proc.*, vol. 2572, pp. 16–25, 2020.
- [62] G. Haas, M. Haubenschild, and V. Leis, “Exploiting Directly-Attached NVMe Arrays in

- DBMS,” *10th Annu. Conf. Innov. Data Syst. Res. CIDR 2020*, 2020.
- [63] B. Haynes, M. Daum, A. Mazumdar, M. Balazinska, A. Cheung, and L. Ceze, “VisualWorldDB: A DBMS for the Visual World,” *10th Annu. Conf. Innov. Data Syst. Res. CIDR 2020*, 2020.
- [64] C. I. Dbms, L. Heinzl, B. Hurdelhey, M. Boissier, and M. Perscheid, “Evaluating Lightweight Integer Compression Algorithms in Column-Oriented In-Memory DBMS,” no. August 2021, 2021.
- [65] C. Chaubey and M. Kumar Nanda, “Cloud Database Management System Architecture,” *Int. J. Electr. Eng. Technol.*, vol. 11, no. 10, pp. 260–266, 2020, doi: 10.34218/IJEET.11.10.2020.036.
- [66] A. Bernhardt, S. Tamimi, F. Stock, T. Vinçon, A. Koch, and I. Petrov, “Cache-Coherent Shared Locking for Transactionally Consistent Updates in Near-Data Processing DBMS on Smart Storage,” no. i, pp. 424–428, 2022.
- [67] A. Mousa, M. Karabatak, and T. Mustafa, “Database Security Threats and Challenges,” *8th Int. Symp. Digit. Forensics Secur. ISDFS 2020*, no. July 2022, 2020, doi: 10.1109/ISDFS49300.2020.9116436.
- [68] P. K. Paul and P. S. Aithal, “Database Security: An Overview and Analysis of Current Trend,” *Int. J. Manag. Technol. Soc. Sci.*, vol. 4, no. 2, pp. 53–58, 2019, doi: 10.47992/ijmts.2581.6012.0070.
- [69] S. W. Harahap, A. Anisa, S. N. Pane, M. A. R. Purba, and Nurbaiti, “Database Management System PT Sierad Produce Tbk di Medan,” *J. Ilm. Sains Teknol. dan Inf.*, vol. 1, no. 3, pp. 20–26, 2023.
- [70] M. R. Anwar, R. Panjaitan, and R. Supriati, “Implementation Of Database Auditing By Synchronization DBMS,” *Int. J. Cyber IT Serv. Manag.*, vol. 1, no. 2, pp. 197–205, 2021, doi: 10.34306/ijcitsm.v1i2.53.

Appendix - Supplementary Evidence

This appendix includes additional visual evidence supporting the practical implementation aspects of my research. It contains screenshots and other relevant images demonstrating key steps in the setup and execution of database environments, API interactions and performance results.

1.

This diagram provides a visual representation of the entity-relationship model of the PostgreSQL database used in this study. It illustrates the database schema, including the tables, their respective fields, and the relationships between these tables. Key entities and their attributes are detailed, highlighting how data is structured within the database.

