**Jurnal Teknik Informatika (JUTIF)**
Vol. 5, No. 4, August 2024, pp. 1009-1014

DOI: https://doi.org/10.52436/1.jutif.2024.5.4.1962
p-ISSN: 2723-3863
e-ISSN: 2723-3871

# *COMPARISON OF INDEX, PARTITION, AND MATERIALIZED VIEW METHODS ON THE ORACLE DATABASE STUDY ON CENTRAL GOVERNMENT FINANCIAL REPORTS (LKPP)*

**M. Harviandi Rachman[*1], Samidi[2], Eko Aprianto[3]**

[1,2,3]Faculty of Information Technology, Master of Computer Science Study Program, Universitas Budi Luhur, Indonesia
Email: [1]2311600874@student.budiluhur.ac.id, [2]samidi@budiluhur.ac.id, [3]2311600882@student.budiluhur.ac.id

(Article received: March 30, 2024; Revision: May 03, 2024; published: July 29, 2024)

***Abstract***

*The Indonesian Central Government Financial Report (LKPP) is a financial document prepared to increase transparency and accountability in the implementation of the State Revenue and Expenditure Budget (APBN). It is prepared within a tight schedule, hence changes made by each entity must be updated promptly. Therefore, this research focuses on the optimal table design for presenting financial reports. Query optimization is a major concern in database design, with the use of indexing concepts to increase data search speed. Table partitioning is also a strategy to consider, namely dividing a table into parts that form separate data ranges. The use of a Materialized View (MV) is another alternative, providing increased performance with the space-for-time trade-off principle. Experiments were carried out by comparing the response time of applying index, partition, and materialized views to produce financial report data. Experimental results indicate that materialized views can provide significant advantages when faced with large volumetric data. The decision to choose a materialized view can be considered contextually, depending on the specific needs and characteristics of the data encountered in a database system.*

**Keywords**: *Indexing, Materialized View (MV), Performance optimization, Query optimization, Table partitioning.*

## 1. INTRODUCTION

The Indonesian Central Government Financial Report (LKPP) is a financial document prepared by the Indonesian Central Government to increase transparency and accountability in the implementation of the State Revenue and Expenditure Budget (APBN) [1]. The large number of reporting entities creates a large growth in the volume of financial data, in line with the developing dynamics in public administration.

Large data sets can cause problems with the Database Management System (DBMS) in the performance of the database used [2]. Even with a strong database, a mature table design for presenting financial reports must be prepared carefully. The main challenge in preparing LKPP is the process of forming financial reports which depends on transactions carried out by many reporting entities, that must always reflect the current conditions. This is a challenge in itself because the time to finalize the final LKPP value is limited, so the value in the financial statements must be able to be updated following changes made by each entity.

When designing a financial report database, it is necessary to optimize the queries used. The main goal of database optimization is usually general, namely to increase a numerical value, which characterizes database performance well [3]. Current databases are typically designed as general-purpose systems and aren't customized on a case-by-case basis to suit the specific workload and data characteristics of individual users. [4]

Chopade [5] suggests that indexing is an important concept for faster data retrieval. Index tuning, as part of physical database design, is the task of selecting, creating, deleting, and rebuilding index structures to reduce workload processing time [6].

Apart from indexing, table partitioning is also something you need to pay attention to. Table partitioning is the division of a table into arbitrary parts that form several separate data ranges [7]. This division occurs within the database files, yet to the developer, they appear as a unified entity [8]. The combination of using Index and Partition will produce better time records than not using them [9].

Another solution is to use a materialized view. Materialized View (MV) represents a sophisticated redundancy optimization technique tailored for analytical workloads [10]. Materialized View is quite important in DBMS which can significantly improve query performance based on the space-for-time trade-off principle [11]. Rewriting with materialized views will improve the performance of SQL statements [12]. The optimal solution would enable operations on the database to remain as swift as they were during its initial implementation, even with a growing number of records in the tables [13].

The ongoing process of database tuning, aimed at enhancing the performance of applications interacting with a database [14], is complemented by the use of synonyms, which provide methods enabling users to transparently display and utilize other users' objects [15].

In this study, researchers tried to carry out further tests comparing the response time of implementing partition and materialized view to produce central government financial report data. As a comparison, the time needed to run a query is used to determine which database optimization method is better. The difference between this research and previous research is the object, method, combination of data, and research time.

## 2. RESEARCH METHODS

This research uses an experimental method using a sample of general ledger tables for central government financial reports for the period January - February 2021. The general ledger table was chosen because the data in the general ledger can represent the latest financial report data. The data used was 10,002,137 raw data using Oracle as the database.

