Query Performance in Database Operations

Abdulghafor Abbas
Faculty of Information Science and Technology,
Universiti Kebangsaan Malaysia, 43600 Bangi, Selangor, Malaysia
E-mail: whaleabbas@live.com

Kamsuriah Ahmad
Center for Software Technology and Management,
Faculty of Information Science and Technology,
Universiti Kebangsaan Malaysia, 43600 Bangi, Selangor, Malaysia
E-mail: kamsuriah@ukm.edu.my

Abstract - Database management systems have become the most important process since millions and billions of data transactions taking place every second. It comes as surprise that database optimization and tuning has become the main key research. If the database processes are not handled properly, it will lead to a slow system and it might cause a lot of errors and there is a possibility for the system to crash. Since database main task is storing and accessing the data according to the user needs through SQL operations, therefore there is a need to optimize the database operations by reducing their response times. There are many ways to optimize database operations, but among those; database tuning seems the most challenging area. There are many studies done on the improvement of database tuning approach however they are still suffer from a slow query processing time. Factors causing slow processing time normally are due to small-shared pool size and improper execution plans used during queries. This study focused on improving database indexing to overcome these two factors. A combination of clustered index, non-clustered index and bitmap is proposed as a new approach in query processing. These combinations of indexes have been tested using complex and simple queries. The results of this experiment are being compared with the result from the existing indexing approaches when using the same datasets. The result shows that these combinations are able to optimize the database systems. The proposed solution able to provide a database system that is fast in data retrieval and an improved performance percentage of 10% to 20% depending on the query, and the dataset used. Indirectly, this proposed solution enables companies' database systems to achieve its highest potential with maximum performance.

Keywords: query processing, clustered index, non-clustered index, DBMS, database optimization

I. INTRODUCTION

Database management system (DBMS) plays a crucial role in the organizations. DBMS is a program that manages the data inside the database and the structures that holds this data [1]. The stored data are accessed and managed by query language such as SQL. Optimizing the query processed involves improving the speed thus reducing system response time [2]. Systems with a good database management are vital since company relies on an effective and efficient system to operate their daily activities. Data analyst, database designers and administrators work closely to optimize the performance of the system through various strategies. From a software company's perspective, the lack of optimization process in databases can cause unreasonable costs to both clients and companies [3]. The way the database is operated should be the focused in order to improve the database performance [4]. There are many ways to optimize the database operation: such as avoiding unused tables, proper indexing usage, avoiding temporary tables and coding loops, and many others. The interest of this study is to tune the database structure and optimize the architecture rather than focusing on improving the query writing. The aim of database tuning is to maximize and to improve the system resources. Even though most systems are able to manage their resources but there is still room for improvement in terms of their efficiency by customizing the settings and configurations [5]. The configurations of query processing are based on the steps stated in the execution plans. The purpose of execution plan is to calculate the most efficient way to execute the query. However, to improve data retrieval process during query execution is to improve the indexing method used in the query [6][7]. Therefore the focus of this study is to improve the indexing method in the query execution plan. Database users and administrators shall manually examine and tune this plan to get better performance.

During query execution, the type of indexing method used is very important to improve the database operations. The advantage of index is its ability to search data without searching every single row in a database table during query [5]. Therefore proper choice of indexing technique used in the query execution plan is very important. There are a number of indexing techniques exist, such as cluster index, non-cluster index, bitmap index, and others. There are researchers who embed these techniques in their query improvement process. However, the existing applications of these indexes are still faced problems in terms of the time taken to process the query [8]. This study explores whether the combination of cluster index, non-cluster index and bitmap index in one process able to improve the time processing during queries. Therefore, to explore the possibility of these combinations in the query execution plan, and to achieve the aim of the study, this paper is organized as follows. In the following section is the discussion of the index management in database operations. Section III discusses different type of indexes exists in the literature, Section IV reports the analysis of SQL query processing in database systems. Section V explains on the development of the proposed method to improve query processing and in Section VI discusses on the evaluation of the proposed approach through an experiment. The last

section states the conclusion of the paper and the future works.

II. Index Management In Database

Indexing is a significant data structure in database. The purpose of having an index in a database is to sort records into multiple fields. When index is created on a field table, another data structure is created that holds a pointer to the record and the value of the field [5]. As an analogy, imagine a user tries to search a word in a textbook using a search function. Based on user request, the system will go directly to the desired word in the textbook. 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 operations. However, by having an index on each field table, additional space will be created on the disk since the indices are stored together in a table. The database file can reach the size limits quickly if within the same table there are many field are being indexed [6].

