Edu Komputika 13 (1) (2026)

Edu Komputika Journal

https://journal.unnes.ac.id/journals/edukom/

Designing Summary Tables to Optimize Performance in Relational Databases: An Empirical Approach

Edi Widodo[1)]✉, Kartika Imam Santoso[2)], and Wawan Setiawan[3)]

[1]Department of Information Sistem, Faculty of Information Technology and Communication, Universitas Semarang, Jl. Sukarno Hatta, Semarang, 50196, Indonesia

[2]Department of Computer Science, Universitas An Nuur, Jl. Gajah Mada No.7, Grobogan, 58112, Indonesia

[3]Department of Management, Faculty of Economy, Universitas Semarang, Jl. Sukarno Hatta, Semarang, 50196, Indonesia

Article Info

Article History:
Received: 16 May 2026
Revised: 10 August 2026
Accepted: 29 August 2026

Keywords:
Database Performance, RDBMS, Table Partitioning

Abstract

The performance of transactional applications involving databases will decline over time due to the increasing amount of stored data and the continuously rising volume of transactions. This research proposes a summary table design method that integrates table partitioning techniques and data aggregation to improve query efficiency without sacrificing the consistency and accuracy of the stored data. We tested this method using 30,275 transaction records on Google Cloud infrastructure and MySQL Server 8.0, demonstrating a data processing speed increase of up to 92.36% compared to conventional table designs. The research results show that this method not only accelerates data processing but also simplifies data management in relational database management systems (RDBMS). This method is relevant for transactional database applications with high transaction loads, such as the financial and e-commerce sectors. By combining the physical table design architecture and efficient query processing, this research contributes to the development of more scalable, reliable, and suitable database designs for high workload scenarios.

✉ Correspondence Address:
Jl. Sukarno Hatta, Semarang, 50196
E-mail: ediwidodo@usm.ac.id

ISSN 2252-6811
E-ISSN 2599-297X

INTRODUCTION

The Relational Database Management System (RDBMS) is crucial for supporting the ongoing performance of applications through its design structure. The evolution of relational data models and DBMS design techniques highlights the importance of shared databases (Y. T. Lee, 1999). Complex data models drive the design of these systems, providing efficiency in both time and space (Albert & Bozza, 2016). Although object-based technologies have become prevalent in software development, RDBMS remains the dominant technology (Loand & Hung, 2014). Various RDBMS-based applications have demonstrated the operational functionality of relational database systems, showcasing excellent scalability and performance capabilities (Albert & Bozza, 2016).

The decline in database application performance over time is a common issue, as databases continue to accumulate data and experience increased usage. The growing volume of data often causes this performance to drop, making it difficult for a single database and server to handle efficiently (Samidi et al., 2022). As databases evolve over time, the need for effective database management techniques becomes crucial to maintaining optimal performance. Database fragmentation plays an important role in distributed data management environments, helping to reduce costs and improve response times (Castro-Medina et al., 2022).

As databases grow, there is an increasing need to improve their design structure to ensure sustainable application performance (Acharya et al., 2020). Query optimization is a critical process in database management systems (DBMS) aimed at identifying the most efficient way to execute a query (Chandrashekariah & H. A., 2024; Azhir et al., 2021; Wang et al., 2014). A normalized table structure in an RDBMS enhances query speed by reducing data redundancy and ensuring data integrity. By organizing data into separate tables and defining relationships between them, normalization minimizes the amount of data processed during queries, leading to faster execution (Kumar & Azad, 2017; Sug, 2020). It also facilitates efficient data access, allowing quick retrieval without sifting through duplicate information. The challenge is how to design RDBMS structures and implement RDBMS queries that can address the issue of performance degradation in operational database applications after long-term use and when dealing with large capacities (Liu & Idreos, 2016).

In MySQL RDBMS, master tables and transaction tables serve different but complementary roles in database management. Master tables usually contain static or slowly changing data that defines entities in the database. People often use these tables to store reference data. For example, in an e-commerce system, a master table may store product information, customer details, or supplier data. Transaction tables use this relatively stable data to maintain data integrity and prevent redundancy (Mulyawan et al., 2020; Santosa et al., 2020). Transaction tables, on the other hand, record dynamic, frequently changing data representing business transactions or events. The design of these tables allows them to manage large volumes of insert, update, and delete operations. In an e-commerce system, for instance, the transaction table would store sales transactions, including details such as transaction date, product ID, customer ID, and quantity of goods sold. Real-time processing and analytics often utilize this data.

Master tables ensure data consistency and integrity by providing a single source of truth for reference data, which transaction tables can link to via foreign keys (Santosa et al., 2020). In the meantime, transaction tables optimize heavy write operations and facilitate real-time data processing through integration with other systems. For efficient data handling and analytics, you can use MySQL alongside NoSQL databases and Hadoop (Rodrigues et al., 2018), (Bt Wan Abu Bakar et al., 2015). In an e-commerce use case, the master table stores product and customer information, while the transaction table records sales and payment transactions (Mulyawan et al., 2020; Suryana et al., 2019). In another example, within the healthcare sector, the master table may store patient and provider information, while the transaction table handles healthcare service data exchanges (Kim et al., 2024).