In this research, the trial balance table was created using views and materialized views. The trial balance table is used to shorten queries originating from the general ledger. Testing was carried out on a computer with the Debian Linux 11 operating system which has 4 core processors, 16 GB RAM, with 100 GB SSD. The database used is Oracle 19c which is accessed using SqlDeveloperTools.

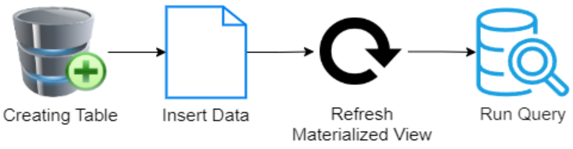


Figure 1. Research Method

The steps taken in this research were to create a table for research. Next, enter the research data little by little into the table. Next is to refresh the materialized view to update the data in the materialized view. Then run a test query to get the time needed to run the query.

### 2.1. Creating Table

In this stage, the researcher prepared the essential tables needed. The main table created is the general ledger which contains journals for validated transactions that are used to produce reports. The general ledger table is created using the indexing, partitioning method, and materialized view.

Creating tables using the index method in the Oracle database is done by creating a table and then adding indexes for the selected columns. Creating tables using the partition method in the Oracle database is done when creating the table by selecting the partition type. To summarize long queries in generating reports and to make queries more efficient, the general ledger table can be summarized by creating a table view or materialized view.

### 2.2. Insert Data

After the tables have been created, the next step is to enter the research data in stages. Data comes from the original table which was taken in part to be included in the research table. The experiment was carried out in stages by increasing the amount of data from 29,976 rows to 10,002,137 rows. At each query execution, the time required to execute is recorded for further analysis. This aims to gain a deep understanding of query performance along with significant data growth.

### 2.3. Refresh Materialized View

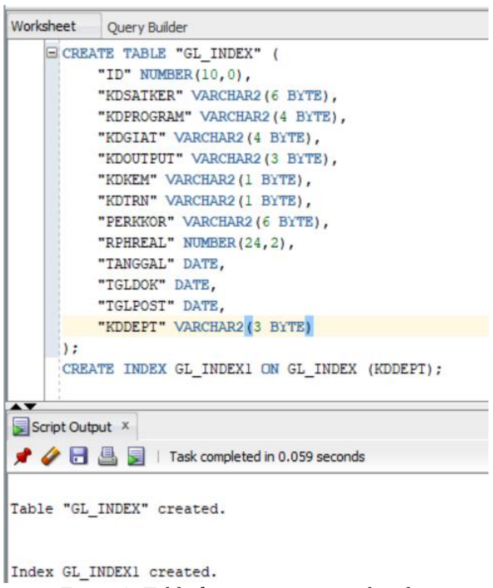
Each addition of data must be followed by updating the materialized view table. The view table does not require data refresh/update, because the data will automatically become the newest data when data is added to the GL table.

### 2.4. Run Query

In this stage, researchers choose three types of reports that are used as a basis for testing queries. The three types of reports are Financial Position Report (LPE), Operational Report (LO), and Balance Sheet. After running the query, the researcher recorded the time needed to run the query for each case.

The LPE report contains a report on changes in equity which contains a summary of equity and additional profits/losses from operational reports. The LO report contains operational reports containing income and expenses. The balance sheet report contains the composition of assets compared to the debts owned plus the amount of capital available.

## 3. RESULTS



```
CREATE TABLE "GL_INDEX" (
    "ID" NUMBER(10,0),
    "KDSATKER" VARCHAR2(6 BYTE),
    "KDPROGRAM" VARCHAR2(4 BYTE),
    "KDGIAT" VARCHAR2(4 BYTE),
    "KDOUTPUT" VARCHAR2(3 BYTE),
    "KDKEM" VARCHAR2(1 BYTE),
    "KDTRN" VARCHAR2(1 BYTE),
    "PERKKOR" VARCHAR2(6 BYTE),
    "RPHREAL" NUMBER(24,2),
    "TANGGAL" DATE,
    "TGLDOK" DATE,
    "TGLPOST" DATE,
    "KDDEPT" VARCHAR2(3 BYTE)
);
CREATE INDEX GL_INDEX1 ON GL_INDEX (KDDEPT);
```

