

Maintaining Query Performance through Table Rebuilding & Archiving

Widyastuti Andriyani*¹, Pujiyanto ²

^{1,2} Department of Information Technology, Indonesian Digital Technology University,
Yogyakarta, Indonesia

e-mail: *¹widya@utdi.ac.id, ²student.pujiyanto@mti.utdi.ac.id

Abstrak

Kemajuan pesat dalam akumulasi data transaksional dan frekuensi pembaruan telah menimbulkan tantangan dalam mempertahankan kecepatan query pada database relasional, meskipun konfigurasi sistem telah dioptimalkan. Meskipun telah ada peningkatan pada konfigurasi dan pengaturan query sebelumnya, peningkatan volumetrik pada tabel transaksi di database, yang ditandai dengan pertumbuhan data dan pembaruan konstan di setiap entri, telah mempengaruhi kecepatan query. Beberapa mesin basis data saat ini kadang-kadang melewati isu-isu seperti fragmentasi blok yang berdampak pada performa basis data. Mengingat data kini merupakan aset utama bagi bisnis, menjaga kinerja query yang optimal menjadi esensial dalam mendukung kegiatan dan pengambilan keputusan bisnis. Oleh karena itu, bagi administrator basis data, memilih metode yang tepat untuk memastikan performa dan integritas data menjadi hal yang krusial.

Kata kunci— Sistem Manajemen Basis Data Relasional (RDBMS), Fragmentasi basis data, Rekonstruksi Tabel, Pengarsipan data

Abstract

Despite optimized system configurations, rapid advances in transactional data accumulation and update frequency have created challenges in maintaining query speed on relational databases. While there have been improvements in configuration and query settings in the past, the volumetric increase in transaction tables in databases, characterized by data growth and constant updates on each entry, has impacted query speed. Some current database engines sometimes miss issues such as block fragmentation that impact database performance. Since data is now businesses' main asset, maintaining optimal query performance is essential in supporting business activities and decision-making. Therefore, for database administrators, choosing the right method to ensure data performance and integrity is crucial.

Keywords— Relational Database Management System (RDBMS), Database fragmentation, Table Reconstruction, Data archiving

1. INTRODUCTION

Relational Database Management System (RDBMS) has become one of the basic needs for companies. Nowadays data is a valuable asset and information for the customers, and having access to it is absolutely necessary. Obviously, a reliable RDBMS is expected to support organizations by providing information accurately and timely [1].

Database, before being used in production systems, has been properly configured, optimizing business process tuning, including indexing tables and performance testing [2].

Indexing is a technique to increase the performance query of table [3]. Bitmap index is a technique that is common to make performance databases better [4].

In a paper titled Archiving ERP data to enhance operational effectiveness: the case of Dolphin explains that Transaction-intensive, customer-facing applications, and the most critical high-volume ERP and CRM applications are collecting and storing huge amounts of data, resulting in an exponential increase in the size of the database, and highlight why data archiving is crucial for the enterprises running SAP solution in optimizing business performance[5].

All about Oracle database fragmentation, written by Craig A. Shallahamer, OraPub, Inc. There are many types of Oracle database fragmentation. Some are harmful, and some are not. He explains the Database Administrator (DBA) to proactive application design and space management steps, addresses tablespace space, segment, data block, index leaf block, and row fragmentation [6].

In a B-tree (row store) index, fragmentation exists when the index has pages where the logical ordering in the index, based on the index key values, does not match the physical order of the index pages. The Database Engine automatically modifies the index whenever an insert, update, or delete operation is performed on the underlying data. For example, adding a row in a table can cause existing pages in the row store index to split, making room for the insertion of new rows. Over time, these modifications can cause the data in the index to be spread across the database (fragmented). A query that reads many pages uses a full index or range scan; a highly fragmented index can reduce query performance because additional I/O may be required to read the data required by the query. Instead of a small number of large I/O requests, a query would require many small I/O requests to read the same amount of data [7].