Figure 1 illustrates how query optimization in MySQL plays a crucial role in enhancing the efficiency and performance of SQL queries. Efficient join processing is a fundamental aspect. MySQL can use various algorithms to optimize joints, such as sort-based algorithms, which have proven to be superior to traditional methods that don't use advanced techniques like parallel processing or hashing (Shah et al., 2018). Other techniques include cost-based optimization (A. Lee et al., 2009), window functions, and partitioning (Luo et al., 2019). MySQL can also leverage advanced optimization techniques such as hash-joins, sort-merge, and the use of distributed computing frameworks like SparkSQL, which have been shown to significantly reduce execution time (Chandrashekariah & H. A., 2024).

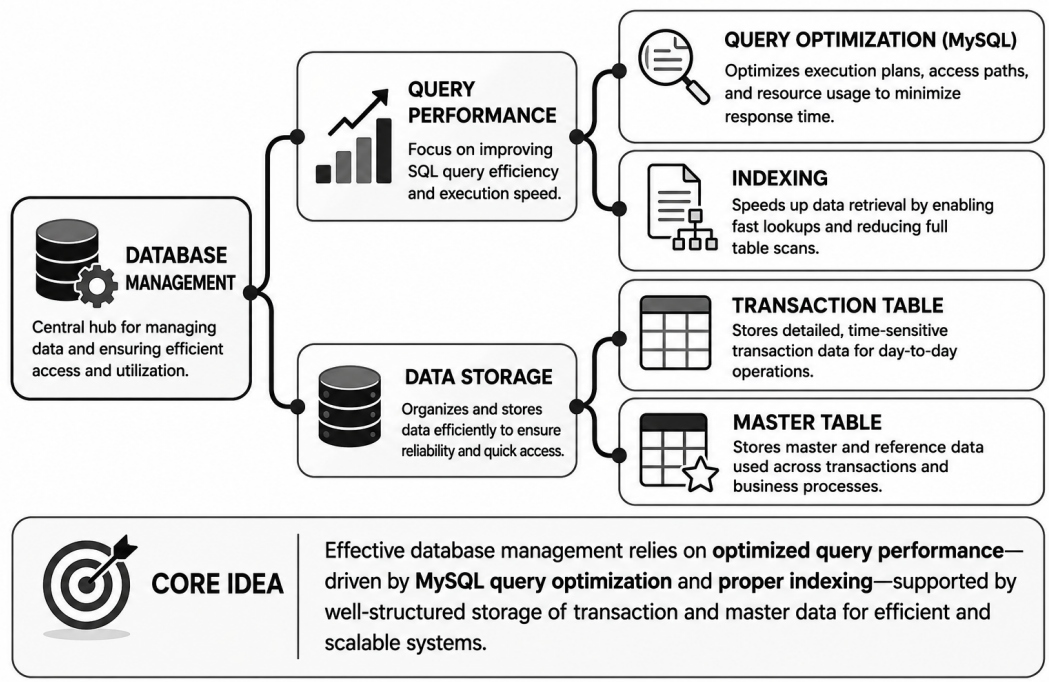

Figure 1. Database management concept map

Although query response times in single databases are still faster than in distributed systems, Samidi et al. (Samidi et al., 2022) showed that relational DBMS like MariaDB and MySQL can apply sharding, a technique more frequently used in NoSQL DBMS, for effective data distribution and load processing.

In order to speed up database response times, Murat Tsyurek et al. introduced a temporary denormalization method that temporarily converts date data into numbers during the recording process. Researchers found that this method outperformed normalization methods in Microsoft SQL Server and Oracle by about eight times. Several previous works have studied denormalization approaches to optimize database performance, such as Tasyurek's temporary denormalization method (Taşyürek, 2022). However, this method has limitations in handling the increasing data volume, especially in real-time transaction scenarios. This study introduces a summary table design that can manage large volumes while maintaining efficiency and scalability.

Based on the background analysis, what sets this research apart from previous studies is the focus of the approach (Luo et al., 2019; Shah et al., 2018)(Shah et al., 2018)(A. Lee et al., 2009). While earlier research primarily emphasized database access mechanisms, this study centers on the physical design of the database. By concentrating on physical design, the proposed method not only enhances data management efficiency but also strengthens the foundational structure of the database itself. This approach provides a complementary solution to previous methods, where combining efficient access mechanisms with optimal physical design results in a more significant improvement in database application performance.

This paper proposes a partitioned table design and a summary table structure approach, allowing queries to avoid accessing unnecessary records and access only the summary tables when needed. This method significantly increases query processing speed.

