

GRADIENT ANALYSIS IN IMPLEMENTATION OF B-TREE INDEXING IN REPORTING ANNUAL TAX DATABASE

Samidi, Shofinurdin, Andra Setiadi, Danar Darmawan, Dika Andharu

Universitas Budi Luhur, Indonesia

Email: samidi.indonesia@gmail.com, shofinurdin@gmail.com,
andrasetiadi85@gmail.com, danard.darmawan@gmail.com,
dikaandharu04@gmail.com

Abstract

A query is a syntax or command used in a database system to access and display data. Queries can be used to make data interact with each other. To display query results in the database, of course, requires execution time which is usually denoted in seconds. Execution time is directly proportional to the amount of data to be displayed and the level of complexity of the database. To speed up query execution time, the term database optimization is known. One of the database optimization methods is to use the b-tree indexing technique the database. This study aims to compare the execution time of databases that have not been indexed or that have been indexed with data objects in the MySQL database MPN-Info application at the Jakarta Palmerah Tax Office (KPP Pratama Jakarta Palmerah). This application was developed to supervise taxpayers which is used in almost all tax service offices throughout Indonesia. The data used is active employee status taxpayer data registered until 2020 with the reporting of Annual Personal Income Tax Returns in 2019 and 2020. The study uses an experimental method by applying a select query, then the execution time is recorded well for unindexed and indexed databases. The gradient method will be used for compare the results of the two. The results obtained are after taking the population of taxpayers registered employees up to year In 2020 at KPP Pratama Jakarta Palmerah, 31,238 data were compared with the annual reporting data, the average execution time before indexing was 255.585 seconds and the average time after indexing was 1.341 seconds and the gradient value before indexing was 0.0211 and after indexing was 0.0001. This proves that the indexing technique has a significant impact in accelerating the query execution process.

Keywords: queries; mysql; b-tree indexing; gradient; database optimization

Introduction

Today there are many applications that use databases, either free or paid, such as MySQL, Postgree SQL, Oracles and others. One of the functions of the database is to group data and simplify the process of identifying data. The database will display data according to user requests with a fast process using Database Management Systems (DBMS) software. Another function is to facilitate access, edit, add, delete and store data.

Among the applications that use the database is the MPN-info (State Revenue Module) application. MPN-Info is an application developed by Ichdyan Thalasa and Yogi Iskandar and used by *Account Representative* (Tax Supervisor) in supervising taxpayers. This application is used in almost all tax service offices throughout Indonesia. The desktop-based MPN-info application is deployed using C++ language while the database uses MySQL.

Taxpayer supervision is carried out, among others, by supervising taxpayer reporting compliance. This reporting compliance data can be retrieved from the mpn-info database by executing a select query that pairs the masterfile table and the annual_report table. When tested on the query turns out to take a long time. To retrieve a data value of 100 records takes 18.5 seconds and for 30,000 data records it takes 650 seconds. This causes dissatisfaction and the question arises whether the execution time can be accelerated or not.

Some theories state that using an index, namely giving an identifier to the data in the database can streamline the search process, including (Elmasri & Navathe, 2010) explains that index used in the database can speed up data retrieval when running queries. Balasubramanian et al (Balasubramanian & Sabharwal, 2013) also argue that information retrieval can be made more efficient by using indexes to provide quick access to databases. Furthermore, Guzun et al (Guzun & Canahuate, 2016) suggested that to support query efficient mechanism is needed right indexing.

In the indexing technique there are also several methods that can be used, one of which is the B-Tree method. B-Tree is a one-dimensional indexing method and is a nested hierarchical indexing method for data access from peripheral data stores. B-Tree is a tree data structure where each leaf has the same height. B-Tree was first created by Rudolf Bayer and Ed McCreight in 1972. B-Tree was made possible to store a lot of data in one node, the number of subtrees can also be very large. For this reason, B-Tree is very suitable for use in managing data on disk (Mushofan, 2014).