Hariprasath Rajaram explains that performance is sometimes degraded in the Online Transaction Processing (OLTP) database environment because of table plan changes and row-chaining and row-migration issues. Maintenance activities such as table reorganization may be required based on transactions on the table. After table reorg, performance was seen to improve a lot [8].

In the current study conducted at PT XYZ, both rebuild table and data archiving methods are synergistically combined to optimize query performance. While earlier research might have individually explored these techniques, the integration of the two within this unique context stands out. Specifically, the study harnesses a high transaction table updating with staggering data of 45 million rows every month, a dataset scale potentially uncharted in prior investigations. Notably, observations pointed out challenges like big datasets leading to overheating in the Database Management System (DBMS), a revelation that could be a novel finding not addressed in earlier works. This study also draws references from several noteworthy papers, such as those focusing on Archiving ERP data and Oracle database fragmentation. A juxtaposition of these findings with the current research could elucidate distinct facets. Central to this research's distinctiveness is its contextual grounding, which is anchored in the tangible operations of PT XYZ. Such a hands-on application in a real-world environment, emphasizing these specific methodologies, might have been overlooked in previous scholarly endeavors.

2. METHODS

2.1. Infrastructure preparation

Besides setting up server hardware and the pre-configured Oracle version 11 database, the database is categorized into two types based on transaction nature: Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) [9]. The transaction schema is designed to house transaction tables, and the history schema is dedicated to data archiving. Archived data refers to information that remains unchanged yet is essential for reporting purposes [10].

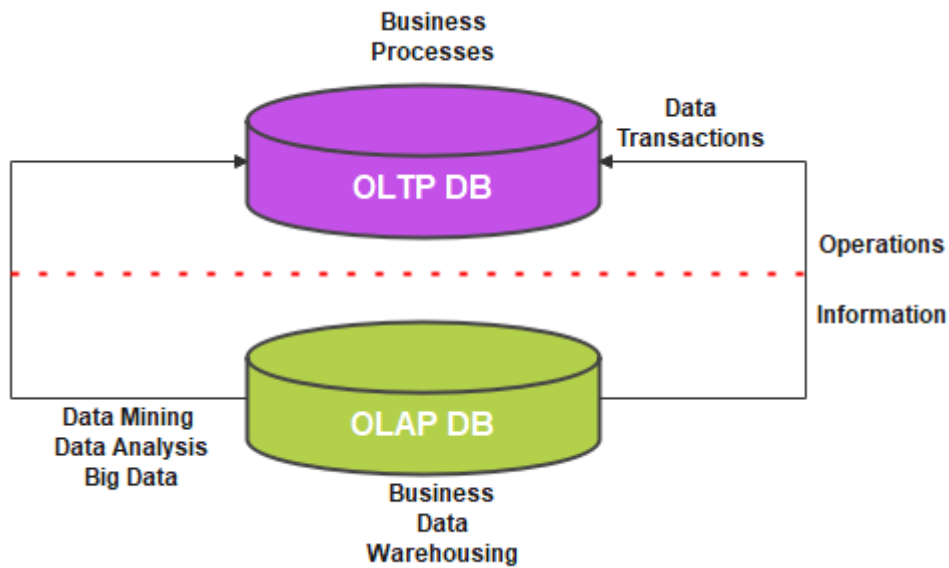


Figure 1 Database OLTP OLAP <https://www.geeksforgeeks.org/>

2.2. Data Tablespace

Tablespace serves as a logical storage unit within a database. It may be composed of one or multiple physical data files. Within the Transaction schema, both the transaction table and its corresponding index have distinct tablespaces and datafiles. Similarly, in the history schema, the index and data are allocated to separate tablespaces.