RESEARCH METHODS

The proposed methodology optimizes performance by utilizing both a summary table and a partition table. This method involves identifying master tables and transaction tables (Mulyawan et al., 2020; Santosa et al., 2020) in a database design that already complies with the third normal form (Chandrashekariah & H. A., 2024; Ma & Wang, 2013; Mendoza & Piedra, 2020; Sug, 2020). We added a "period" column to the master table, adjusting it based on the needs and data volume within a given period. Figure 2 illustrates the structure of this table.

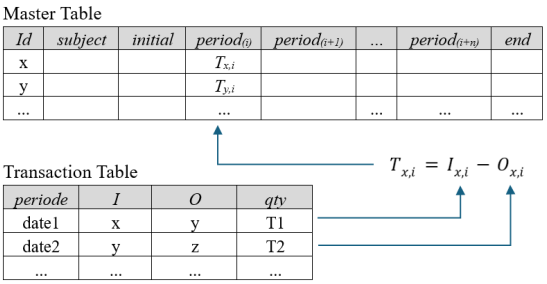

Figure 2. Summary table design

We use formula 1 to determine the value in row x of column i for each period.

$$T_{x,i} = I_{x,i} - O_{x,i} \quad (1)$$

T represents the total transactions in the transactional table per period; symbol I is the total incoming transactions; and symbol O is the total outgoing transactions, while x, y, and z represent unique rows in the master table.

The "period" in operational support system (OSS) software refers to the time cycle for processing, managing, and analyzing transaction data. This period serves as a time boundary for calculating and grouping data based on the volume of transactions occurring in the system (Carroll, 2017). The larger the transaction volume, the shorter the period duration needed to ensure that the system remains efficient and responsive to the high transaction load. Therefore, the cycle period becomes a key parameter in determining when and how data is processed, depending on the transaction volume managed by the OSS software.

In this method, the summary calculation concludes at the end of one year. At the end of the annual period (n), Formula 2 calculates the total summary for the master table for the year, thereby transforming it into the master table for period n-1. We will then create a new master table for period n, which is the active system period, and reset the total transactions per period to 0. The "initial" column will contain the value from the "end" column of period n-1.

$$end_x = initial_x + \sum_{i=1}^{12} T_{x,i} \quad (2)$$

Variable "end" represents the accumulated year-end value, "x" refers to the unique row in the master table, "initial" is the beginning-of-year value, "i" is the total amount for each period, and "T" is the value of the period for row x. Figure 3 explains the process of calculating the final value.

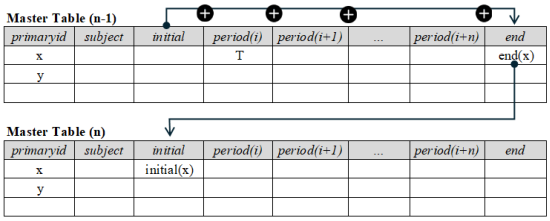

Figure 3. Transition the master table from period n to period n-1

In this approach, each monthly column in the account table provides an accurate representation of the total transactions that occurred that month. Users can easily track monthly transactions with this method, which also systematically stores transaction data according to the correct time period. Separating transactions by month also simplifies periodic financial performance evaluations and enables monitoring of transaction dynamics throughout the year.

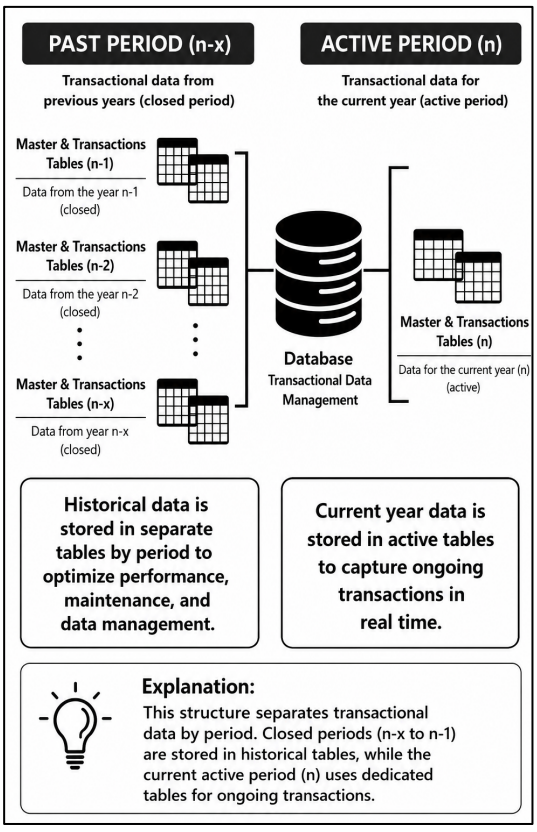

Figure 4. Database partition schema

Figure 4 illustrates the schema for separating the transaction table and the master table. Tables from past periods remain static and do not undergo any changes. This explanation elucidates the organization and maintenance of monthly data to facilitate efficient reporting and analysis.