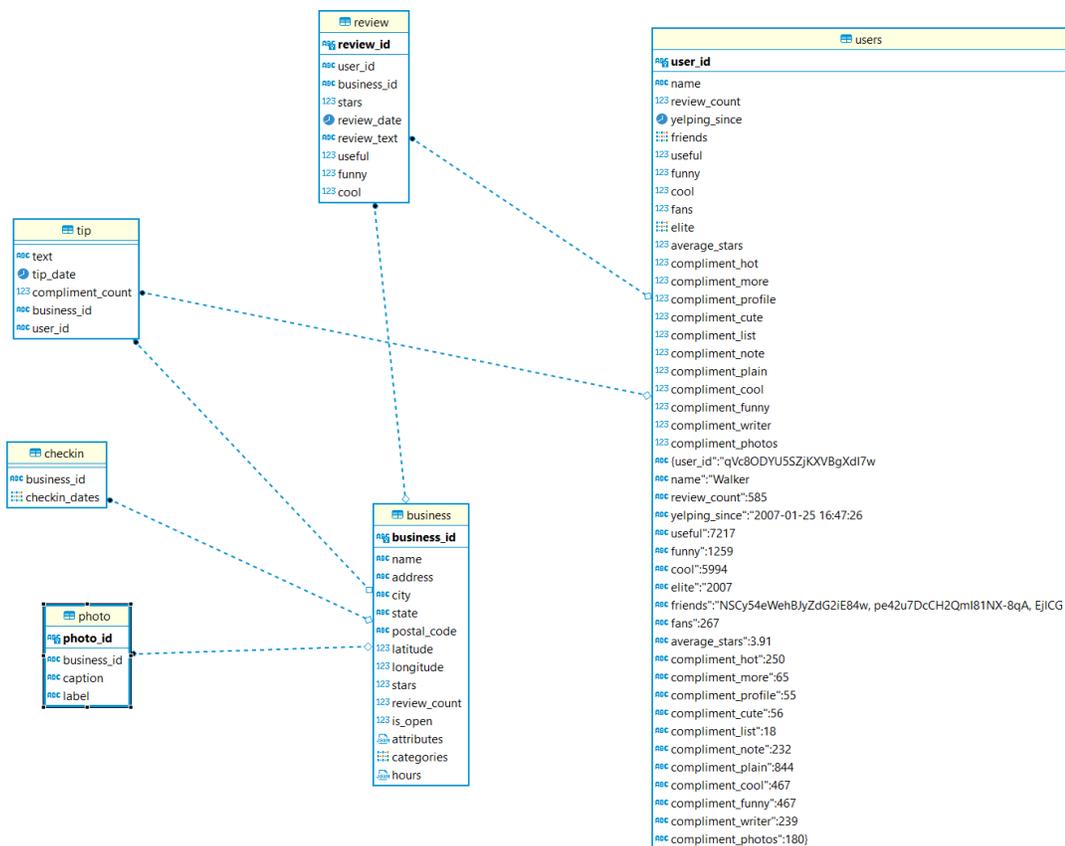


Figure 1 ER diagram for PostgreSQL

2.

This diagram provides a detailed structure of the ClickHouse database utilized in this study.

Given that ClickHouse is a column-oriented database management system and does not inherently support traditional relational database relationships, the diagram focuses on illustrating the schema, tables, and their respective columns. It clearly represents the data organization within the database, including the data types and any indexing strategies employed.

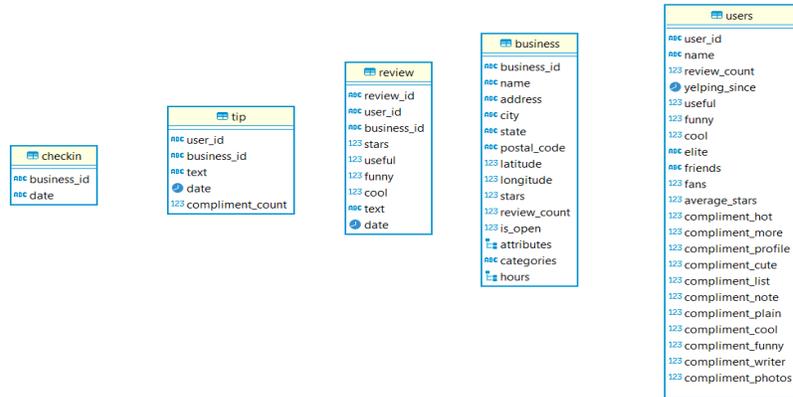


Figure 2 ER diagram for ClickHouse

3.

This figure illustrates the Docker Desktop environment used in the study, highlighting the graphical interface and the setup of containers, images, and other Docker resources

critical to the research workflow.