Several studies that have discussed the use of indexing include: Putra et al (Putra, Darwiyanto, & Gozali, 2015) who implemented B-Tree to perform full text indexing on electronic documents. In his research, Putra et al conducted indexing of text documents. Then Dongoran et al (Dongoran, Saleh, & Gozali, 2015) also conducted research related to indexing. In their research, Dongoran et al discusses the application of indexing to graph databases. Graph database itself is a database representation using a different graph with a general relational database. Next Huda Ayesha Mashaan Alrashidi (Alrashidi & Farhan, 2011) on his research six index techniques, namely B-tree, reverse, organization, clustered, non-clustered and bitmap on Oracle and MS SQL Server. Then Ammar et al (A. Ammar, M. Zainuri Sarringan, S. A-mostafa, A. Mustapha, 2020) in their research discussed the differences in the application of indexing with the B-Tree and Hash Map methods using the MySQL and PostgreSQL platforms. Aminudin et al (Mostafa, 2020) also conducts research related to indexing. In their research, Aminudin et al implements B-Tree indexing for PostgreSQL databases. Amminudin et al

recommend further investigation such as improved algorithm of existing B-Tree where B-Tree tends to have longer time against large amount of data.

The studies mentioned above have implemented indexing techniques but none have compared the slope/ gradient value to query execution time with different data both before indexing and after indexing. As for this study, besides comparing the difference in the average execution time before and after indexing with the B-Tree indexing technique, it also analyzes the value of the slope/trend of the relationship between the amount of data and its execution time using gradient analysis.

Gradient analysis is a method for comparing trend patterns from several locations by observing the level of slope of the line connecting the two variables. The gradient or direction coefficient (m) is a constant that indicates the level of slope of a line (Handajani, 2009). Look at the following picture:

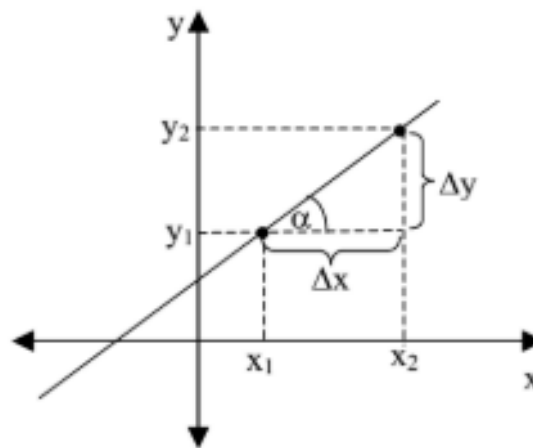


Figure 1
Gradient

The commonly used mathematical formula is $y = mx + c$, where $m, c \in \mathbb{R}$, c are constants (intercepts), where m represents the gradient (slope) of the coefficient of the straight line.

The purpose of this study is to prove the database indexing theory by comparing the query execution speed in the mpn-info database before indexing with after indexing, and to see the difference in the slope/trend of execution speed for querying data with different amounts using the gradient analysis method.

Research Method

This study uses an experimental method by applying a select query to the MPN-Info application database and then recording the execution time for both unindexed and deindexed databases for later analysis.

The scope of this research is to retrieve data from the MySQL database in the MPN-Info application. The query used is a query to find taxpayer data on the status of active employees registered in 2020 and earlier with annual SPT reporting 2019 and

2020, the query is run to generate results record 100, 500, 1000, 5000, 10000, 15000, 20000, 25000 and 30000 data, the index technique used is the index with B-Tree, the method used to compare the level of slope is the gradient analysis method with the mathematical formula $y = mx + c$.

This research was conducted using a macbook pro mid 2012 laptop with specifications for Intel Core i5-3210M CPU @ 2.50GHz, 16 GB RAM, Windows 10 Pro 64 bit OS, while the tools used are SQLYog Professional for GUI databases, Microsoft Excel for visualization, Jupyter notebook for visualization and modelling. The steps taken in this study after understanding the problem and literature review are data collection, query creation, query result analysis before indexing, indexing, query analysis after indexing and gradient analysis.