Based on its important to improve the query response time, there are many types of indexes used. There are seven index types which are among the most popular used for query improvement, such as: clustered index [9], non-clustered index [5], hash index [11], bitmap and bitmap-join index [12], partitioned index [13], domain index [14], and function-based index [6]. Although these indexes boost the performance of the database system, however, they have some limitations. Thus, the motivation of this project is to propose an improved indexing approach to help in reducing the query processing time for the database systems. In the following section, is the discussion on each type of index.

III. Types of Indexes

There are several type of indexes used in the existing DBMS. Database indexes can be categorized into the following types.

A. *Clustered Index*

A clustered index is an index that alters the way records is organized in the table, as the result the data is being stored in-order. Because of this organization one clustered index is assigned to a table. Since clustered index used B-tree in its structured, the leaf nodes stored the actual data while the table is sorted based on its key values. Since the value of the leaf nodes is unique, only one clustered index is assigned in the database table. Because of its structure, clustered index is able to speed up the retrieval process. But it only applicable if the data is retrieved sequentially in the same order or reverse order of the clustered index or the data is selected based on its range of items [9]. There are studies done in improving query processing using this type of index such as [5][15][16]. This index scale well because it used B-tree structure in its processing.

B. *Non-clustered Index*

Like clustered index, non-clustered index also used b-tree as its structure. However the different is non-clustered index separate the data and the index in two different places. The leaf nodes stored the index rows, which points to the location of the data. With this structure, it is possible to have multiple non-clustered in a table. Studies done by [5][16][17] embed this index in their works. This index is very helpful in retrieving data quickly from database table.

C. *Hash Index*

Hash index is like a collection of buckets that organized in an array. Each bucket has a pointer that points to a data row. Studies done by [11][18][19] improved the query processing using this type of index. However, this index seems complex to be implemented by junior DBA.

D. *Bitmap and Bitmap-Join Index*

The bitmap indexes utilized in the queries is applied to low or medium cardinality columns. Studies done by [12] [20][21] explore the features provided by this index. However, this type of index is good for low cardinality columns (example gender field).

E. *Partitioned Index*

A partitioned index in Oracle 11g is basically an index that breaks into numerous pieces. Studies done by [13] [22] [23] explore the suitability of this index in their studies. It allows quick and ease of access and efficient for smaller data units.

F. *Domain Index*

This index is suitable for a particular domain, for example spatial or picture handling or video file. Studies done by [13] [24] [25] explore the suitability of this index in their studies. It can only be used in some tables and requires a lot of maintenance and various monitor size.

G. *Function-based Index*

A function-based index calculates the result of a function that involves one or more columns and stores that result in the index. Studies done by [6] [26] used this index to query the web. It is a useful index and can help the DBA in creating a fast database.

As discussed, there are advantages and disadvantages of each index on database operations. The existing DBMS normally used single index during query processing,

however this study tries to explore the possibility of combining multiples indexes in order to improve the query processing. The aim of this study is to improve the time taken for query processing; therefore three steps are outline as the methodology in this study. The details for each step discussed in the following section.

i. Analyzing and understanding query processing steps
ii. Proposing an approach to improve query processing
iii. Evaluating the proposed approach

IV. ANALYZING QUERY PROCESSING

SQL (Structured Query Language) is a language used to access and manipulate database contents. The performance of the query is measured based on the time taken to process the query. When a user post a SQL statement to the server the required processing time for the server consist of the following:

i. Application processing time: the time taken for the application to process data from the previous response, before sending the next request.
ii. SQL processing time: the time taken to process the request and send back the results.

Since this study focuses on improving SQL processing time, therefore there is a need to analyze the steps taken in query processing. SQL is different from other programming languages on the way the codes are being executed. Most programming languages execute statement from top to bottom. On the other hand, SQL defines the order in which the clauses of a query are executed. Figure 1 explains the order of SQL process.

```
(8)  SELECT (9) DISTINCT (11) <TOP_specification> <select_list>
(1)  FROM <left_table>
(3)    <join_type> JOIN <right_table>
(2)      ON <join_condition>
(4)  WHERE <where_condition>
(5)  GROUP BY <group_by_list>
(6)  WITH {CUBE | ROLLUP}
(7)  HAVING <having_condition>
(10) ORDER BY <order_by_list>
```

Fig. 1 The order of SQL clauses [10]

As stated in the figure, the first clause that is processed is the FROM clause, while the SELECT clause, which appears first, is executed almost at the end of the query processing. Normally, the database systems through query execution plan will follow this order when process the query [27]. The function of execution plan is to calculate the most efficient way to execute the query. Query execution plan is a sequence of steps used to access data in database. Database users and administrators shall manually examine and tune this plan to get better performance. This plan will guide the system the way the SQL should be executed to improve the query processing time. There are a number of steps or stages the database systems do before the execution of SQL statement. Figure 2 illustrates the stages of SQL processing.