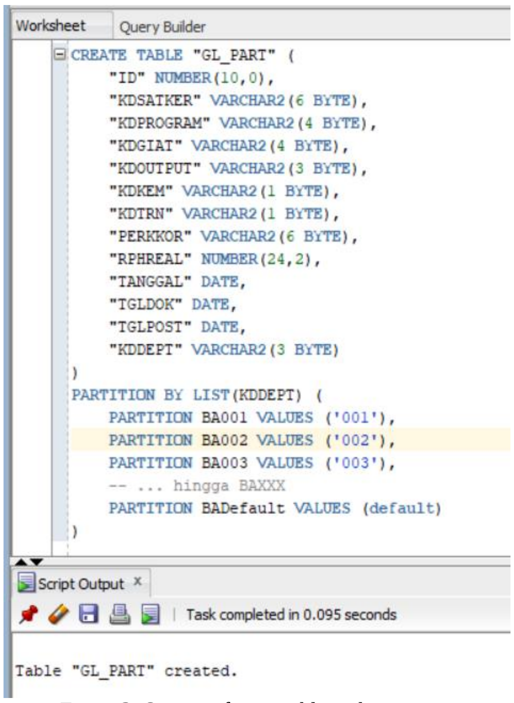
```
Table "GL_INDEX" created.


Index GL_INDEX1 created.
```

Figure 2. Table formation query with index

The research began by creating a general ledger table. In this research, the query for creating a general ledger table using the index method can be seen in Figure 2.

In this research, the partition chosen is a list partition with the kddept column as the partition column. The query for creating a general ledger table using the Parisi method can be seen in Figure 3.



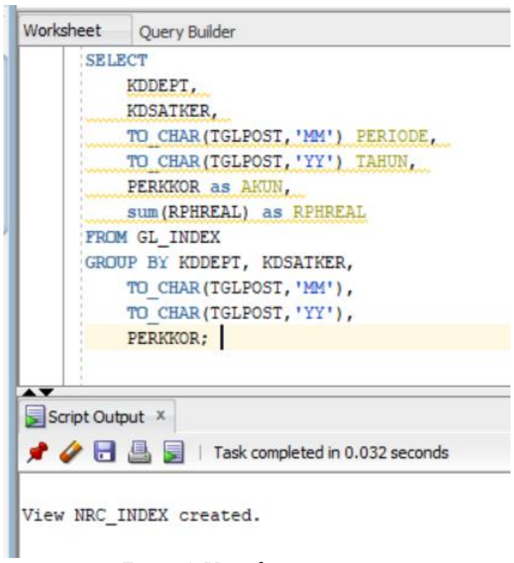
```
CREATE TABLE "GL_PART" (
    "ID" NUMBER(10,0),
    "KDSATKER" VARCHAR2(6 BYTE),
    "KDPROGRAM" VARCHAR2(4 BYTE),
    "KDGIAT" VARCHAR2(4 BYTE),
    "KDOUTPUT" VARCHAR2(3 BYTE),
    "KDKEM" VARCHAR2(1 BYTE),
    "KDTRN" VARCHAR2(1 BYTE),
    "PERKKOR" VARCHAR2(6 BYTE),
    "RPHREAL" NUMBER(24,2),
    "TANGGAL" DATE,
    "TGLDOK" DATE,
    "TGLPOST" DATE,
    "KDDEPT" VARCHAR2(3 BYTE)
)
PARTITION BY LIST(KDDEPT) (
    PARTITION BA001 VALUES ('001'),
    PARTITION BA002 VALUES ('002'),
    PARTITION BA003 VALUES ('003'),
    -- ... hingga BAXXX
    PARTITION BADefault VALUES (default)
)
```

```
Task completed in 0.095 seconds

Table "GL_PART" created.
```

Figure 3. Query to form a table with partitions

The basic query for creating a view table can be seen in Figure 4. With this query, 2 table views are formed, namely NRC_INDEX and NRC_PART for the index and partition methods.



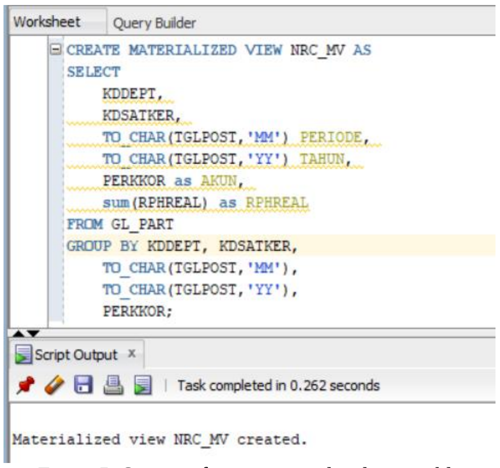
```
SELECT
    KDDEPT,
    KDSATKER,
    TO_CHAR(TGLPOST,'MM') PERIODE,
    TO_CHAR(TGLPOST,'YY') TAHUN,
    PERKKOR as AKUN,
    sum(RPHREAL) as RPHREAL
FROM GL_INDEX
GROUP BY KDDEPT, KDSATKER,
    TO_CHAR(TGLPOST,'MM'),
    TO_CHAR(TGLPOST,'YY'),
    PERKKOR;
```

```
Task completed in 0.032 seconds

View NRC_INDEX created.
```

Figure 4. View formation query

Meanwhile, creating a Materialized View table which is expected to improve query performance can be seen in Figure 5.