This approach offers several notable advantages for database management. First, by limiting the number of records in the active period table, the system can avoid becoming overwhelmed by excessive data, ensuring that performance remains optimal. This prevents the database from becoming too large to handle efficiently, which is crucial as transaction volumes grow over time. Additionally, by partitioning data into distinct tables, either by year or month, based on the transaction volume, the process of data backup and restoration becomes more streamlined. This partitioning not only enhances data organization but also reduces the complexity involved in managing large datasets, making it easier to retrieve or archive information as needed.

While summary tables accelerate queries by reducing the amount of data accessed, this approach introduces potential risks to data consistency if real-time transactions are not integrated into the summary table update mechanism. Therefore, our proposed system includes data maintenance procedures to ensure that integrity remains intact.

Furthermore, this method significantly improves system efficiency by ensuring that only the tables from the active period are involved in daily operations. The system operates smoothly, free from the burden of older, inactive data, thanks to this targeted focus. As the transaction volumes increase, this system design allows for better scalability and ensures that the system remains responsive even with heavy loads. By maintaining the balance between data organization, system performance, and scalability, this approach ensures that both data management and processing remain efficient and reliable over time.

RESULT AND DISCUSSION

This section presents the experimental results, including details of the experimental setup, the benchmark data sets used, and their effectiveness. It also provides descriptions of the datasets used and offers a thorough comparative analysis between existing models and the proposed model.

A. Experimental Setup

For testing purposes, we built a system based on Google Cloud with the following specifications: 8GB RAM, dual-core vCPUs, and 10GB SSD storage. We installed Cloud SQL on the cloud server. The data we tested consisted of accounting data, which included several tables representing the functions of master tables and transactional tables. The master table, named "Account," contained account code records within the accounting system, while the transactional table, named "Journal," contained transaction records for the account codes in the master table.

Table 1. The Numbers of Records

Year	Quantity of Records
2021	1,156
2022	10,131
2023	11,559
2024	7,029

Table 2.Table Account Row List Sample

Account Code	Subject
100000	Assets
110000	Current Assets
111000	Cash and bank
111100	Cash
111101	Petty Cash
111102	Cash Vault
…	…

Table 3.Table Journal Row List Sample

Date	Debit	Credit	Amount
05/08/2021	111101	112100	500,000
05/08/2021	111101	311010	135,000
02/09/2021	111101	112100	500,000
02/09/2021	111101	311010	135,000
03/09/2021	420100	111101	572,500
03/09/2021	420100	111101	572,500
10/09/2021	317010	111101	200,000
01/10/2021	411200	111101	183,400
06/10/2021	111101	112100	500,000
…	…	…	…

Table 1 shows the composition of the "Account" with 110 records and the "Journal" with 30,275 rows. The MySQL server on the Google Cloud system received all data from the "Account" and "Journal" tables. Tables 2 and 3 explain the structure of the "Account" and "Journal".

B. Data Benchmarking

We conducted a series of tests over the internet, using a client computer with the following specifications: Intel 11th Gen Intel® CoreTM i5-1135G7 2.42 GHz processor, 20 GB RAM, 500 GB SSD. We connected the client to the Google Cloud server using a Telkom Indibiz 50 Mbps internet connection and MySQL® WorkBench 8.0.30.

Using formula 3, we tested the table structure to determine the account balance as of a specific date based on transactions recorded in the table "Journal."

$$S(t) = S(0) + \sum_{i=1}^{t} I(i) - \sum_{i=1}^{t} O(i) \qquad (3)$$

S(t) is the balance at time t, S(0) is the initial balance, I(i) is the amount of transaction inflow at period i, and O(i) is the amount of transaction outflow.

Table 4 test results show that processing time increases with the number of records. Figure 5, the time increase graph visually represents this trend.

Table 4.Time Process to Calculate the Balance

Record Numbers	Time	Note
1,556	0.031 sec	December 31, 2021.
11,687	0.125 sec	December 31, 2022.
23,246	0.266 sec	December 31, 2023
30,275	0.406 sec	August 30, 2024.

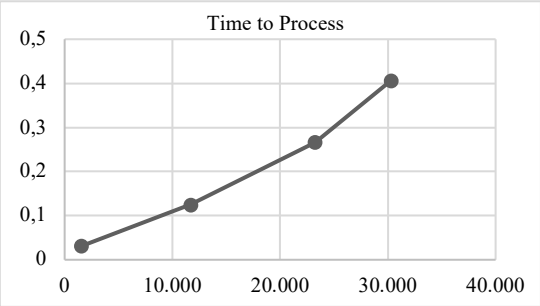

Figure 5. Incremental process time

C. Effectiveness of Summary Tables

The structure of the "akun" table using the summary method is shown in Table 5. Each unique row in the "akun" table has been modified to include additional columns for the opening balance, as well as for each month from January to December, and a final closing balance column. The opening balance column, named "Opening Balance," represents the balance at the start of the period, while the January column summarizes the transactions recorded in the "Journal" table during the month of January. The same applies to the columns from February through December.