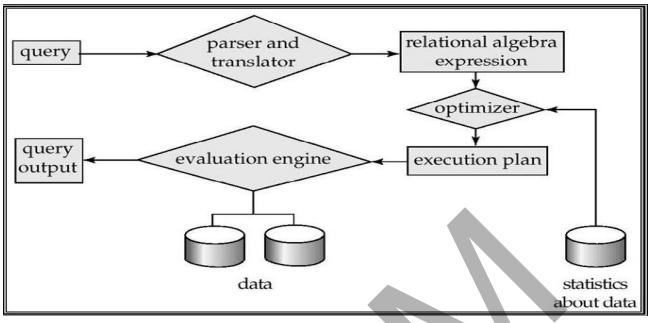

Fig. 2 The stages in Query Processing [28]

When a SQL query is constructed by the user, this query will be sent to the database system by the query processor where the system parses and executes the SQL query. An execution plan will be constructed where this plan will identify the most efficient way to execute the query. Three basic steps in query processing: i) Parsing and translation, ii) Optimization, and iii) Evaluation [29].

i) Parsing and Translating SQL Query

At first, the query statement will undergo a process called parsing. In this process, SQL statement will be divided into a different data structures. This query is then translated into relational algebra [30]. A parser will check the syntax, semantics of the query and the shared pool. The shared pool is like a buffer for SQL statements. It is used to store the SQL statements so that the identical SQL statements do not have to be parsed each time they're executed. This study also looks into the possibility to improve the used of shared pool during query since the size of the pool shall be adjusted to accommodate the query processing.

ii) Optimizating SQL Query

During query, the users will state what data they are looking for and the system will search the data in the database. The searching algorithm will be identified by the DBMS. For a given query there are many possible way to search for the data or execution plans. Normally the system will choose the plan with the lowest cost. The cost is measured using statistical analysis from the database catalog.

iii) Evaluating SQL Query

The query evaluation engine takes a query evaluation plan, executes the plan and returns the answer to that query.

Upon analyzing these three steps in query processing, this study found there are two factors to improve the query processing, which are: i) the shared pool size and ii) the execution plan. These factors will be considered in the

design of the proposed approach to improve query processing. Since execution plan indicates and guides the way the query should be executed, therefore in order to improve query processing time, this study will improve the execution plan by improving the used of shared pool size and the indexing method. Below is the discussion on the shared pool size and the execution plan in query processing before these two factors are used and embedded in the proposed approach.

A. *Shared Pool Size*

Shared Pool contains parsed SQL statements and execution plans. With nonstop utilization of database operations, after a timeframe the shared pool will be divided. The pool is loaded with new and old parsed SQL and execution plans. This will likewise prompt bigger packages being matured out with new sections going into shared pool. Henceforth access to such bigger packages will set aside some effort to parse and make execution plan. This may cause performance issues. To overcome this situation, the user shall use a parameter called SHARED_POOL_RESERVED_SIZE. This will provide some extra space on the defined shared_pool_size. By changing the shared pool size and reserve some extra space in the pool, more execution plans can be stored. However, the user needs to define this parameter manually during query processing.

B. *Execution Plans*

Execution plans are often reviewed by the database admins when reviewing query performance [27]. The execution plan indicates what tuning should be done to improve the query processing. This plan demonstrates the query execution; identify the most expensive query, and the performance of the index used during the query processing. The operator that contributes to low query performance will also be identified. The scan operators that are used to scan the whole table are able to identify a missing index, the index that is wrongly utilized, or the query contains no filtering condition. Since database users and administrators shall manually examine and tune the execution plan to get better performance, therefore this study will tune the execution plan to get better query performance.

Therefore these two factors (shared pool size and the execution plan) will be enhanced in the proposed solution in order to improve the performance of the query. The next section will discuss the proposed improvement approach.