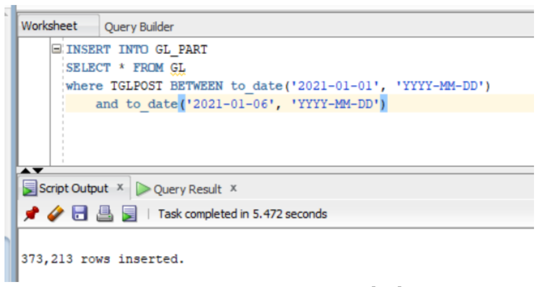
```
CREATE MATERIALIZED VIEW NRC_MV AS
SELECT
    KDDEPT,
    KDSATKER,
    TO_CHAR(TGLPOST,'MM') PERIODE,
    TO_CHAR(TGLPOST,'YY') TAHUN,
    PERKKOR as AKUN,
    sum(RPHREAL) as RPHREAL
FROM GL_PART
GROUP BY KDDEPT, KDSATKER,
    TO_CHAR(TGLPOST,'MM'),
    TO_CHAR(TGLPOST,'YY'),
    PERKKOR;
```

```
Task completed in 0.262 seconds

Materialized view NRC_MV created.
```

Figure 5. Query to form a materialized view table

The annual data from LKPP comprises over 100 million records each year. Summarizing such voluminous data poses a significant challenge due to the extensive time required. To address this, research was conducted by incrementally adding sample data and observing the corresponding time consumption. This approach enables the selection of the optimal method for handling large datasets. The basic query for retrieving and entering research data can be seen in Figure 6.



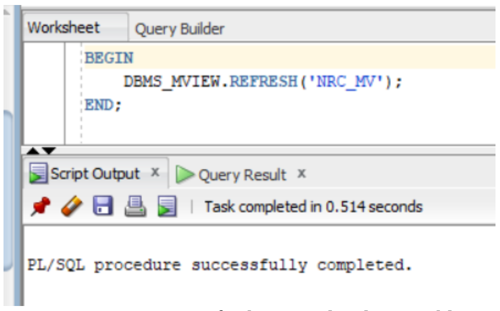
```
INSERT INTO GL_PART
SELECT * FROM GL
where TGLPOST BETWEEN to_date('2021-01-01', 'YYYY-MM-DD')
    and to_date('2021-01-06', 'YYYY-MM-DD')
```

```
Task completed in 5.472 seconds

373,213 rows inserted.
```

Figure 6. Query to insert sample data

Based on the experiments that have been carried out, to determine the speed performance or response time of queries, it is necessary to compare the experimental results using the index, partition and Materialized Views methods by comparing the time needed to run report generation queries on various amounts of data. The first thing you need to know is the time to refresh the materialized view.



```
BEGIN
    DBMS_MVIEW.REFRESH('NRC_MV');
END;
```

```
Task completed in 0.514 seconds

PL/SQL procedure successfully completed.
```

Figure 7. Query to refresh materialized view table

The query to refresh/update data in the materialized view is presented in Figure 7.

Table 1. Materialized view refresh time

| Raw Data | refresh Materialized View (seconds) |
|---|---|
| 29.976 | 0,707 |
| 72.550 | 0,72 |
| 137.163 | 0,875 |
| 258.695 | 1,472 |
| 468.408 | 6,97 |
| 750.630 | 8,457 |
| 999.213 | 10,287 |
| 1.335.663 | 13,604 |
| 1.772.295 | 20,123 |
| 2.108.382 | 23,072 |
| 2.502.068 | 30,762 |
| 2.957.039 | 33,286 |
| 3.523.279 | 35,289 |
| 4.263.159 | 46,475 |
| 5.032.893 | 64,788 |
| 5.860.514 | 78,46 |
| 6.801.181 | 91,037 |
| 7.764.000 | 108,645 |
| 8.833.705 | 122,077 |
| 10.002.137 | 138,466 |

In updating the data in the materialized view table, the time required experiences a significant spike with each addition of data. The time required for the amount of data 29,976 is 0.707 seconds, while the time required to refresh the data for the amount of data 10,002,137 is 138.466 seconds. Details of the time spike in refreshing the materialized view can be seen in Table 1.