Figure 6 shows the structure of the master table using the summary method. We have modified each unique row in the master table to include additional columns for the opening balance, each month from January to December, and a final closing balance column. The opening balance column, named "Opening Balance," represents the balance at the start of the period, while the January column summarizes the transactions recorded in the transactional table during the month of January. The same applies to the columns from February through December.

Formula 1 calculates the values for each monthly column from January to December. The closing balance column, named "akhir," is only filled when the system period changes to a new year. Formula 2 computes the closing balance at that point, providing a summary of the transactions at the end of the year.

TABLE: ACCOUNTS (GENERAL LEDGER)

Account Code	Account Name	Opening Balance (Before January)	January	February	March	April	May	...	December	Ending Balance (After December)
...	...		...	...	...	...	...	...	...	
111101	Petty Cash (Cash on Hand)	0	752,000						0	0
...	...	...	...	...	...	...	...	...	...	...

TABLE: JOURNAL ENTRIES

Date	Account Debited (Debit)	Account Credited (Credit)	Description	Amount
01/10/2022	111101 Petty Cash	112100 Cash in Bank	Cash in	600,000
01/10/2022	111101 Petty Cash	311010 Office Supplies Expense	Cash out	120,000
01/10/2022	111101 Petty Cash	112100 Transportation Expense	Cash out	466,700
01/10/2022	111101 Petty Cash	311010 Office Supplies Expense	Cash out	158,300
01/10/2022	112100 Cash in Bank	111101 Petty Cash	Cash in (replenish petty cash)	375,000
01/10/2022	111101 Petty Cash	311010 Office Supplies Expense	Cash out	102,000
01/11/2022	112100 Cash in Bank	111101 Petty Cash	Cash in (replenish petty cash)	400,000
01/12/2022	111101 Petty Cash	311010 Office Supplies Expense	Cash out	80,000
Total Transactions in January				**2,302,000**

TRANSACTION SUMMARY (JANUARY)

Type	Amount (IDR)	Description
Cash In	1,527,000	Total cash inflows into petty cash
Cash Out	775,000	Total cash outflows from petty cash
Net Change (Cash In - Cash Out)		**752,000**

Explanation:
The Petty Cash account (111101) has an increase of IDR 752,000 in January.
This amount is the net change from cash inflows (IDR 1,527,000) minus cash outflows (IDR 775,000) as recorded in the journal entries.

Figure 6. How to fill the resume value on the master table

Table 5. Table Account (General Ledger) Structure and Row Data Sample

Account Code	Subject	Opening Balance	January	February	March	…	November	December	Ending Balance
100000	Assets	0	0	0	0	...	0	0	0
110000	Current Assets	0	0	0	0	...	0	0	0
111000	Cash and bank	0	0	0	0	...	0	0	0
111100	Cash	0	0	0	0	...	0	0	0
111101	Petty Cash	0	0	0	0	...	0	0	0
111102	Cash Vault	0	0	0	0	...	0	0	0
111200	Bank	0	0	0	0	...	0	0	0
111201	Bank Mandiri	0	0	0	0	...	0	0	0
112000	Accounts Receivable	0	0	0	0	...	0	0	0

Note: The values shown in this table are placeholder values intended to illustrate the table structure and field organization only. Due to layout limitations, displaying the complete dataset with actual values is not feasible. The full original data is available as supplementary material upon request.

The process of filling the summary table, which in our experiment is represented by the table master named "Account", is illustrated in Figure 6. In this process, the period columns, named after months in this example, store the total incoming transactions minus the total outgoing transactions from the transactional table named "Journal". Each month reflects the net balance of transactions for that period, ensuring that the summary table accurately captures the financial movements for each specific month. This method enables simple tracking and analysis of monthly transactional data within the system.

During the year-end process, as demonstrated in this experiment, if the active period is the year 2024, all data from the master table for previous years will be stored in separate tables "master" named "transaction2023", "master2023", and so on. This partitioning method systematically archives past transaction records and separates the active year's data for efficient processing. Figure 7 illustrates the partitioning approach, which simplifies data management and retrieval while preserving system performance and organization as new years commence.