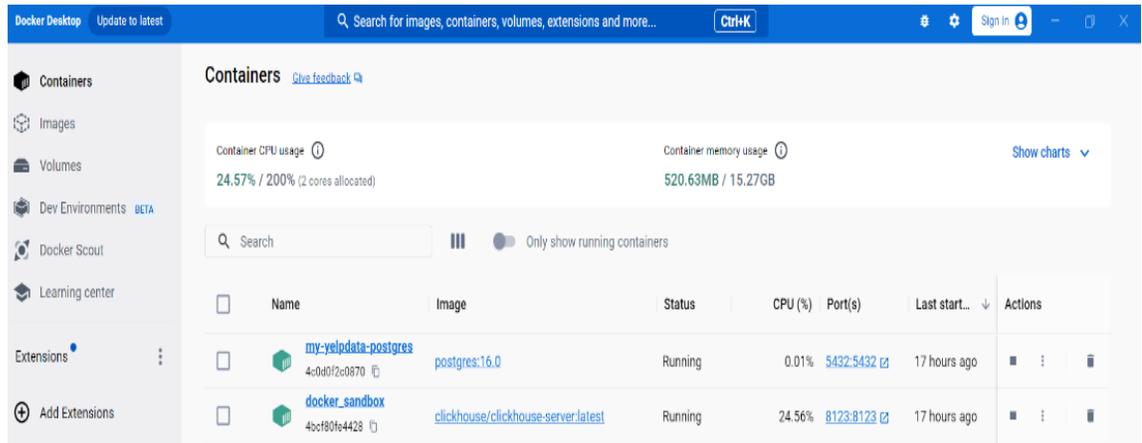


Figure 3 Docker Desktop Environment

4.

This figure displays the structure of the 'User' table within the database, detailing its columns, data types, and any key constraints. It provides an overview of how user data is created, organized, and stored.

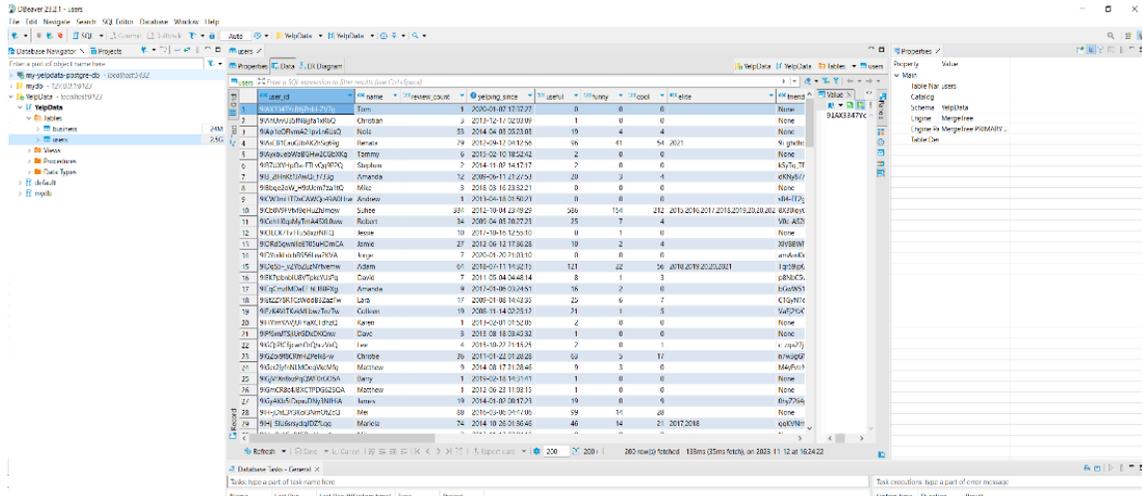


Figure 4 User Table

5.

This figure illustrates the process of parsing JSON files and inserting their data into corresponding database tables using a script. It visually represents the proof of transformation and integration of data from JSON format into structured table formats