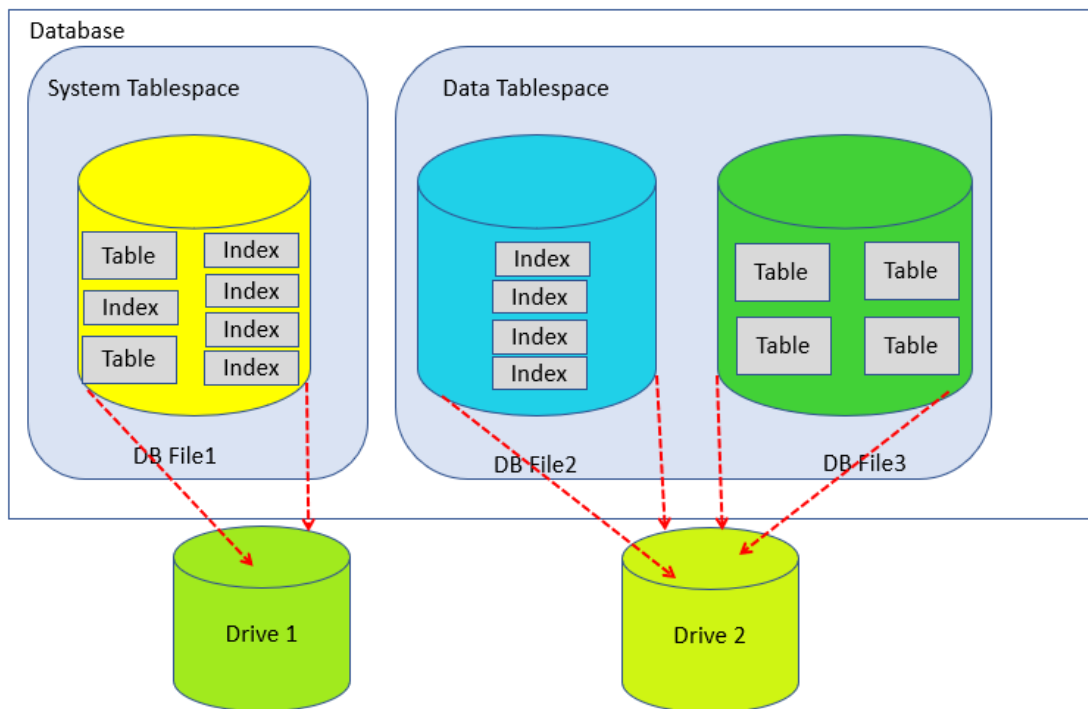


Figure 2 Logic Table Schema in Tablespace

Within the transaction schema, a data table is designed with 47 columns and nine indexes, encompassing primary key indexes, combined indexes, and individual column indexes to expedite query times according to the specific columns users search for.

3. RESULTS AND DISCUSSION

Before initiating the table rebuild and data archiving process, ensure that the table designated to hold the data being transferred to the history schema (for archiving) is set up according to the subsequent design.