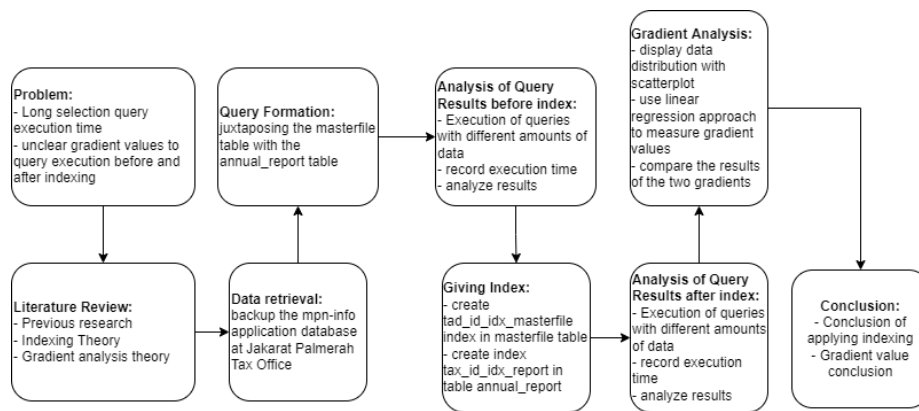


Figure 1
Research Framework

Discussion

A. Data Collection

This study uses the MySQL database MPN-Info application at KPP Pratama Jakarta Palmerah (Jakarta Palmerah Tax Office). Data retrieval is taken by backuping manually using the SQLYog application.

B. Query Creation

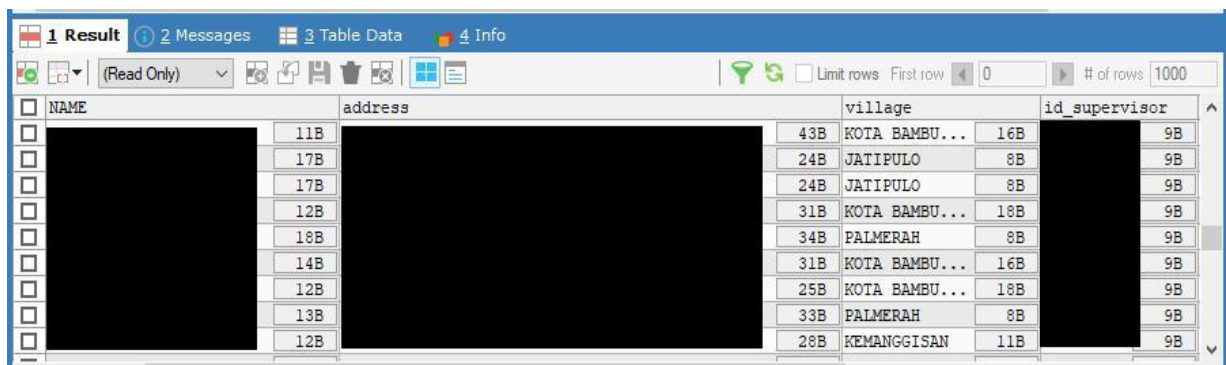
The selection query to retrieve individual taxpayer data on employee status registered up to 2020 and annual tax returns for the 2019 and 2020 tax years is by juxtaposing the taxpayer *masterfile* table and the *annual report* transaction table.