within the database

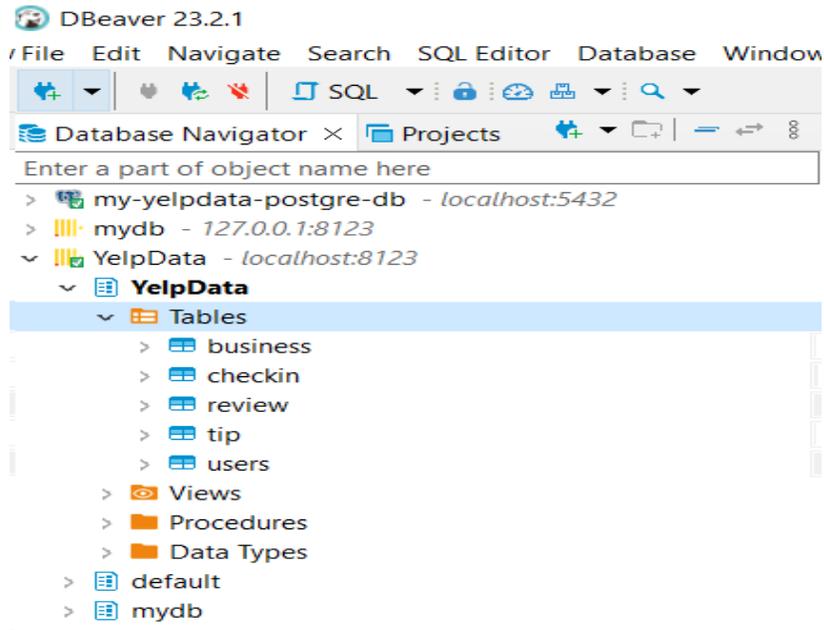


Figure 5 JSON Files to Corresponding Tables

6.

This figure provides evidence of successful data insertion into the database, achieved through RESTful requests via the REST API. It displays the request-response exchange, highlighting the API's input and the database's acknowledgment or confirmation of the

inserted records

YelpDataset / New Request Save Send

POST http://localhost:8080/v1/internal/file/upload?database=postgres Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	file	yelp_academic_dataset_user.json ×			
	Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 3 m 3.84 s Size: 189 B Save Response

Pretty Raw Preview Visualize Text 🔍

1 File uploaded to Postgres

Figure 6 PostgreSQL Database

7.

This figure displays the response time for a RESTful POST request to the PostgreSQL and ClickHouse database. It illustrates the time taken for the database to process the request and return a response, highlighting PostgreSQL's and ClickHouse performance

characteristics in handling such operations.

The screenshot shows a REST client interface with the following details:

- Request:** GET `http://localhost:8080/v1/internal/result?calculation=TopReviewed&database=postgres`
- Query Params:**

Key	Value	Description
calculation	TopReviewed	
database	postgres	
- Status:** 200 OK, Time: 2.40 s, Size: 14.31 MB
- Response Body (JSON):**

```

1 [{"business_id": "_akr7P0nacW_VizRkBPcIA",
2  "name": "Blues City Deli",
3  "stars": 5,
4  "review_count": "991"}
5 ]
6 [{"business_id": "8QqnRpM-QxGsJDNuu0E57A",
7  "name": "Carlillos Cocina",
8  "stars": 5,
9  "review_count": "799"}
10 ]
11 [{"business_id": "zxIF-bnaJ-eKIszn87yu7A",
12  "name": "Free Tours By Foot",
13  "stars": 5,
14  "review_count": "769"}
15 ]
16 ]

```

Figure 7 PostgreSQL Database Result

The screenshot shows a REST client interface with the following details:

- Request:** GET `http://localhost:8080/v1/internal/result?calculation=TopReviewed&database=clickhouse`
- Query Params:**

Key	Value	Description
calculation	TopReviewed	
database	clickhouse	
- Status:** 200 OK, Time: 495 ms, Size: 14.31 MB
- Response Body (JSON):**

```

1 [{"business_id": "_akr7P0nacW_VizRkBPcIA",
2  "name": "Blues City Deli",
3  "stars": 5,
4  "review_count": "991"}
5 ]
6 [{"business_id": "8QqnRpM-QxGsJDNuu0E57A",
7  "name": "Carlillos Cocina",
8  "stars": 5,
9  "review_count": "799"}
10 ]
11 [{"business_id": "zxIF-bnaJ-eKIszn87yu7A",
12  "name": "Free Tours By Foot",
13  "stars": 5,
14  "review_count": "769"}
15 ]
16 [{"business_id": "DVRJHvncpkqvY1anKroaMg",
17  "name": "Tunotico",
18  "stars": 5,
19  "review_count": "786"}
20 ]
21 [{"business_id": "gP_0M3yK2RocIs_GuzrWQ",
22  "name": "Yata",
23  "stars": 5,
24  "review_count": "623"}
25 ]

```

Figure 8 ClickHouse Database Result