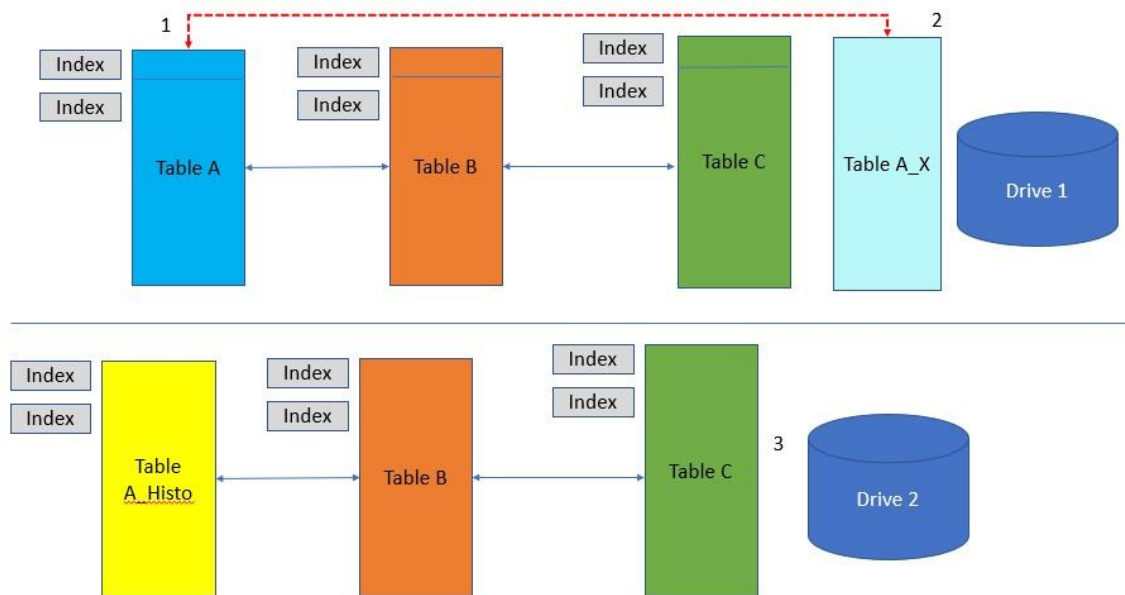


Figure 3 OLTP and OLAP table schemas

Before proceeding with the table rebuild, transactions must be halted to ensure data remains consistent. The rebuilding typically occurs during designated maintenance periods or when transactions are paused. A crucial preliminary step involves drafting a DDL script detailing the creation of tables, indexes, and constraints. During this phase, meticulous attention to detail is required when setting up the script for object regeneration. With Oracle's Toad editor, a table reconstruction script is available, designed to minimize potential mishaps. It's advised to keep a copy of this rebuild script as it will be instrumental when remodeling database components, specifically tables, indexes, and constraints for table A. Following these steps, the transaction table named 'A' should be rebranded as 'A_X.'

Before initiating the table rebuild, halt all transactions to ensure the consistency of the data. The rebuilding process typically occurs during scheduled maintenance windows or transaction break points. Before starting the rebuild, it's essential to draft a DDL script outlining the creation of tables, indexes, and constraints. Precision and caution are required when devising the script for object reestablishment. A specialized table reconstruction script has been devised using Oracle's Toad editor to minimize potential mistakes. Ensure this rebuild script is stored securely as it will

be crucial for reshaping database elements, particularly tables, indexes, and constraints in table A. Subsequently, it's advisable to re-label the transaction table from 'A' to 'A_X.'

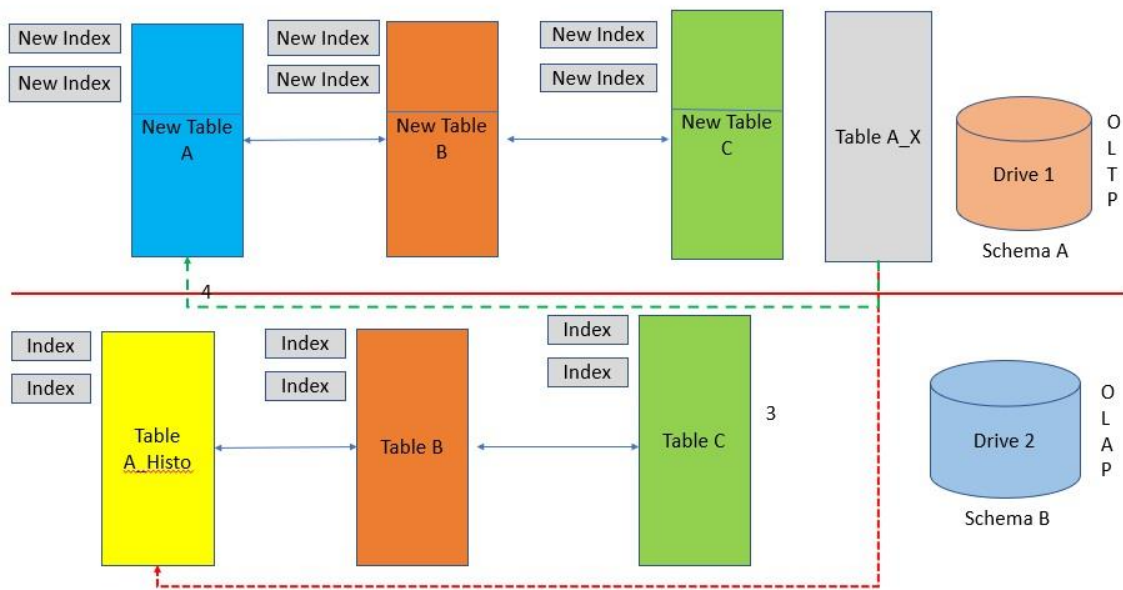


Figure 4 *Rebuild and Archiving*

3.1. Pre-Implementation

After setting up the infrastructure and software, the prerequisites for pre-implementation include:

- Establishment of both transaction and transaction index tablespaces for the logical placement of transaction data.
- Formulation of this tablespace and history index tablespace for relocating updated data. It's imperative that the history tablespace and history index are situated on separate physical drives.
- Instituting a transaction user granted rights to store data in the transaction and transaction index tablespaces, as well as a history user bestowed with access to the Histo tablespace and Histo Index.
- During the actual implementation phase, it's crucial to pinpoint tables with high transaction volumes, encompassing both data insertions and updates alongside those accessed frequently. Specific measures are vital at this juncture to ascertain the smooth execution of the rebuild & archiving operations while minimizing potential hazards.

3.2. Implementation

During the table rebuilding and data archiving processes, several phases are undertaken as described below:

- Pinpoint tables with significant transaction volumes and frequently executed queries.
- Within the database schema, numerous database entities exist, encompassing tables, views, constraints, and stored procedures. Tables with minimal data and transactions, like user information and primary data, aren't the focal points of this research as their impact on the observation procedure is minor. Conversely, thorough preparation is required for tables with heavy transactions, which includes understanding their interrelations and associated entities, such as indexes and constraints, which will be essential during the rebuild

- c. Halt all transactions by ensuring no ongoing operations and blocking new application session access.
- d. Briefly suspend access to the database by restricting user access. This measure ensures data consistency throughout the rebuilding and archiving stages and avoids potential transaction mishaps.

During the process of table rebuilding and data archiving, several steps will be taken:

- a. Pinpoint tables with high transaction volumes and frequently executed queries.
- b. Within the database schema, numerous entities exist, encompassing tables, views, constraints, and stored procedures. Tables that contain minimal data or experience low transaction rates, like user and master data, aren't the primary focus of this research due to their limited significance in the study. Conversely, it's essential to set up tables with high transaction volumes, ensuring all related entities, such as indexes and constraints, are in place for future rebuilding.
- c. Halt all transactions, making sure that no active tasks are in progress and no fresh sessions access the application.
- d. Temporarily restrict access to the database by locking out users. This precaution ensures data integrity throughout the rebuilding and archiving stages and mitigates potential transactional errors.

3.3. Transaction Table Observation

Table 1 Previous Query Observations

Case	Month	Number of rows (Million)	Time Per Rows (ms)	Time Per Month(s)	Time All Data	Time difference per (ms)	Time difference per Month(s)	Time difference All Data (s)
Ori	1	45	289	2	2	3	3	3
Updated	1	45	292	5	5			
Ori	2	90	294	6	4	72	8	1
Updated	2	90	366	14	5			
Ori	3	135	302	9	6	57	18	19
Updated	3	135	359	27	25			
Ori	4	180	306	10	10	442	6	44
Updated	4	180	748	16	54			
Ori	5	225	454	62	62	-80	13	7
Updated	5	270	374	75	69			
Ori	6	270	840	73	64	160	-12	3
Updated	6	270	1000	61	67			
Ori	7	315	100	89	96	47	2	1
Updated	7	315	1047	91	97			

Every month, data accumulates at a rate of 45 million rows, with each row undergoing updates within that same month. In this investigation, we tracked the evolution of query performance through seven months of data accumulation. Before any updates, the average data row is sized at 153 bytes, accounting for a total of 45,010,790 rows and 1,066,120 blocks. Post-update, the average size of a row swells to 273 bytes, and the total row count slightly increases to 45,027,430, taking up 1,751,015 blocks. The variation in the block allocation is attributed to data updates necessitating additional space. The preset 10% free space within a block proves inadequate, compelling the formation of a new block to house the revised data. This initiation of a new block paves the way for space fragmentation, leading to the potential situation where a single data row might span across two distinct blocks. Such circumstances contribute to a sluggish response time during data queries, especially as data volume grows and updates are frequent. To

pinpoint this, you can utilize the following query: `select a.owner, a.table_name, b.blocks, alcbkls, a.blocks usdblks, (b.blocks-a.empty_blocks-1) hgwt from dba_tables a, dba_segments b where a.table_name=b.segment_name and a.owner=b.owner and a.owner='OWNER' and a.blocks <> (b.blocks-a.empty_blocks-1) and a.table_name='TABLE_NAME'.`