V. THE PROPOSED INDEXING APPROACH

The aim of this study is to reduce query processing time by improving the execution plans and the shared pool size. This improvement will be embeded in the query processing. In Oracle query processing, clustered and non-clustered index are used during the process but in this study, three combination of indexes are used, which are clustered, non-clustered and bitmap index. These three combinations of indexes are used in the query execution plan and this plan is re-structured to include the seven steps as stated in Table 1 below.

TABLE 1 The proposed indexing steps

Step 1 :	Deletes all current indexes.
Step 2 :	find a suitable column to apply clustered index
Step 3 :	find a column that has low cardinality values.
Step 4 :	find a common column to apply non-clustered index
Step 5 :	create clustered index on the column in step 2.
Step 6 :	create bitmap index on the column that is mentioned in step 3.
Step 7 :	create a non-clustered index on a column that is critical

The combination of three indexing techniques (clustered, non-clustered and bitmap index) hopefully will improve the query execution plan thus reduce the processing time. The explanations of these seven steps are as follows:

Step one is to delete all the current indexes in the datasets, the reason for doing this is to fix performance problem that caused by the existing index and to avoid the new index overlap with the existing ones. The second step is to find a column where a clustered index could be applied. The clustered index could not be applied on any column; it should be applied only on columns that have unique values and does not contain null values. If this column did not exist then a new column will be created, this procedure is necessary to help restructure the database physically. Step three is to search a column with low cardinality values. Low cardinality means column with a small pool of data, such as gender column (M/F), payroll types or others. The reason to search this kind of column is to apply bitmap index in this column. Bitmap index could not be applied on top of column with high cardinality since this will downgrade the performance of the database. Step four is to find a common column where many search operations are being applied to this common column. The reason for doing this is to apply non-clustered index into this column. A column that has both clustered and non-clustered index applied to it will help reduce the query processing time. Step five is to create clustered index in the column found in step two. While step six is to create bitmap index into a column that found in step three. Step seven is to create a non-clustered index into a column that is critical. These seven steps will be embedded in the query execution plan.

VI. EVALUATING THE PROPOSED APPROACH

The proposed approach will be developed using Oracle platform (SQL DEVELOPER). The seven steps as discussed

in the previous section will be embedded in the query execution plan and the size of the shared pool will be re-size to reserve some extra space in the pool. To evaluate the reliability of the proposed approach, three tests are being conducted. The intention of these three tests is to analyze the improvement of query execution when the three indexes are applied. The parameter for evaluation is the number of rows accessed by the database systems to produce the output, where this parameter is called cost. The least number of table rows being accessed by the database system or low cost indicates the index for that particular approach is better. The first test is to analyze the query processing time when no index is being applied in the execution plan. This test is to evaluate the first step in the approach. Payroll dataset [32] which consists of 1203 rows of records with 11 columns is used in this test. A simple query as stated below is used:

Select first_name,salary,job_id from employees
Where job_id= 'SA_MAN';

Query execution plan as stated in Fig. 3 is executed. As stated in the figure the number of cost is three, which indicates that the number of rows being accessed by the systems to produce the result is three rows.

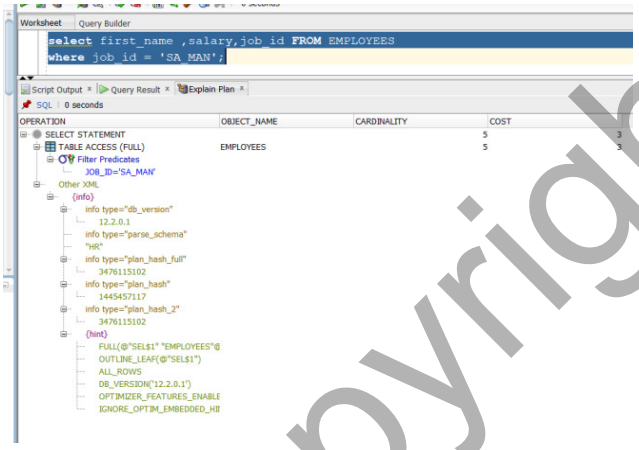

Fig. 3 Execution plan without index

The second test is to analyze the query processing time when a bitmap index is being applied in the execution plan while the same query is being used. Fig. 4 states the result of this execution.