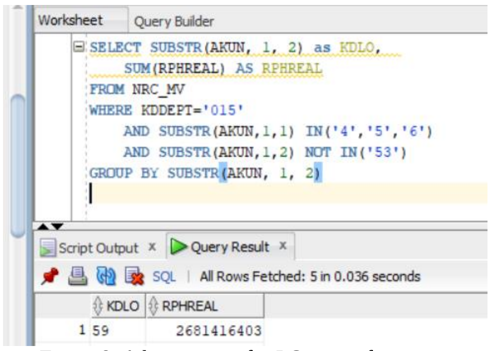

```
SELECT SUBSTR(AKUN, 1, 2) as KDLO,
    SUM(RPHREAL) AS RPHREAL
FROM NRC_MV
WHERE KDDEPT='015'
    AND SUBSTR(AKUN,1,1) IN('4','5','6')
    AND SUBSTR(AKUN,1,2) NOT IN('53')
GROUP BY SUBSTR(AKUN, 1, 2)
```
All Rows Fetched: 5 in 0.036 seconds
```
  KDLO RPHREAL
1 59      2681416403
```

Figure 8. A basic query for LO report formation

The basic query to be able to form an LO report is presented in Figure 8.

Table 2. LO reports experimental results (seconds)

| Raw Data | Index | Partition | MV |
|---|---|---|---|
| 29.976 | 0,079 | 0,069 | 0,022 |
| 72.550 | 0,99 | 0,112 | 0,018 |
| 137.163 | 0,183 | 0,175 | 0,022 |
| 258.695 | 0,299 | 0,333 | 0,021 |
| 468.408 | 0,454 | 0,47 | 0,046 |
| 750.630 | 0,658 | 0,594 | 0,052 |
| 999.213 | 0,916 | 0,822 | 0,069 |
| 1.335.663 | 1,236 | 1,084 | 0,078 |
| 1.772.295 | 1,725 | 1,435 | 0,118 |
| 2.108.382 | 2,042 | 1,87 | 0,13 |
| 2.502.068 | 2,537 | 1,996 | 0,163 |
| 2.957.039 | 2,976 | 2,337 | 0,153 |
| 3.523.279 | 3,407 | 3,002 | 0,183 |
| 4.263.159 | 5,255 | 3,526 | 0,238 |
| 5.032.893 | 5,982 | 4,094 | 0,262 |
| 5.860.514 | 7,337 | 4,964 | 0,278 |
| 6.801.181 | 7,81 | 5,831 | 0,349 |
| 7.764.000 | 8,396 | 6,143 | 0,49 |
| 8.833.705 | 12,021 | 7,796 | 0,514 |
| 10.002.137 | 15,199 | 7,807 | 0,572 |

The query execution time to form the LO report using the table view with the indexing method takes 0.079 seconds, for the partitioning method, it takes 0.069 seconds while the materialized view table is 0.022. The execution time of this query increases along with data growth, where the data amount is 10,002,137 table rows, with the indexing method the time required is 15.199 seconds, the partitioning method takes 7.807 seconds while the materialized view table is 0.572. In Table 2, it can be seen that the time required increases along with the growth in the amount of data, especially in the index and partition methods.

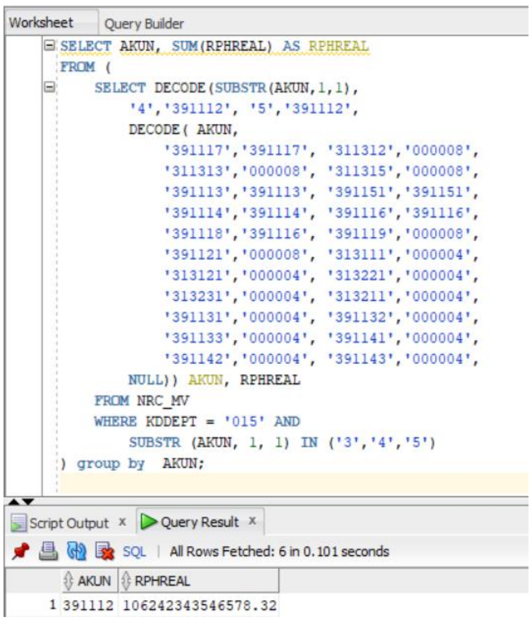

```
SELECT AKUN, SUM(RPHREAL) AS RPHREAL
FROM (
    SELECT DECODE(SUBSTR(AKUN,1,1),
        '4','391112', '5','391112',
        DECODE( AKUN,
            '391117','391117', '311312','000008',
            '311313','000008', '311315','000008',
            '391113','391113', '391151','391151',
            '391114','391114', '391116','391116',
            '391118','391116', '391119','000008',
            '391121','000008', '313111','000004',
            '313121','000004', '313221','000004',
            '313231','000004', '313211','000004',
            '391131','000004', '391132','000004',
            '391133','000004', '391141','000004',
            '391142','000004', '391143','000004',
        NULL)) AKUN, RPHREAL
    FROM NRC_MV
    WHERE KDDEPT = '015' AND
        SUBSTR (AKUN, 1, 1) IN ('3','4','5')
) group by  AKUN;
```
All Rows Fetched: 6 in 0.101 seconds
```
  AKUN   RPHREAL
1 391112 106242343546578.32
```