3.4. Rebuild table

Reconstructing a table involves reproducing a specific table (TABLE A) within the database, especially after it has experienced numerous Data Manipulation Language (DML) activities, such as insertions and modifications. This operation mandates a pause in application activities, typically scheduled during the early morning when transactional demands are minimal. To prevent naming conflicts that could result in creation failures, it's imperative to first retitl the current table (TABLE A) to another name (TABLE A_X). This renaming doesn't impact the existing data or its indexes. However, following the renaming, any stored procedures or functions may become dysfunctional, as they won't be able to locate TABLE A.

Table 2 Rename the transaction table




TABLE A	
Month 1	45.000.000
Month 2	45.000.000
Month 3	45.000.000
Month 4	45.000.000
Month 5	45.000.000
Month 6	45.000.000
Month 7	45.000.000

TABLE A_X	
Month 1	45.000.000
Month 2	45.000.000
Month 3	45.000.000
Month 4	45.000.000
Month 5	45.000.000
Month 6	45.000.000
Month 7	45.000.000

The recreation of table A ensues, mirroring its original state, including its tablespace positioning. During this phase, only the foundational table structure is set up, excluding any indexes and constraints. The newly established TABLE A is then populated with specific data based on the predetermined transaction history duration that users will access. It's essential to highlight that the period for retaining accessible data is subject to each organization's guidelines, which might differ. For this particular phase, the volume of data designated for reintroduction into the new TABLE A spans the last five months, specifically from the third to the seventh month, amassing 225,000,000 rows derived from 45,000,000 rows multiplied by five.

Table 3 Insert Data Into New Table

NEW TABLE A	
Month 3	45.000.000
Month 4	45.000.000
Month 5	45.000.000
Month 6	45.000.000
Month 7	45.000.000




TABLE A_X	
Month 1	45.000.000
Month 2	45.000.000
Month 3	45.000.000
Month 4	45.000.000
Month 5	45.000.000
Month 6	45.000.000
Month 7	45.000.000

After the insert process is complete, enter the process of creating indexes back to New TABLE A. This needs to be done to refresh the index and serve the queries required for the application.

Table 4 Insert Data Into Table

Column Index 1
Column Index 2
Column Index 3
Column Index 4
Column Index 5
Column Index 6
Column Index 7
Column Index 8
Column Index 9

NEW TABLE A	
Month 3	45.000.000
Month 4	45.000.000
Month 5	45.000.000
Month 6	45.000.000
Month 7	45.000.000

During this phase, the transaction table, TABLE A, has been rejuvenated with fresh and index blocks. This comes post the DML transaction, with negligible alterations in data across rows. The culmination of this process involves instituting database attributes, specifically constraints, and recompiling the PL/SQL stored procedures and functions that serve as the database's business logic. This ensures the smooth operation of all business-related processes concerning TABLE A. The final move involves table analysis to retrieve database statistical insights and refine query optimization. Subsequently, the transaction table is primed to store data that customers will reimburse. This regimen is periodically executed in the database's upkeep, emphasizing transaction tables and data sustainability.

3.5. Data Archiving

At this point, it's been established that the data subject to the DML (update) has been retained in the transaction table for the previous five months and the current month. This allows customers to view data statuses for the most recent six months.

Table 5 Insert Data Into Table Histo

Column Index 1
Column Index 2
Column Index 3
Column Index 4
Column Index 5
Column Index 6
Column Index 7
Column Index 8
Column Index 9

TABLE A_HISTO	
Month 1	45.000.000
Month 2	45.000.000




TABLE A_X	
Month 1	45.000.000
Month 2	45.000.000
Month 3	45.000.000
Month 4	45.000.000
Month 5	45.000.000
Month 6	45.000.000
Month 7	45.000.000

3.6. Outcome Measurement

The following are the results of the query comparison before the rebuild was carried out.

Table 6 Comparison table of query speed before and after the update

Month	Number of Data (Million)	Query per rows before updated (ms)	Query per month before updated (s)	Query count all before update (ms)	Query per rows after updated (ms)	Query per month after updated (s)	Query count all before after (ms)
1	45	289	2	2	292	5	5
2	90	294	6	4	366	14	5
3	135	302	9	6	359	27	25
4	180	306	10	10	748	16	54
5	225	454	62	62	374	75	69
6	270	840	73	64	1000	61	67
7	315	1000	89	96	1047	91	97

The effectiveness of the rebuild is also obtained from the disk capacity used, not using the word "essentially" to mean "approximately" or "effectively."

Table 7 Comparison of block sizes before and after rebuild

TABLE	Before Rebuild		After Rebuild		Result	
	SIZE (GB)	INDEX SIZE (GB)	SIZE (GB)	INDEX SIZE (GB)	SIZE (GB)	INDEX SIZE (GB)
A	92,44	104,02	68,76	48,04		
A_HISTO			27,48	19,30		
TOTAL			96,24	67,34	-3,8	20,72

4. CONCLUSIONS

From the research conducted, it was found that updated rows can lead to block fragmentation in the transaction table by more than 4.9 times, impacting the query speed depending on the table's characteristics. The solution in rebuilding the table has proven effective by reducing table and index block fragmentation by 36 times the disk consumption. Furthermore, after adding data, the query speed increased to 4 times faster with this rebuild process. Additionally, after the sixth month, archiving data successfully maintained the query speed by limiting the amount of data accessed. In the future, there may be other methods in the process of rebuilding and archiving that are more advanced (reducing manual processes), and the impact will

be more significant if there is a DML process several times on each row, including the process of deleting rows before the rebuild is carried out.

REFERENCES

- [1] Oracle, "Overview of SQL," *oracle.com*, 2022.
- [2] S. J. Kamatkar, A. Kamble, A. Viloría, L. Hernández-Fernández, and E. García Cali, "Database performance tuning and query optimization," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10943 LNCS, pp. 3–11, 2018, doi: 10.1007/978-3-319-93803-5_1.
- [3] D. Iskandar, M. Fachrurroji, W. A. Septyo, and A. K. A., "Database Tuning in Hospital Applications Using Table Indexing and Query Optimization," vol. 6, pp. 1960–1967, 2022.
- [4] B. Ding, S. Chaudhuri, and V. Narasayya, "Bitvector-aware Query Optimization for Decision Support Queries," *Proc. ACM SIGMOD Int. Conf. Manag. Data*, pp. 2011–2026, 2020, doi: 10.1145/3318464.3389769.
- [5] S. Srivastava and R. Misra, "Archiving ERP data to enhance operational effectiveness: The case of Dolphin," *Int. J. Inf. Technol. Manag.*, vol. 16, no. 2, pp. 162–172, 2017, doi: 10.1504/IJITM.2017.083866.
- [6] C. Shallahamer, "All About Oracle Database Fragmentation," *White Pap. OraPub, Inc. Lake Oswego, USA*, 2000.
- [7] Microsoft, "Konsep: fragmentasi indeks dan kepadatan halaman," *docs.microsoft.com*, 2022.
- [8] Hariprasath Rajaram, "Reorganization Table," *oracledbwr.com*, 2022.
- [9] Y. Liu, R. Kumar, A. Tripathi, A. Sharma, and M. Rana, "The Application of Internet of Things and Oracle Database in the Research of Intelligent Data Management System," *Inform.*, vol. 46, no. 3, pp. 403–410, 2022, doi: 10.31449/inf.v46i3.4019.
- [10] P. Dix, "InfluxData (InfluxDB) | Time Series Database Monitoring & Analytics," *InfluxData, Inc.*, vol. 12, no. 2, pp. 168–174, 2017, [Online]. Available: <https://www.influxdata.com>