Gradient Analysis In Implementation of B-Tree Indexing In Reporting Annual Tax Database

```
1 SELECT a.*, b.tax_year,b.number,b.methode,c.tax_year,c.number,c.methode FROM
2 (SELECT tax_id,office,branch,NAME,address,village,id_supervisor,business_classification,register_date,
3 YEAR(register_date) FROM masterfile
4 WHERE TYPE='OP'
5 AND STATUS='Normal'
6 AND business_classification LIKE '963%'
7 AND branch ='000'
8 AND YEAR(register_date)<'2021') a
9 LEFT JOIN
10 (SELECT tax_id,office,branch,number,TYPE,tax_year,STATUS,methode FROM annual_report
11 WHERE TYPE LIKE 'SPT Tahunan PPh Orang%'
12 AND tax_year = 2019) b
13 ON a.tax_id=b.tax_id AND a.office=b.office AND a.branch=b.branch
14 LEFT JOIN
15 (SELECT tax_id,office,branch,number,TYPE,tax_year,STATUS,methode FROM annual_report
16 WHERE TYPE LIKE 'SPT Tahunan PPh Orang%'
17 AND tax_year = 2020) c
18 ON a.tax_id=c.tax_id AND a.office=c.office AND a.branch=c.branch
```

Figure 3
Query Selection

Furthermore, from the query execution, the results can be seen in the following figure:



NAME	address	village	id_supervisor
	11B	43B KOTA BAMBU...	16B 9B
	17B	24B JATIPULO	8B 9B
	17B	24B JATIPULO	8B 9B
	12B	31B KOTA BAMBU...	18B 9B
	18B	34B PALMERAH	8B 9B
	14B	31B KOTA BAMBU...	16B 9B
	12B	25B KOTA BAMBU...	18B 9B
	13B	33B PALMERAH	8B 9B
	12B	28B KEMANGGISAN	11B 9B

Figure 4
Query Execution Results

C. Analysis of Query Results before indexing

Sequentially by executing queries for databases that have not applied indexing to different amounts of data, the execution time can be seen in the graphic image as follows:

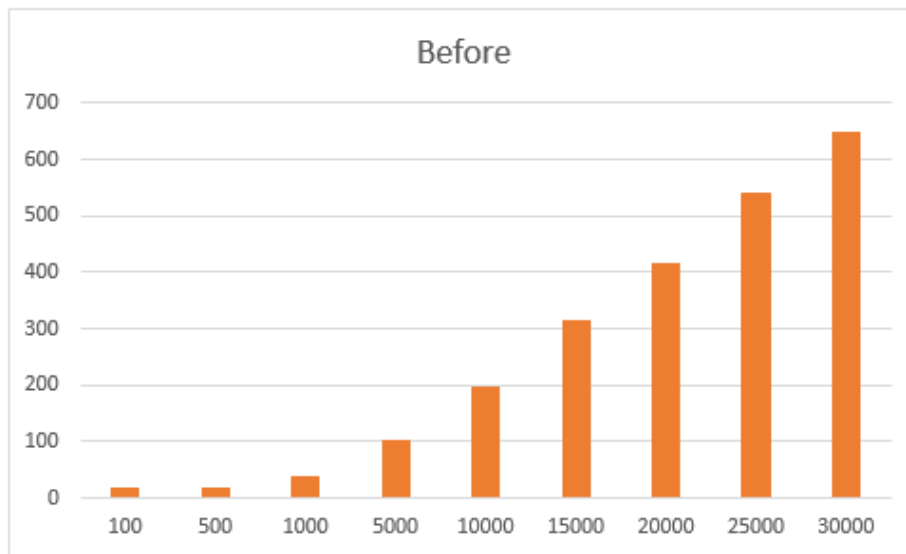


Figure 5
Graph of Query Results Before Indexing

Based on the graphic image, we can see that the more data displayed, the more query execution time is required.

D. Giving Index

Giving an index on the masterfile table with the name `tax_id_idx_masterfile` with the command:

```
CREATE INDEX tax_id_idx_masterfile ON masterfile(tax_id,office,branch) USING BTREE
```

Giving an index to the `annual_report` table with the name `tax_id_idx_report` with the command:

```
CREATE INDEX tax_id_idx_report ON annual_report(tax_id,office,branch) USING BTREE
```

E. Query Analysis After Indexing

After indexing the database, the query execution is carried out and the results can be seen in the graphic image:

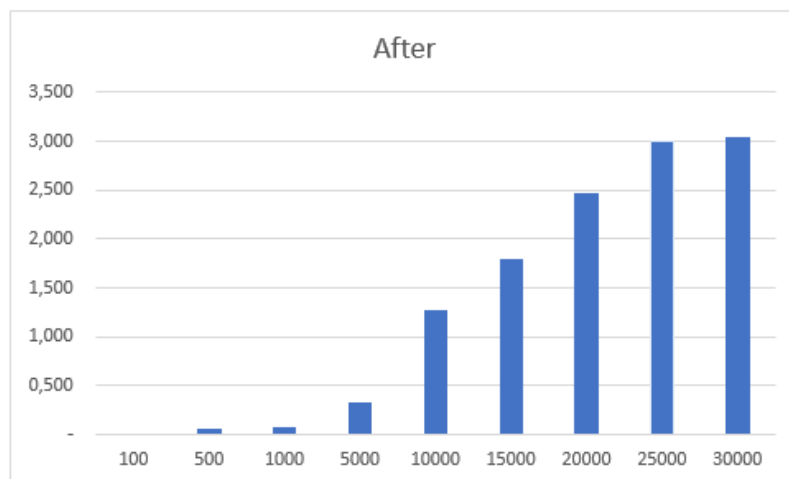


Figure 6
Graph of Query Results After Indexing

Based on Figure 6, we can see that the more data displayed, the more query execution time is needed, this looks the same as the query execution results before indexing, but for more details it can be seen the average execution time of both in the following table and graph:

Table 1
Query execution time

<i>Amount of data retrieved (records)</i>	<i>Time (seconds)</i>	
	<i>After indexing</i>	<i>Before indexing</i>
<i>100</i>	<i>0.015</i>	<i>18.49</i>
<i>500</i>	<i>0.057</i>	<i>19,265</i>
<i>1000</i>	<i>0.082</i>	<i>39,511</i>
<i>5000</i>	<i>0.328</i>	<i>103</i>
<i>10000</i>	<i>1,267</i>	<i>197</i>
<i>15000</i>	<i>1,791</i>	<i>316</i>
<i>20000</i>	<i>2,466</i>	<i>418</i>
<i>25000</i>	<i>3.010</i>	<i>539</i>
<i>30000</i>	<i>3.052</i>	<i>650</i>
<i>Average</i>	<i>1.341</i>	<i>255.585</i>

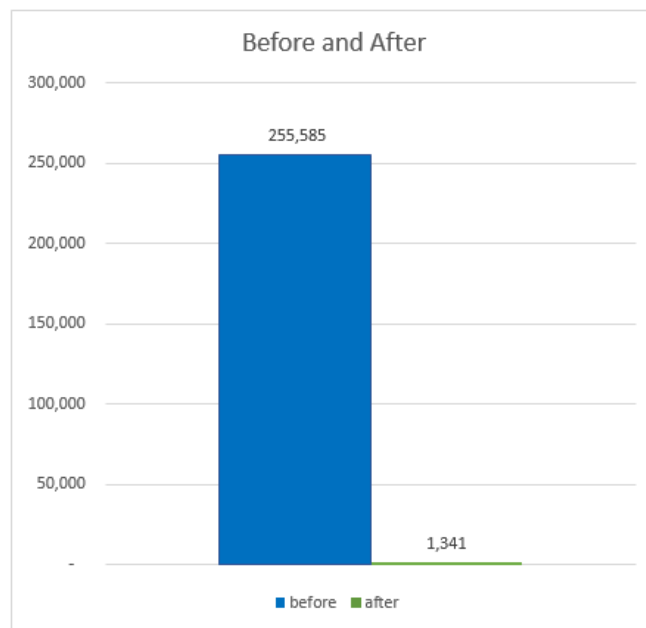


Figure 7
Comparison Graph of Average
Query Execution Time

It can be seen in table 1 and figure 6 that the average execution time before indexing is 255.585 seconds and the average execution time after indexing is 1.341 seconds. There is a significant time difference of 254.244 seconds.