Figure 9. A basic query for generating LPE reports

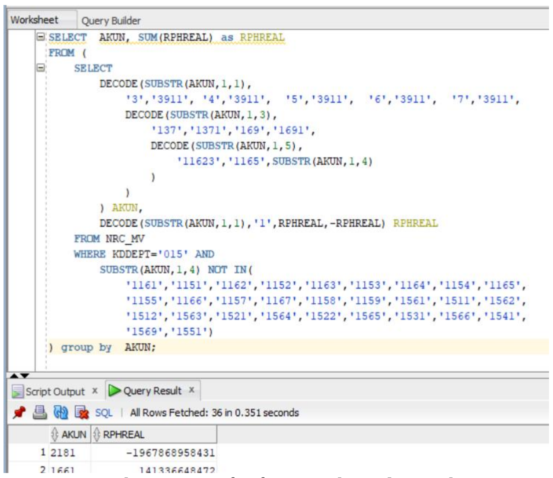
The basic query to be able to form an LPE report is presented in Figure 9.

Table 3. LPE reports experimental results (seconds)

| Raw Data | Index | Partition | MV |
|---|---|---|---|
| 29.976 | 0,058 | 0,061 | 0,022 |
| 72.550 | 0,105 | 0,11 | 0,013 |
| 137.163 | 0,165 | 0,205 | 0,024 |
| 258.695 | 0,291 | 0,359 | 0,026 |
| 468.408 | 0,587 | 0,42 | 0,051 |
| 750.630 | 0,761 | 0,642 | 0,059 |
| 999.213 | 1,142 | 0,839 | 0,062 |
| 1.335.663 | 1,391 | 1,26 | 0,069 |
| 1.772.295 | 1,826 | 1,485 | 0,101 |
| 2.108.382 | 2,099 | 1,768 | 0,103 |
| 2.502.068 | 2,516 | 2,237 | 0,117 |
| 2.957.039 | 3,054 | 2,364 | 0,129 |
| 3.523.279 | 4,386 | 3,116 | 0,135 |
| 4.263.159 | 4,582 | 3,682 | 0,196 |
| 5.032.893 | 6,332 | 4,26 | 0,217 |
| 5.860.514 | 6,111 | 4,91 | 0,221 |
| 6.801.181 | 8,28 | 5,655 | 0,288 |
| 7.764.000 | 7,32 | 6,194 | 0,317 |
| 8.833.705 | 9,191 | 7,329 | 0,37 |
| 10.002.137 | 10,814 | 8,17 | 0,354 |

The query execution time to form an LPE report using a table view with the indexing method takes 0.058 seconds, for the partitioning method, it takes 0.061 seconds while the materialized view table is

0.022. The execution time of this query increases along with data growth, where the amount of data is 10,002,137 rows in the table view using the indexing method. The time required is 10.814 seconds, the partitioning method takes 8.17 seconds while the materialized view table is 0.354. In Table 3, it can be seen that the time required increases along with the growth in the amount of data, especially in the index and partition methods.



```
SELECT  AKUN, SUM(RPHREAL) as RPHREAL
FROM (
    SELECT
        DECODE(SUBSTR(AKUN,1,1),
            '3','3911', '4','3911',  '5','3911',  '6','3911',  '7','3911',
            DECODE(SUBSTR(AKUN,1,3),
                '137','1371','169','1691',
                DECODE(SUBSTR(AKUN,1,5),
                    '11623','1165',SUBSTR(AKUN,1,4)
                )
            )
        ) AKUN,
        DECODE(SUBSTR(AKUN,1,1),'1',RPHREAL,-RPHREAL) RPHREAL
    FROM NRC_MV
    WHERE KDDEPT='015' AND
        SUBSTR(AKUN,1,4) NOT IN(
            '1161','1151','1162','1152','1163','1153','1164','1154','1165',
            '1155','1166','1157','1167','1158','1159','1561','1511','1562',
            '1512','1563','1521','1564','1522','1565','1531','1566','1541',
            '1569','1551')
) group by  AKUN;
```

```
All Rows Fetched: 36 in 0.351 seconds

    AKUN  RPHREAL
  1 2181     -1967868958431
  2 1661      141336648472
```

Figure 10. A basic query for forming the Balance Sheet report

The basic query to be able to form a Balance Sheet report is presented in Figure 10.