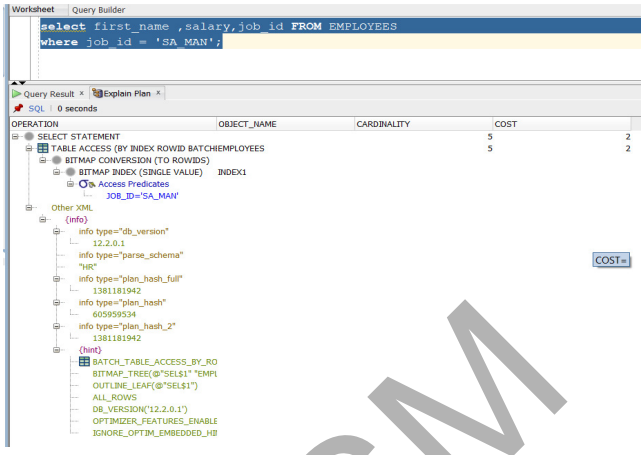

Fig. 4. Execution plan with bitmap index

As stated in the Fig.4, the cost of this query is reduced to two. This indicates that when bitmap index is applied in the table field this will improve the searching process. Third test is then conducted, which is to analyze the query processing time when the combination of bitmap index and the clustered index is being applied in the execution plan while using the same query.

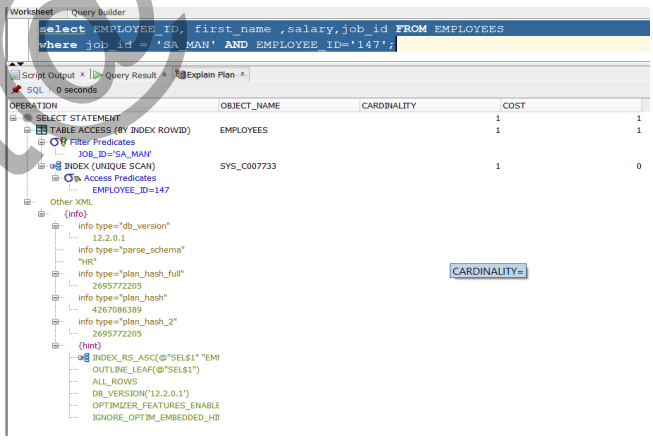

Fig. 5. Execution plan with the combination of bitmap and clustered index

As indicated in the Fig.5 the cost of the query is reduced to one. This indicates that when two indexes are used during query, it will reduce the cost and hence reduce the query processing times. And it is believed that if large datasets are used, or the combinations of three indexes are used then the result will be improved as well.

The next experiment is to compare the performance of the proposed approach with the existing approaches. Three existing approaches are used in this experiment [5][12][17], each of them used single index in query processing. These existing approaches will be developed based on the type of index used in each approach. Two datasets will be used for this experiment which is payroll [32] and Human Resource [33]. The existing approaches and the proposed approach

will be executed using two queries, one for each dataset. The output produced by all the approaches will be compared to see their significance. Fig. 6 is the query used in this experiment.

```
SELECT
  e.employee_id AS "Employee #"
  , e.first_name || ' ' || e.last_name AS "Name"
  , e.email AS "Email"
  , e.phone_number AS "Phone"
  , TO_CHAR(e.hire_date, 'MM/DD/YYYY') AS "Hire Date"
  , TO_CHAR(e.salary, 'L99G999D99', 'NLS_NUMERIC_CHARACTERS = ''.,'' NLS_CURRENCY = ''$''') AS "Salary"
  , e.commission_pct AS "Comission %"
  , 'works as ' || j.job_title || ' in ' || d.department_name || ' department (manager: '
    || dm.first_name || ' ' || dm.last_name || ') and immediate supervisor: ' || m.first_name || ' ' || m.last_name AS "Current Job"
  , TO_CHAR(j.min_salary, 'L99G999D99', 'NLS_NUMERIC_CHARACTERS = ''.,'' NLS_CURRENCY = ''$''') || ' - ' ||
    TO_CHAR(j.max_salary, 'L99G999D99', 'NLS_NUMERIC_CHARACTERS = ''.,'' NLS_CURRENCY = ''$''') AS "Current Salary"
  , l.street_address || ', ' || l.postal_code || ', ' || l.city || ', ' || l.state_province || ', '
    || c.country_name || ' (' || r.region_name || ')' AS "Location"
  , jh.job_id AS "History Job ID"
  , 'worked from ' || TO_CHAR(jh.start_date, 'MM/DD/YYYY') || ' to ' || TO_CHAR(jh.end_date, 'MM/DD/YYYY') ||
    ' as ' || jj.job_title || ' in ' || dd.department_name || ' department' AS "History Job Title"
FROM employees e
-- to get title of current job_id
  JOIN jobs j
    ON e.job_id = j.job_id
-- to get name of current manager_id
  LEFT JOIN employees m
    ON e.manager_id = m.employee_id
-- to get name of current department_id
  LEFT JOIN departments d
    ON d.department_id = e.department_id
-- to get name of manager of current department
-- (not equal to current manager and can be equal to the employee itself)
  LEFT JOIN employees dm
    ON d.manager_id = dm.employee_id
-- to get name of location
  LEFT JOIN locations l
    ON d.location_id = l.location_id
  LEFT JOIN countries c
    ON l.country_id = c.country_id
  LEFT JOIN regions r
    ON c.region_id = r.region_id
-- to get job history of employee
  LEFT JOIN job_history jh
    ON e.employee_id = jh.employee_id
-- to get title of job history job_id
  LEFT JOIN jobs jj
    ON jj.job_id = jh.job_id
-- to get namee of department from job history
  LEFT JOIN departments dd
    ON dd.department_id = jh.department_id
ORDER BY e.employee_id;
```