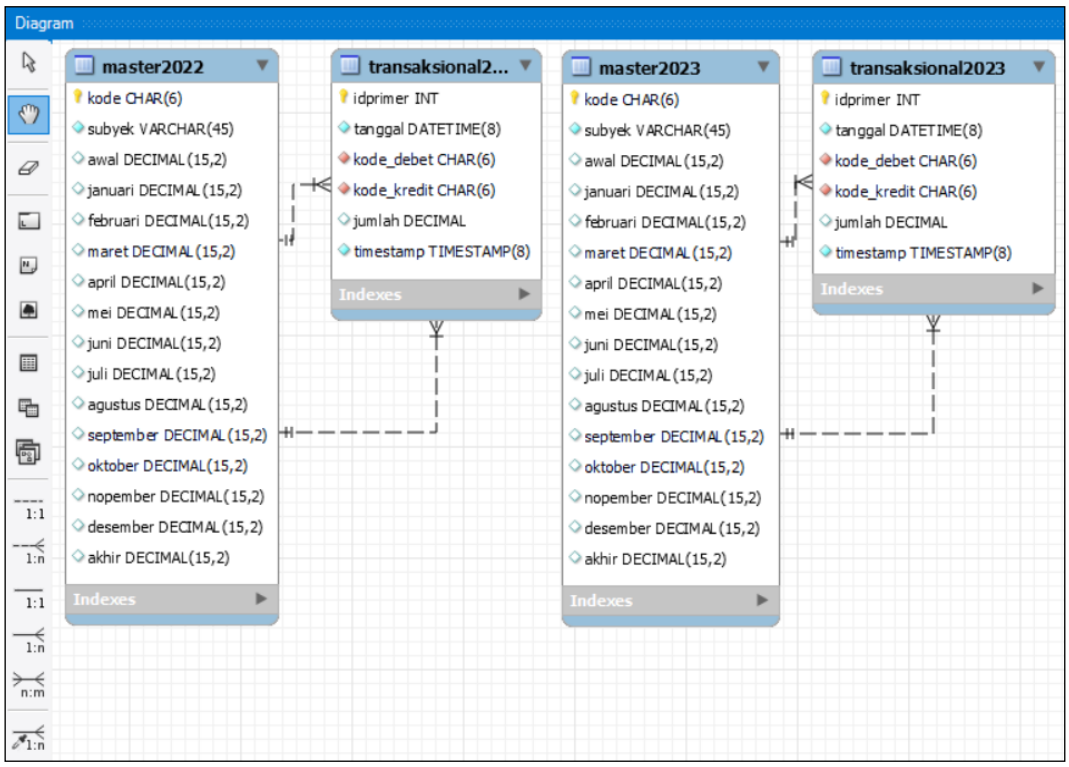

Figure 7. Database partition model design

This method is more suitable for applications with batch transaction patterns rather than scenarios with pure OLTP workloads. We could conduct further testing on platforms like PostgreSQL or MongoDB to evaluate their effectiveness in systems supporting multi-model features.

To test the performance of this summary table structure, we created a procedure in MySQL named 'loadsaldoakun' to calculate the balance of a specific account on a given date. Table 6 presents the results of the tests we ran to determine the balance as of December 31 for each year.

Table 6. Summary Table Result Test

Number of Records	Time to Process	Note
1,556	0.016	December 31, 2021.
11,687	0.031	December 31, 2022
23,246	0.047	December 31, 2023.
30,275	0.031	August 30, 2024

Table 7 presents a comparison between the previous method and the proposed method, showing a significant improvement in process efficiency.

Table 7. Comparison of Calculation Methods

Number of Records	Before	After	Efficiency (%)
1,556	0.031	0.016	48.39
11,687	0.125	0.031	75.20
23,246	0.266	0.047	82.33
30,275	0.406	0.031	92.36

Based on the test results, the summary table design demonstrated a significant performance improvement compared to a standard table. As the number of records increased, the processing time before using the summary table tended to be longer, while after implementing the summary table, the processing time drastically decreased.

For instance, we reduced the processing time from 0.031 seconds to 0.016 seconds with 1,556 records, resulting in a 48.39% efficiency gain. Efficiency increased even more as the number of records grew. With 11,687 records, the processing time dropped from 0.125 seconds to 0.031 seconds, achieving an efficiency of 75.20%. Similarly, for 23,246 records, the efficiency surged to 82.33%, and with 30,275 records, the use of the summary table resulted in a remarkable efficiency increase of 92.36%, with the processing time reduced from 0.406 seconds to just 0.031 seconds.

To test the performance of this summary table structure, we created a procedure in MySQL named 'loadsaldoakun' to calculate the balance of a specific account on a given date. The algorithm procedure is provided below.

```
Algorithm 1:
  Calculation of Final Account Balance

Input:
  cakun: String (Account Code)
  dtglakhir: Date (Target End Date)
  Output: saldo: Double (Calculated Balance)

Initialize:
 dtglawal            ← first day of month
 dtglakhir, th1, bl1 ← extract year and month from
                       dtglakhir

Account Code Normalization:
  jmlnol    ← count trailing zeros in cakun
  akuncari ← substring(cakun, 1, length - jmlnol)

Transactional Data Retrieval:
if th1 < active_system_year then
   jurnaltmp ← query transactions from journal
               where account matches akuncari
else
   flj ← "jkas" + th1
   jurnaltmp ← query transactions from archive
               table flj
end if

Classification and Calculation:
aruskas← map journaltmp to (debet, kredit) based on
         akuncari
if  rumusan(cakun) = 1 then
  saldojurnal← sum(debet) - sum(kredit)
else
  saldojurnal← sum(kredit) - sum(debet)
end if

Opening Balance Calculation:
saldoawal ← fetch balance from account code or
            archive for month < bl1
if saldoawal is null then
  saldoawal ← 0
end if

return saldojurnal + saldoawal
```

The proposed system implements a robust algorithm to calculate the final balance of a specific account, designated as loadsaldoakun. This process integrates historical data archives with real-time transactional records to ensure data integrity and system performance. The algorithm 1 follows a multi-stage execution flow as described below:

1. Pre-processing and Account Normalization

The algorithm begins by initializing temporal parameters based on the input date, dtglakhir. A critical step in this phase is the normalization of the account code (*cakun*). Since account codes may contain varying numbers of trailing zeros, the system dynamically identifies the significant digits by calculating the length excluding trailing zeros. This ensures that the subsequent data retrieval accurately captures all sub-accounts or related entries under the same parent hierarchy.