Table 4. Results of the Balance Sheet Experiment (seconds)

| Raw Data | Index | Partition | MV |
|---|---|---|---|
| 29.976 | 0,128 | 0,151 | 0,042 |
| 72.550 | 0,306 | 0,316 | 0,04 |
| 137.163 | 0,572 | 0,582 | 0,049 |
| 258.695 | 0,942 | 0,963 | 0,069 |
| 468.408 | 1,333 | 1,354 | 0,151 |
| 750.630 | 2,291 | 2,188 | 0,184 |
| 999.213 | 3,184 | 3,011 | 0,212 |
| 1.335.663 | 4,557 | 4,581 | 0,238 |
| 1.772.295 | 5,723 | 5,221 | 0,362 |
| 2.108.382 | 6,794 | 6,374 | 0,467 |
| 2.502.068 | 8,111 | 7,333 | 1,385 |
| 2.957.039 | 9,784 | 8,599 | 1,478 |
| 3.523.279 | 11,425 | 11,162 | 1,845 |
| 4.263.159 | 14,137 | 13,052 | 2,552 |
| 5.032.893 | 18,096 | 15,212 | 2,79 |
| 5.860.514 | 19,652 | 17,757 | 2,634 |
| 6.801.181 | 22,309 | 20,347 | 3,629 |
| 7.764.000 | 24,491 | 21,957 | 3,765 |
| 8.833.705 | 28,446 | 26,281 | 4,643 |
| 10.002.137 | 31,316 | 27,884 | 4,981 |

The query execution time to form the Balance Sheet report using the table view with the indexing method takes 0.128 seconds, for the partitioning method, it takes 0.151 seconds while the materialized view table is 0.042. The execution time of this query increases along with data growth, where the amount of data is 10,002,137 rows in the table view using the indexing method. The time required is 31.316 seconds, the partitioning method takes 27.884 seconds while the materialized view table is 4.981. In Table 5, it can be seen that the time required increases along with the growth in the amount of data, especially in the index and partition methods.

## 4. DISCUSSION

From the research that has been carried out, it can be seen that the query execution time on a limited amount of data results in a relatively short time. However, as the amount of data increases, it can be seen that the time required to run the query also becomes longer.

Research conducted by Samidi [9] proves that the performance of tables with partitions is better than tables with indexes, this is in line with the results of this study. Apart from that, research conducted by Piotr Bednarczuk [7] also underlines the advantages of using partitions on large-scale datasets.

However, the performance of tables with materialized views is more optimal performance than index and partition methods. This finding is consistent with the results of previous studies, such as those reported by Almeida [14] and Witono [16], who each chose to use materialized views in certain contexts.

In this research, the results of the query to form the basic numbers for the balance sheet still took more than 3 seconds for data of more than 6.8 million raw data. This is less than ideal if you only use the materialized view of the NRC_MV table. For this reason, it is necessary to create a separate materialized view for the balance sheet report so that query report formation can be faster. This applies to other reports that require long query times.

Forming a materialized view table requires quite a significant amount of time if the amount of data increases. One way to overcome the lag time when the materialized view table is being created is to use synonyms.

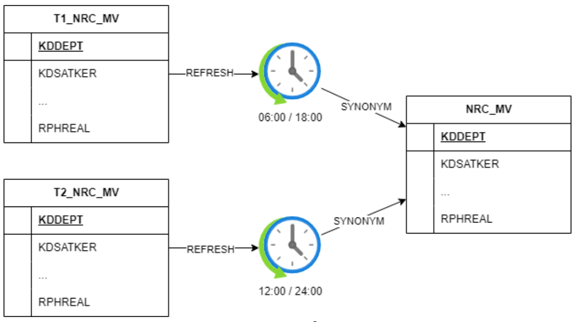


Figure 11. Use of synonyms

Synonyms can be used by creating 2 materialized view tables. In Figure 11, an example is created by creating T1_NRC_MV which will be refreshed at 06:00 and 18:00, as well as the T2_NRC_MV table which will be refreshed at 12:00 and 24:00. After the refresh is complete, the T1 / T2 tables will be synonymized with the NRC_MV table so that there are no data gaps due to the materialized view refresh process.