F. Testing With Gradient Analysis

As previously explained, the query execution time before and after indexing increases when more data is to be displayed. To test the increase in both times, the researcher chose the gradient analysis method to test it. The researcher uses a linear regression mathematical model approach to find the gradient value. By using *scatter plot library* In the python programming language, the results of the distribution of data and gradient lines are as follows:


```
In [12]: # visualize the result
df1.plot.scatter(x='data', y='before')
plt.plot(df1['data'], lm.fittedvalues, c='red')

Out[12]: [<matplotlib.lines.Line2D at 0x2342a15e460>]
```

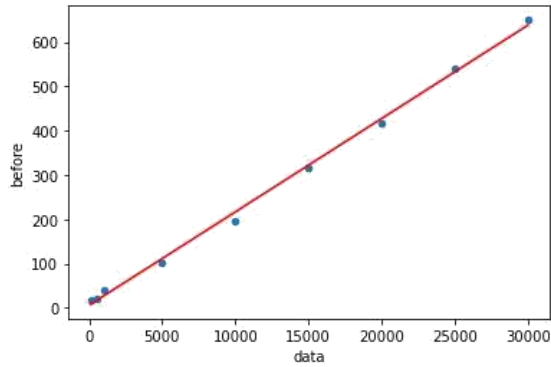


Figure 8
Distribution of Data and Gradient Lines Before Indexing

```
In [18]: # visualize the result
df1.plot.scatter(x='data', y='after')
plt.plot(df1['data'], lm.fittedvalues, c='red')

Out[18]: [<matplotlib.lines.Line2D at 0x2342a32efd0>]
```

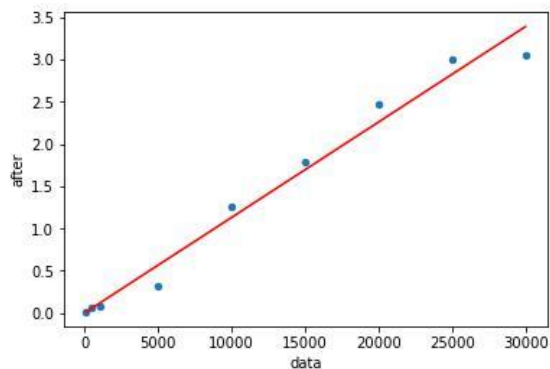


Figure 9
Distribution of Data and Gradient Lines After Indexing

From the two images it is not clear the difference between the two, the researcher then uses Microsoft excel to display the two linear lines in one graph to see the difference as shown in the picture:

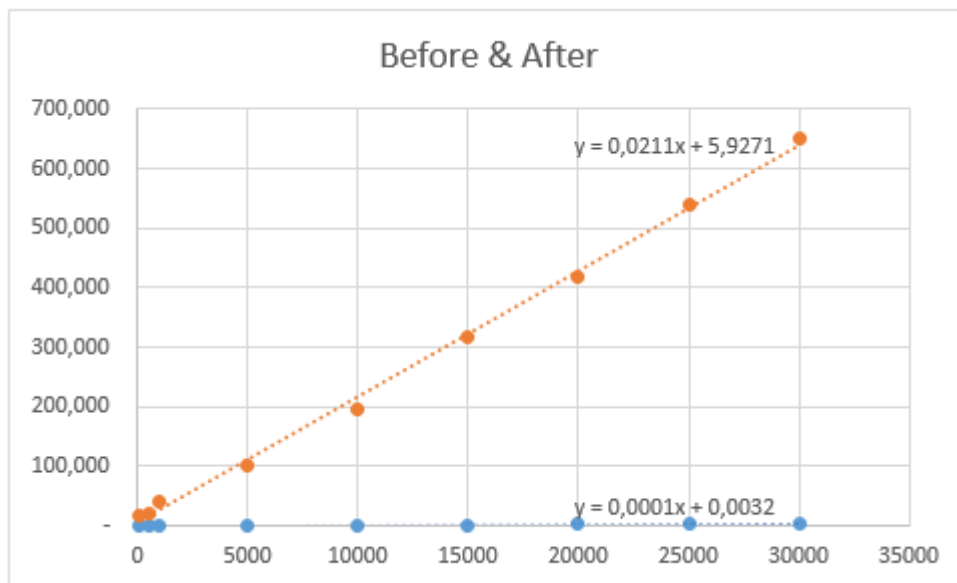


Figure 10
Gradient Comparison

From the figure 10 the equation of the mathematical formula before indexing can be taken $Y = 0.0211X + 5.9271$ and after indexing $Y = 0.0001X + 0.0032$. It can be seen that the gradient and constant values after indexing are smaller than before indexing with a significant difference, namely 0.021 for gradient and 5.9239 for constant.

Conclusion

From the results of this study, it was found that using indexing in the database can speed up query execution time compared to not using indexing, with an average execution time difference of 254.244 seconds with average breakdown 1.341 seconds after indexing and average 255.585 seconds before indexing. In this study also obtained a significant difference in the results of the gradient 0.021 of the query data before and after using indexing where the mathematical formula is obtained before indexing: $Y = 0.0211X + 5.9271$ and after indexing : $Y = 0.0001X + 0.0032$. This proves that the indexing technique has a significant impact in accelerating the query execution process Researchers suggest for management and developers of MPN-info applications to use the B-Tree indexing technique so that the query execution process will be faster, and for further research to compare the B-Tree indexing method with other methods.

BIBLIOGRAFI

- A. Ammar, M. Zainuri Sarringan, S. A-mostafa, A. Mustapha, and S. Hamad Khaleefah. (2020). *Analyzing the Effect of Data Size Variation on the Performance of B-Tree and Hash Map Indexing in MySQL*. 7(12). [Google Scholar](#)
- Alrashidi, Huda Ayesah Mashaan, & Farhan, Hazim A. (2011). *A Comparative Study of Indexing Techniques for Relational Database Management Systems*. Middle East University. [Google Scholar](#)
- Balasubramanian, Meiyalagan, & Sabharwal, Rohit. (2013, July 16). *Dynamic integrated database index management*. Google Patents. [Google Scholar](#)
- Dongoran, Emir Septian Sori, Saleh, W. Kemas Rahmat, & Gozali, Alfian Akbar. (2015). Analysis and implementation of graph indexing for graph database using GraphGrep algorithm. *2015 3rd International Conference on Information and Communication Technology (ICoICT)*, 59–64. IEEE. [Google Scholar](#)
- Elmasri, Ramez, & Navathe, Shamkant B. (2010). *Fundamentals of Databases*. Addison-Wesley. [Google Scholar](#)
- Guzun, Gheorghii, & Canahuate, Guadalupe. (2016). Hybrid query optimization for hard-to-compress bit-vectors. *The VLDB Journal*, 25(3), 339–354. [Google Scholar](#)
- Handajani, Mudjiastuti. (2009). Analisis Gradien Kepadatan Penduduk dan Konsumsi BBM. *Jurnal Teknik Sipil Dan Perencanaan*, 11(2), 141–148. [Google Scholar](#)
- Mostafa, Salama A. (2020). A Case Study on B-Tree Database Indexing Technique. *Journal of Soft Computing and Data Mining*, 1(1), 27–35. [Google Scholar](#)
- Mushofan, A. (2014). “B-trees and their application in databases,.” [Google Scholar](#)
- Putra, Diken Pradana, Darwiyanto, Eko, & Gozali, Alfian Akbar. (2015). Implementasi Fulltext Indexing pada Dokumen Elektronik dengan Algoritma B-Tree. *EProceedings of Engineering*, 2(1). [Google Scholar](#)

Copyright holder:

Samidi, Shofinurdin, Andra Setiadi, Danar Darmawan, Dika Andharu (2022)

First publication right:

Syntax Literate: Jurnal Ilmiah Indonesia

This article is licensed under:

