ISSN: 2614-6754 (print)
ISSN: 2614-3097(online)

Database Tuning in Hospital Applications Using Table Indexing and Query Optimization

Samidi[1], Daniel Iskandar[2], Muhamad Fachruroji[3], Wahyu Adi Septyo Wibowo[4], Afifah Khaerani A[5]
[1,2,3,4,5] Fakultas Teknologi Informasi, Universitas Budi Luhur
e-mail: samidi.indonesia@gmail.com[1], danielisk2000@gmail.com[2], aku1712@gmail.com[3], wahyuadisetyowibowo@gmail.com[4], afifahkhaerani04@gmail.com[5]

Abstrak

RDBMS telah menjadi kebutuhan mendasar bagi semua perusahaan dan diharapkan dapat memberikan informasi secara tepat waktu, akurat, dan andal. Sayangnya, pertumbuhan data yang eksponensial biasanya menyebabkan masalah kinerja database, yaitu waktu eksekusi kueri yang lambat sebagai salah satu masalah utama. Pilihan untuk meningkatkan kinerja database adalah dengan melakukan penyetelan database menggunakan pengindeksan tabel dan optimisasi kueri. Tujuan utama dari penelitian eksperimental ini adalah untuk membandingkan waktu respon database sebelum dan sesudah proses tuning database. Kami memilih Sistem Informasi Rumah Sakit sebagai objek kami di mana kami mengerjakan jutaan data dan mengeksekusi beberapa kueri SQL untuk melihat perbedaan waktu respons sebelum dan sesudah penyetelan basis data. Akibatnya, untuk mengambil 1.039.852 baris data Patient Care dibutuhkan 743 detik sebelum penyetelan, dan hanya 46 detik setelah penyetelan.

Kata kunci: *RDBMS, Penyetelan Basis Data, Pengindeksan Tabel, Pengoptimalan Kueri, SQL*

Abstract

RDBMS has become a fundamental need for all companies and is expected to provide information timely, accurately, and reliably. Unfortunately, exponential growth of data usually leads to database performance issues, namely slow query execution time as one major problem. An option to enhance database performance is to perform database tuning using table indexing and query optimization. The main aim of this experimental research is to compare database response time before and after database tuning process. We chose a Hospital Information System as our object where we worked on millions of data and executed some SQL queries to see the difference of response times before and after the database tuning. As a result, to retrieve 1.039.852 rows of Patient Care data it took 743 seconds before tuning, and only 46 seconds after tuning.

Keywords : *RDBMS, Database Tuning, Table Indexing, Query Optimization, SQL.*

INTRODUCTION

In Industry 4.0, Relational Database Management System (RDBMS) has become one of the basic needs for all companies. Nowadays data is a very valuable asset, and having the access to it is absolutely necessary [1]. Obviously, a reliable RDBMS is expected to support the organization by providing information accurately and timely.

When the frequency of transactions increases and the amount of data has been exploding, we often found a database-based application performs slowly due to poor database performance. It is reflected in the increasing of response time needed when an SQL query executed. Big datasets can cause overhead to Database Management System (DBMS) and lead to database performance issues [2]. Similarly, it happened to the database of Hospital "XYZ", which is one of the issuers that has been using Enterprise Resource Planning (ERP) since five years ago. This research was conducted at the Hospital "XYZ" where a decrease of database performance was shown by the data retrieval process became slow along with the increase in daily transactions and data production.

RDBMS is a relational Database Management System in which data is stored in the form of tables [6]. To access the database contents we can use SQL statements. SQL (Structured Query Language) is a language used to access and manipulate database contents [7].

SQL statement or query must be written with a systematic technique in order to get effectiveness namely faster time response. To make sure the SQL query has been written in a proper way we can perform a query optimization. According to Elmasrti and Navathe, query optimization is an activity conducted by a query optimizer in a DBMS to select the best available strategy for executing the query [8]. In line with that, Patel wrote that query optimization is the procedure of choosing the most systematic technique to accomplish a SQL statement. The main goal of database optimization is usually common, namely to improve a numeric value, which characterizes database performance well [16].

In addition to the query optimization, to get a good database performance, the database itself must be designed and developed properly. One thing must be taken into consideration when we develop a database is the indexing technique since it generally improves the data retrieval operations. The function of index in a database is similar to the index function in a textbook. Using an appropriate index to databases is the most important aspect to optimize database operation [7]. Mentioned by Balasubramanian [10] that information retrieval can be made more efficient by using indexes to provide rapid access to database table contents.

Guzun wrote that to support efficient ad hoc queries, appropriate indexing mechanism must be in place [11]. Indexing has always been a key technique to improve performance in database systems [12]. As presented by Chopade [14] that indexing is an important concept to search data faster. Index tuning, as part of the physical database design, is the task of selecting, creating, deleting, and rebuilding index structures to reduce workload processing time [17].

In the previous study, Bajaj [1] analysed the decrease of database performance, using indexing and query optimization method, and stated that database performance can be improved by modify the SQL query and database indexing, however his study does not include experimental testing and comparison result. Rahman [4] found that slow performance issue in database based application can be solved by indexing which make SQL queries may be much faster. Chopade [14] also concluded that in the MongoDB database indexing has an important role to improve database performance. In another research related to MySQL database slow performance issue, Hidayat [5] succeeded in improving the database performance by implementing query optimization that make query process faster.

The difference between this study and previous studies is that the object of this research is the database of Hospital "XYZ" and we used table indexing and query optimization methods. The main aim of this experimental research is to compare database response time before and after database tuning process.

METHOD RESEARCH

This experimental research choose the Patient Care module of Hospital “XYZ” application as the object. This module recorded hospital daily transactions namely the procedures done by the medics and the consumption of drugs and consumables given to the patients as a treatment. Data produced in this module will be the main data source for patient invoicing process. Sample data are taken from eight tables in Patient Care module. In this experiment we used a MySQL sample database contained 32,043,471 rows of data.

Several tests are performed by running several SQL commands on the selected tables and measuring the response time to display the data from the SQL command. The tests were performed in two different database conditions, before and after tuning process.

Eight steps taken in the study were (1) extracting data from several tables of the HospitalDB database, which is the "XYZ" Hospital database; (2) data transformation; (3) load data into MySQL sample database; (4) run the SQL command and record the results of the first test. The next step is (5) Table Indexing, which is to determine and create indexes on all tables; then (6) is performing SQL Optimization by rewriting the SQL query, (7) is executing the SQL command and recording the results of the second test. The final step is (8) comparing the results of the first and second tests. Figure 1 illustrates the steps.

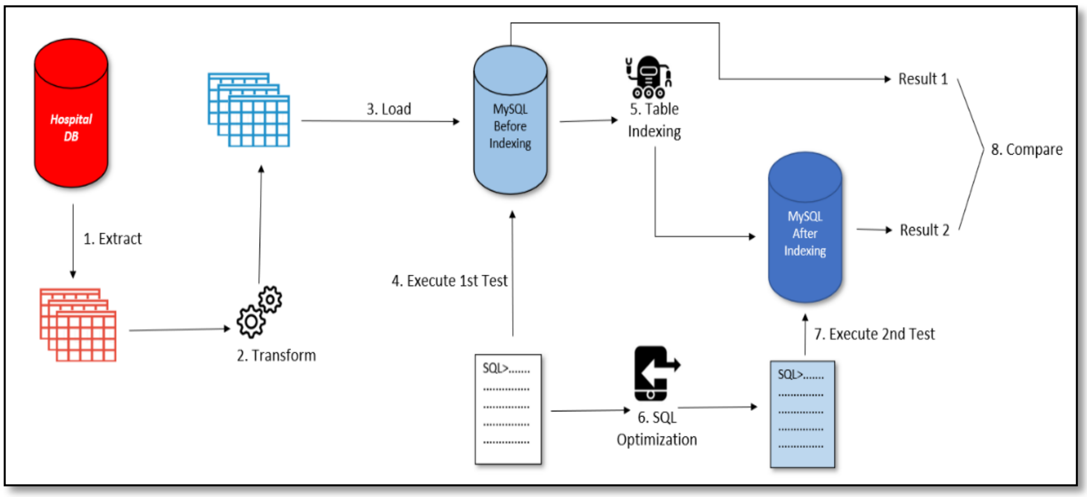


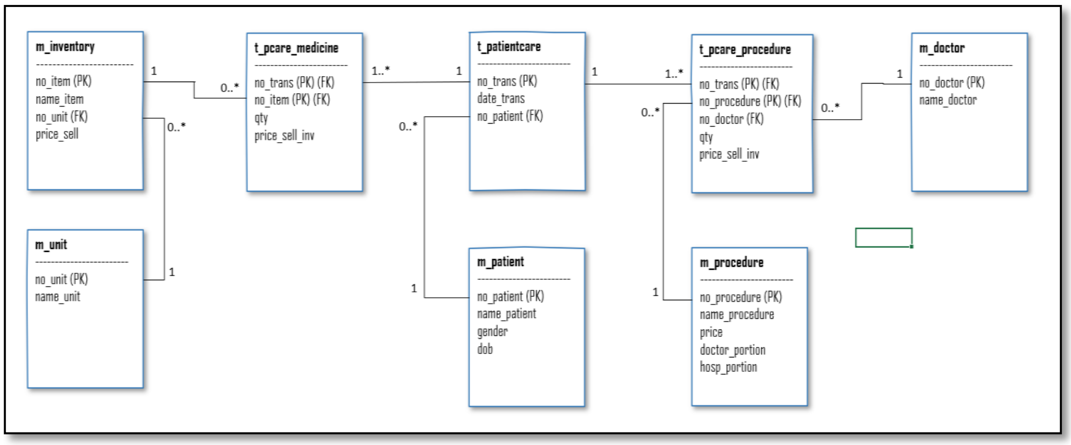
Figure 1 – Eight steps in this study

Figure 1 illustrates the steps.

EXPERIMENTAL AND COMPARISON STUDY

Hospital “XYZ” uses an ERP, where there are several modules including Inventory, Bed Management, Procurement, Consignment, Patient, Registration, Patient Care, Medical Record, Invoice, Payment, Voucher, and BPJS as well as several other modules.

The objects in this study are eight tables related to the Patient Care module which records all procedures done by the medics and all medicines given to the patients as a therapy. The tables are m_doctor, m_inventory, m_patient, m_procedure, m_unit, t_patientcare, t_medicine, and t_procedure, which are depicted in an Entity Relationship Diagram (ERD) as shown in Figure 2.



The differences between SQL queries executed before and after database tuning are as follows:

Tabel 1 – SQL queries before and after query optimization

BEFORE	***AFTER***
Query 1	
a.no_trans, a.date_trans, a.no_patient, e.name_patient, b.*, c.*, d.* t_patientcare as a, t_pcare_procedure as b, m_doctor as c, m_procedure as d, m_patient as e a.no_trans = b.no_trans and b.no_doctor = c.no_dokter and d.no_procedure = b.no_procedure and a.no_patient = e.no_patient and month (a.date_trans) in (1) and year(a.date_trans) in 2021	a.no_trans, a.date_trans, a.no_patient, e.name_patient, b.no_dokter, c.name_doctor, d.price t_patientcare as a, t_pcare_procedure as b, m_doctor as c, m_procedure as d, m_patient as e a.no_trans = b.no_trans and b.no_doctor = c.no_dokter and d.no_procedure = b.no_procedure and a.no_patient = e.no_patient and a.date_trans between ‘2021-01-01 00:00:00’ and ‘2021-01-31 23:59:59’
Query 2	
a.no_trans, a.date_trans, a.no_patient, e.name_patient, b.*, c.*, d.* t_patientcare as a, t_pcare_procedure as b, m_doctor as c, m_procedure as d, m_patient as e a.no_trans = b.no_trans and b.no_doctor = c.no_dokter and d.no_procedure = b.no_procedure and a.no_patient = e.no_patient and month (a.date_trans) in (1,2) and year(a.date_trans) in 2021	a.no_trans, a.date_trans, a.no_patient, e.name_patient, b.no_dokter, c.name_doctor, d.price t_patientcare as a, t_pcare_procedure as b, m_doctor as c, m_procedure as d, m_patient as e a.no_trans = b.no_trans and b.no_doctor = c.no_dokter and d.no_procedure = b.no_procedure and a.no_patient = e.no_patient and a.date_trans between ‘2021-01-01 00:00:00’ and ‘2021-02-28 23:59:59’
BEFORE	***AFTER***
Query 3	
a.no_trans, a.date_trans, a.no_patient, e.name_patient, b.*, c.*, d.* t_patientcare as a, t_pcare_procedure as b, m_doctor as c, m_procedure as d, m_patient as e a.no_trans = b.no_trans and b.no_doctor = c.no_dokter and d.no_procedure = b.no_procedure	a.no_trans, a.date_trans, a.no_patient, e.name_patient, b.no_dokter, c.name_doctor, d.price t_patientcare as a, t_pcare_procedure as b, m_doctor as c, m_procedure as d, m_patient as e a.no_trans = b.no_trans

and a.no_patient = e.no_patient and month (a.date_trans) in (1,2,3) and year(a.date_trans) in 2021	and b.no_doctor = c.no_dokter and d.no_procedure = b.no_procedure and a.no_patient = e.no_patient and a.date_trans between ‘2021-01-01 00:00:00’ and ‘2021-03-31 23:59:59’
Query 4	
a.no_trans, a.date_trans, a.no_patient, e.name_patient, b.*, c.*, d.* t_patientcare as a, t_pcare_procedure as b, m_doctor as c, m_procedure as d, m_patient as e a.no_trans = b.no_trans and b.no_doctor = c.no_dokter and d.no_procedure = b.no_procedure and a.no_patient = e.no_patient and month (a.date_trans) in (1,2,3,4) and year(a.date_trans) in 2021	a.no_trans, a.date_trans, a.no_patient, e.name_patient, b.no_dokter, c.name_doctor, d.price t_patientcare as a, t_pcare_procedure as b, m_doctor as c, m_procedure as d, m_patient as e a.no_trans = b.no_trans and b.no_doctor = c.no_dokter and d.no_procedure = b.no_procedure and a.no_patient = e.no_patient and a.date_trans between ‘2021-01-01 00:00:00’ and ‘2021-04-30 23:59:59’
Query 5	
a.no_trans, a.date_trans, a.no_patient, e.name_patient, b.*, c.*, d.* t_patientcare as a, t_pcare_procedure as b, m_doctor as c, m_procedure as d, m_patient as e a.no_trans = b.no_trans and b.no_doctor = c.no_dokter and d.no_procedure = b.no_procedure and a.no_patient = e.no_patient and month (a.date_trans) in (1,2,3,4,5) and year(a.date_trans) in 2021	a.no_trans, a.date_trans, a.no_patient, e.name_patient, b.no_dokter, c.name_doctor, d.price t_patientcare as a, t_pcare_procedure as b, m_doctor as c, m_procedure as d, m_patient as e a.no_trans = b.no_trans and b.no_doctor = c.no_dokter and d.no_procedure = b.no_procedure and a.no_patient = e.no_patient and a.date_trans between ‘2021-01-01 00:00:00’ and ‘2021-05-31 23:59:59’
Query 6	
a.no_trans, a.date_trans, a.no_patient, e.name_patient, b.*, c.*, d.* t_patientcare as a, t_pcare_procedure as b, m_doctor as c, m_procedure as d, m_patient as e	a.no_trans, a.date_trans, a.no_patient, e.name_patient, b.no_dokter, c.name_doctor, d.price t_patientcare as a, t_pcare_procedure as b, m_doctor as c,

a.no_trans = b.no_trans and b.no_doctor = c.no_dokter and d.no_procedure = b.no_procedure and a.no_patient = e.no_patient and month (a.date_trans) in (1,2,3,4,5,6) and year(a.date_trans) in 2021	m_procedure as d, m_patient as e a.no_trans = b.no_trans and b.no_doctor = c.no_dokter and d.no_procedure = b.no_procedure and a.no_patient = e.no_patient and a.date_trans between ‘2021-01-01 00:00:00’ and ‘2021-06-30 23:59:59’

In this study, SQL commands are used to call patient care transaction data with several variations in the time span.

In this research, the Query Optimization process is carried out by applying some changes to the SQL command. The first change removes all asterisks '*' contained in 'SELECT' since this slows down the query process. The asterisk is replaced by the required column name. In the example above, 'b.*' is replaced by 'b.no_doctor' because what is really needed in the query is only the doctors number. Then 'c.*' was changed to 'c.name_doctor' because the only column needed was the doctor's name information. And finally, 'd.*' is replaced by 'd.price' since we only need the selling price column.

The second change is in the 'WHERE CLAUSE' section. To limit the transaction time range, before optimization, the conditions 'month (a.date_trans) in (1) and year(a.date_trans) in 2021' are used. The use of this condition makes the process of calling data slower even though the 'date_trans' column has been indexed. Then optimization is done by changing this condition to "a.date_trans between '2021-01-01 00:00:00' and '2021-01-01 23:59:59'"

In this study, indexes were made on several tables so that the data retrieval process became faster. The index created is as in Table 2. Some SQL commands that are executed to create the index in this study are:

```
CREATE INDEX idx_no_patient ON t_patientcare (no_patient);
CREATE INDEX idx_no_doctor ON m_doctor (no_doctor);
CREATE INDEX idx_no_procedure ON m_procedure (no_procedure);
```

Table 2 – Creating table indexes.

Tables	***Columns***
‘t_patientcare’	*‘no_trans’, ‘no_patient’, ‘date_trans’.*
‘t_pcare_procedure’	*‘no_trans’ dan ‘no_procedure’, ‘no_procedure’, ‘no_doctor’.*
‘t_pcare_medicine’	*‘no_trans’ dan ‘no_item’, kolom ‘no_item’.*
‘m_doctor’	*‘no_doctor’.*
‘m_procedure’	*‘no_procedure’.*
‘m_patient’	*‘no_patient’.*
‘m_inventory’	*‘no_item’.*
‘m_unit’	*‘no_unit’.*

All Primary Key and Foreign Key columns are indexed. An example is the column 'no_trans' in the table 'tpatientcare'. Then the column that is read in the 'WHERE CLAUSE' condition is also indexed so that the query process becomes faster, for example, the 'date_trans' column in the 't_patientcare' table.

In the database there are three transaction tables with the following data: the transaction table 't_patientcare' has 8,118,063 rows of data with a storage usage of 871 MB; the transaction table 't_pcare_medicine' has 6,955,626 rows of data using 824 MB; and the table 't_pcare_procedure' contains 16,959,729 rows of data consuming 2.4 GB.

The test is carried out by calling the number of rows of data that vary, so we can see the response time trend of each query 177,608 rows of data were successfully called. Before optimization it needed 699 seconds while after optimization it took only 8 seconds. Next is to obtain 337,640 rows of data in the second query, before optimization it took 713 seconds and then becomes 16 seconds after optimization. The third query produces 515,442 rows of data, before optimization it took 722 seconds, after optimization it became 24 seconds.

The fourth query resulted in 689,656 rows of data, before optimization it took 729 seconds, and after optimization it became 32 seconds. The fifth query displays 867,310 rows of data with a response time of 735 seconds and then decreases to 39 seconds after optimization. The last query managed to call 1,039,852 rows of data, which before optimization it took 743 seconds, and after optimization it became 46 seconds. The test results are shown in table 3 and Figure 3 below.

Table 3 – Time response comparison between "Before Tuning" and "After Tuning".

SQL Query	**DATA ROW** (records)	**BEFORE** (seconds)	**AFTER** (seconds)	**DIFFERENCE**
Query 1	*177.608*	*699*	*8*	*99%*
Query 2	*337.640*	*713*	*16*	*98%*
Query 3	*515.442*	*722*	*24*	*97%*
Query 4	*689.656*	*729*	*32*	*96%*
Query 5	*867.310*	*735*	*39*	*95%*
Query 6	*1.039.852*	*743*	*46*	*94%*

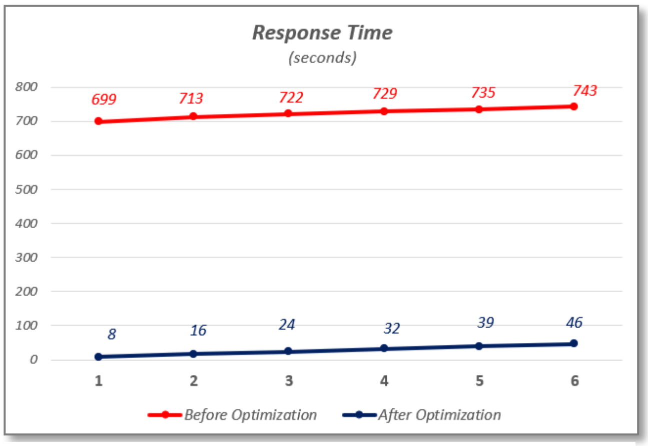


Figure 2 - Graph comparison between "Before Tuning" and "After Tuning"

CONCLUSION AND RECOMMENDATION

The test results show the comparison of the time required for the database to run SQL commands before and after database optimization is carried out. SQL queries run faster after database optimization. This supports previous studies which state that there are several ways to improve database performance, including by doing Table Indexing [7] and Query Tuning [6].

Recommendation for Hospital "XYZ" is to run database tuning on all application modules. The research that has been carried out on the Patient Care module can be used as an example to be applied to other modules that also produce large amounts of operational data such as the Inventory Management and Invoicing modules.

For further research, we suggest another database research that measures the burden and costs incurred by creating an index in the database because table indexing is estimated to require 5%-15% of additional storage media [13]. Finally, please keep in mind that query optimization must be done efficiently in terms of time and resources [15.

REFERENCES

Moura et. al, “Management and Ownership: A Data Strategy in the Industry 4.0 Context”, Conference Paper, BDIOT, 2019.

Khushairi et. al, “Database Performance Tuning Methods for Manufacturing Execution System”, World Applied Sciences Journal 30, 2014.

Bajaj, PL, “A Survey on Query Performance Optimization by Index Recommendation”, International Journal of Computer Applications (0975 – 8887) Volume 113 – No. 19, 2015.

Rahman, et. Al, “Analyze Database Optimization Techniques“, IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.8, 2010.

Hidayat et. al, “Genetic Algorithm for Relational Database Optimization in Reducing Query Execution Time”, Scientific Journal of Informatics Vol. 5, No. 1, 2018.

Bachav et. al, “Query Optimization for Databases in Cloud Environment: A Survey”, International Journal of Database Theory and Application, Vol.10, No.6 ISSN: 2005-4270 IJDTA, 2017.

Abbas et. al, “Query Performance in Database Operation”, PS-FTSM-2020-045, 2020.

Elmasri and Navathe, “Fundamental Of Database Systems Seventh Edition”, ISBN-13: 978-0-13-397077-7, Pearson, 2016.

Patel and Patel, “A Review Paper on Different Approaches for Query Optimization using Schema Object base View”, International Journal of Computer Applications (0975 – 8887) Volume 114 – No. 4, 2015.

Balasubramanian et. al, “Dynamic Integrated Database Index Management”, United States Patent No. 8489565 B2, 2013.

Guzun, “Hybrid Query Optimization For Hard-To-Compress Bit-Vectors”, The VLDB Journal DOI: 10.1007/s00778-015-0419-9, 2016.

Medina et. al, “Evaluation of indexing strategies for possibilistic queries based on indexing techniques available in traditional RDBMS”, International Journal of Intelligent Systems Vol. 31 Issue 12, 2016.

Yu et. al, “Two Birds, One Stone: A Fast, yet Lightweight, Indexing Scheme for Modern Database Systems”, Proceedings of the VLDB Endowment, Vol. 10, No. 4, 2016.

Chopade et. al, “MongoDB Indexing For Performance Improvement”, ICT System and Sustainability Proceedings of ICT4SD Vol. 1, 2019.

Ortiz et. al, “Learning State Representations for Query Optimization with Deep Reinforcement Learning”, DEEM 18, https://doi.org/10.1145/3209889.3209890, 2018.

Moszi et. al, “A Session-based Approach to Autonomous Database Tuning”, Acta Polytechnica Hungarica, Vol. 17, No. 1, 2020.

Fuentes et. al, “Database Tuning With Partial Indexes”, Conference Paper SBBD 33rd, https://www.researchgate.net/publication/336497706_Database_Tuning_with_Partial_Indexes, 2018