## 5. CONCLUSION

Based on the results of experiments carried out, there are findings that the query execution time on materialized view tables shows faster performance compared to tables that use indexing or partitioning methods. These results indicate that materialized views can provide significant advantages when faced with large volumetric data. The decision to choose a materialized view can be considered contextually, depending on the specific needs and characteristics of the data encountered in a database system.

As a suggestion for future development, you can consider the application of cache technology in preparing financial reports. Implementing a cache can provide a solution to improve performance by storing previous query results in memory, thereby reducing query execution time on frequently accessed financial reports. In addition, the use of cache can provide flexibility in handling data changes in source tables, by periodically synchronizing the cache according to business needs.

## REFERENCES

[1] Kementerian Keuangan. “Laporan Keuangan Pemerintah Pusat Tahun 2020 Audited.” Kementerian Keuangan, 2020.

[2] N. M. Khushairi, N. A. Emran, and M. M. M. Yusof, "Database performance tuning methods for manufacturing execution system." *World Applied Sciences Journal*, *30* (30 A), 2014, doi: 10.5829/idosi.wasj.2014.30.icmrp.14.

[3] K. Mózsi, and A. Kiss, "A session-based approach to autonomous database tuning." *Acta Polytechnica Hungarica*, vol. 17, no. 1, 2020, doi: 10.12700/APH.17.1.2020.1.1.

[4] KARAGKOUNI, Dimitra, et al. "DIANA-LncBase v3: indexing experimentally supported miRNA targets on non-coding transcripts." *Nucleic acids research*, 48.D1: D101-D110, 2020.

[5] R. Chopade and V. Pachghare, "MongoDB Indexing for Performance Improvement." *Advances in Intelligent Systems and Computing*, p. 1077, 2020, doi: 10.1007/978-981-15-0936-0_56.

[6] A. D. Fuentes, A. C. Almeida, R. L. de C. Costa, V. Braganholo, and S. Lifschitz, "*Database Tuning with Partial Indexes*.", 2020, doi: 10.5753/sbbd.2018.22229.

[7] P. Bednarczuk, "OPTIMIZATION IN VERY LARGE DATABASES BY PARTITIONING TABLES." *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Srodowiska*, vol. 10, no. 3, 2020, doi: 10.35784/iapgos.2056.

[8] P. Bednarczuk and A. Borsuk, "EFFICIENTLY PROCESSING DATA IN TABLE WITH BILLIONS OF RECORDS." *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Srodowiska*, vol. 12, no. 4, 2020, doi: 10.35784/iapgos.3058.

[9] Samidi, Fadly, Y. Virmansyah, R. Y. Suladi, and A. B. Lesmana, "Optimasi Database dengan Metode Index dan Partisi Tabel Database Postgresql pada Aplikasi E-Commerce. Studi pada Aplikasi Tokopintar." *Jurnal Pendidikan Tambusai*, vol. 6, no. 1, 2022.

[10] M. Kechar and L. Bellatreche, "Safeness: Suffix Arrays Driven Materialized View Selection Framework for Large-Scale Workloads." *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 13428 LNCS*, 2022, doi: 10.1007/978-3-031-12670-3_7.

[11] G. Li, X. Zhou, J. Sun, X. Yu, Y. Han, L. Jin, W. Li, T. Wang, and S. Li, "Opengauss: An autonomous database system." *Proceedings of the VLDB Endowment*, vol. 14, no. 12, 2021, doi: 10.14778/3476311.3476380.

[12] R. Ahmed, R. Bello, A. Witkowski, and P. Kumar, "Automated Generation of Materialized Views in Oracle." *Proceedings of the VLDB Endowment*, vol. 13, no. 12, 2020 doi: 10.14778/3415478.3415533.

[13] M. Bandle, J. Giceva, and T. Neumann, "To Partition, or Not to Partition, That is the Join Question in a Real System." *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2021, doi: 10.1145/3448016.3452831.

[14] A. C. Almeida, F. Baião, S. Lifschitz, D. Schwabe, and M. L. M. Campos, "Tun-OCM: A model-driven approach to support database tuning decision making." *Decision Support Systems*, p. 145, 2021, doi: 10.1016/j.dss.2021.113538.

[15] M. Malcher and D. Kuhn, "Views, Synonyms, and Sequences." In *Pro Oracle Database 18c Administration*, 2019, doi: 10.1007/978-1-4842-4424-1_9.

[16] E. Witono and Parno. "Perbandingan Response Time Penggunaan Index, Views, dan Materialized Views Database Mysql." *Jurnal Sains Komputer & Informatika (J-SAKTI*, vol. 6, no. 1, 2022.