2. Adaptive Data Sourcing Strategy

To optimize query performance and maintain scalability, the system employs an adaptive data sourcing strategy. It distinguishes between the current active period and historical periods. Active period is if the target year corresponds to the system's current active period, data is fetched directly from the primary journal table. The archive period is for historical data; the system dynamically constructs references to archived tables (e.g., jkas[year]). This separation prevents the primary transactional table from becoming a bottleneck as the dataset grows over time.

3. Transactional Mutation Analysis

Once the relevant records are retrieved into a temporary structure (*journaltmp*), the system classifies transactions into debits and credits. The net mutation (saldojurnal) M_{jurnal}, is calculated based on the accounting nature of the account (*rumusan*). For normal debit accounts, the mutation is defined as:

$$M_{jurnal} = \begin{cases} \sum D - \sum K \,, if\ normal\ debit \\ \sum K - \sum D \,, if\ normal\ kredit \end{cases} \quad (4)$$

4. Integration of Opening Balances

The final stage involves the aggregation of the opening balance with the current period's mutation. The opening balance S_{init} (*initialbalance*) is calculated by accumulating the initial balance and the balances of all months preceding the target month within the fiscal year. This is expressed as:

$$S_{init} = A_{base} + \sum_{i=1}^{m-1} B_i \quad (5)$$

Where A_{base} is the initial balance retrieved from the account table or its respective archive? B_i It represents the balance of the i month. m is the month extracted from *dtglakhir*. Finally, the total balance S_{total} representing the accurate financial standing at the specified point in time is defined as:

$$S_{total} = S_{init} + M_{jurnal} \quad (6)$$

CONCLUSION

Based on the test results, the summary table design demonstrated a significant performance improvement compared to a standard table. As the number of records increased, the processing time before using the summary table tended to be longer, while after implementing the summary table, the processing time drastically decreased.

The summary table design proposed in this study bridges the gap between traditional normalized table structures and the need for high-speed data retrieval in modern database applications. By integrating a partitioned table schema with aggregated summaries, our approach reduces the computational overhead of querying large datasets. This method demonstrates clear advantages over existing solutions, such as denormalization and horizontal partitioning, by offering a balanced trade-off between query speed and system complexity. Overall, the summary table design has proven to be highly effective in reducing data processing time as the number of records increases, leading to much more efficient system performance.

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to the Directorate General of Higher Education, Ministry of Education and Culture of the Republic of Indonesia, for funding this research in 2024. Our thanks also go to Universitas Semarang and Universitas An Nuur for their invaluable support throughout this study. We deeply appreciate our partners who contributed significantly to the successful completion of this research.

.

REFERENCES

Acharya, S., Ghadge, K., Ranjan, B. S. C., Devadula, S., & Chakrabarti, A. (2020). Evaluating the effectiveness of InDeaTe tool in supporting design for sustainability. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *34*(1), 45–54. https://doi.org/10.1017/S0890060419000337

Albert, A., & Bozza, C. (2016). The Relational Database System of KM3NeT. In *Epj Web of Conferences*. https://doi.org/10.1051/epjconf/201611607004

Azhir, E., Jafari Navimipour, N., Hosseinzadeh, M., Sharifi, A., & Darwesh, A. (2021). A technique for parallel query optimization using MapReduce framework and a semantic-based clustering method. *PeerJ Computer Science*, *7*, e580. https://doi.org/10.7717/peerj-cs.580

Bt Wan Abu Bakar, W. A., Abdullah, Z. B., Md Saman, Md. Y. B., Abd Jalil, M. M. B., Man, M. B., & Herawan, T. (2015). Vertical Association Rule Mining: Case study implementation with relational DBMS. 2015 International Symposium on Technology Management and Emerging Technologies (ISTMET), 279–284. https://doi.org/10.1109/ISTMET.2015.7359044

Carroll, N. (2017). The Routledge Companion to Accounting Information Systems (M. Quinn & E. Strauss, Eds.). Routledge. https://doi.org/10.4324/9781315647210

Castro-Medina, F., Rodríguez-Mazahua, L., López-Chau, A., Cervantes, J., Alor-Hernández, G., Machorro-Cano, I., & Arrioja-Rodríguez, M. L. (2022). A New Method of Dynamic Horizontal Fragmentation for Multimedia Databases Contemplating Content-Based Queries. Electronics, 11(2), 288. https://doi.org/10.3390/electronics11020288

Chandrashekariah, Y. A. B., & H. A., D. (2024). Structured query language query join optimization by using rademacher averages and mapreduce algorithms. *Bulletin of Electrical Engineering and Informatics*, *13*(3), 1730–1740. https://doi.org/10.11591/eei.v13i3.6837

Kim, K., Kim, S.-M., Park, Y., Lee, E., Jung, S., Kang, J., An, D., Min, K., Shim, S. R., Yu, H. W., & Han, H. W. (2024). A blockchain-based healthcare data marketplace: prototype and demonstration. *JAMIA Open*, *7*(2). https://doi.org/10.1093/jamiaopen/ooae029

Kumar, K., & Azad, S. K. (2017). Database normalization design pattern. *2017 4th IEEE Uttar Pradesh Section International Conference on* Electrical*, Computer and Electronics (UPCON)*, 318–322. https://doi.org/10.1109/UPCON.2017.8251067

Lee, A., Zait, M., Cruanes, T., Ahmed, R., & Zhu, Y. (2009). Validating the Oracle SQL engine. Proceedings of the Second International Workshop on Testing Database Systems, 1–7. https://doi.org/10.1145/1594156.1594161

Lee, Y. T. (1999). *An overview of information modeling for manufacturing systems integration*. https://doi.org/10.6028/NIST.IR.6382

Liu, Z., & Idreos, S. (2016). Main Memory Adaptive Denormalization. *Proceedings of the 2016 International* Conference on Management of Data, 2253–2254. https://doi.org/10.1145/2882903.2914835

Loand, C.-M., & Hung, H.-Y. (2014). Towards a UML Profile to Relational Database Modeling. In Applied Mathematics & Information Sciences. https://doi.org/10.12785/amis/080233

Luo, X., Huo, X., & Fu, L. (2019). Query Optimization of Data Based on Window Function and Distributed Cluster in Visual Academic Search System. Shanghai Jiaotong Daxue Xuebao/Journal of Shanghai Jiaotong University, 53(8), 978–982.

Ma, C. X., & Wang, Y. M. (2013). Application of the Standardization of the Relationship in the Database Design. Applied Mechanics and Materials, 401–403, 1809–1812. https://doi.org/10.4028/www.scientific.net/AMM.401-403.1809

Mendoza, D., & Piedra, N. (2020). TutNorBD: Assistant for teaching and learning process of relational database normalization up to 3NF from a universal table. 2020 XV Conferencia Latinoamericana de Tecnologias de Aprendizaje (LACLO), 1–8. https://doi.org/10.1109/LACLO50806.2020.9381184

Mulyawan, B., Vionelsy, & Sutrisno, T. (2020). Product recommendation system on building materials shopping using FP-Growth algorithm. IOP Conference Series:

Materials Science and Engineering, 1007, 012144. https://doi.org/10.1088/1757-899X/1007/1/012144

Rodrigues, R. A., Filho, L. A. L., Gonçalves, G. S., Mialaret, L. F. S., da Cunha, A. M., & Dias, L. A. V. (2018). Integrating NoSQL, Relational Database, and the Hadoop Ecosystem in an Interdisciplinary Project involving Big Data and Credit Card Transactions (pp. 443–451). https://doi.org/10.1007/978-3-319-54978-1_57

Samidi, S., Suladi, R. Y., & Lesmana, A. B. (2022). Implementation of Database Distributed Sharding Horizontal Partition in MySQL. Case Study of Application of Food Serving On Kemkes. JURNAL SISFOTEK GLOBAL, 12(1), 50. https://doi.org/10.38101/sisfotek.v12i1.477

Santosa, A., Rinaldy, A. M., & Andriani, D. (2020). Database Design for Distribution Simulation Game. *IOP Conference Series: Materials Science and Engineering*, *879*(1), 012179. https://doi.org/10.1088/1757-899X/879/1/012179

Shah, B., Pareek, J., & Kanziya, D. (2018). A Novel Approach to Optimize Subqueries for Open Source Databases (pp. 331–346). https://doi.org/10.1007/978-981-10-6916-1_31

Sug, H. (2020). A Method for Normalization of Relation Schema Based on Data to Abide by the Third Normal Form. WSEAS TRANSACTIONS ON MATHEMATICS, 19, 216–225. https://doi.org/10.37394/23206.2020.19.20

Suryana, A., Supendi, Y., & Yulianto, E. (2019). Designing software management work unit of research and development department of public works and public housing. *International Journal of Advanced Science and Technology*, *28*(6), 55–61.

TAŞYÜREK, M. (2022). A Novel Approach to Improve the Performance of the Database Storing Big Data with Time Information. *Balkan Journal of Electrical and Computer Engineering*, *10*(4), 388–396. https://doi.org/10.17694/bajece.1059070

Wang, F. Q., Liu, Y., & Han, Q. L. (2014). Research on Improving Data Query Speed Methods. *Applied Mechanics and Materials*, *556–562*, 5825–5828. https://doi.org/10.4028/www.scientific.net/AMM.556-562.5825