Fig. 6. Complex query used to access HR datasets

The query used on payroll dataset is as follows:

```
SELECT A.employee_name AS empName1,
B.employee_name AS empName2, A.City
FROM payroll A, payroll B
WHERE A.employee_name <> B.employee_name
AND A.City = B.City
ORDER BY A.City;
```

After applying these two queries on each of the approach the results are indicated as below:

TABLE 2 The results using two datasets

Reference	Technique used	Query Cost using HR dataset	Query Cost using Payroll dataset
[5]	clustered index	270	94
[12]	bitmap index	273	94
[17]	non-clustered index	272	94
the proposed approach	combination of clustered index, non-clustered index and bitmap index	267	87

As stated in Table 2 the results produced by the proposed solution are the best and the cheapest cost among all other approaches when the two queries are applied. This proves the efficiency of the proposed approach with an improvement percentage of 20%. The existing approaches that used in this experiment only apply single index. The authors of [5] used clustered index only in the approach. Even though this approach is able to produce a good result when tested using the two datasets but the results are not the best, hence the approach needs some improvement. The authors of [12] used bitmap index in their approach and able to produce a good result. However it increases the time taken to process the query, since the number of cost to access the table is the highest among all when using HR dataset. Non-clustered index is used by the author of [17] in his work. This approach is tested using both datasets in this experiment. As the result shown in Table 3, however this approach failed to produce the best result. The data in the table is not sorted properly in the database hence the number of search has increased. However, this study proved that if multiple indexes are applied in the query processing it is able to provide a better performance by reducing the time taken during query.

VII. CONCLUSION

This study shows that the combinations of indexing approaches such as clustered, non-clustered and bitmap index are able to achieve better performance in terms of reducing query processing time. The improved indexing approach shall be implemented on any dataset. For further improvements in the experiment, a different platform shall be chosen other than SQL developer for example: MySQL or DB2. This will open more indexing options that can be supported through SQL code. Other exploration exercise in improving the query processing time is to look on the possibility of combining partitioned indexes with non-clustered index. This work is very important to provide a better approach in improving query processing time.

ACKNOWLEDGMENT

This study is supported and funded by the Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia through the university grant (GGP-2019-024).

REFERENCES

[1] C.Coronel and S. Morris, Database Systems: Design, Implementation, and Management, 13th Edition, Cengage Learning, 2018.

[2] H.Albadri and R.Sulaiman, “A Classification Method For Identifying Confidential Data To Enhance Efficiency Of Query Processing Over Cloud”, Journal of Theoretical And Applied Information Technology, vol.93. no.2, 2016, pp. 215-219

[3] A.Boicea, F.Rădulescu, C.Truică and L.Urse, "Improving Query Performance in Distributed Database", Journal of Control Engineering and Applied Informatics 18(2), 2016, pp. 57-64

[4] Z.Ali Raheem, A.H Mohd Aman, S.Islam, A.H.Abdalla and A.Razzaque, "Performance Analysis for Cloud Query Encryption", International Conference on Sustainable Technologies for Industry 4.0 (STI), 2019, pp. 24-25.

[5] C. Cioloca and M. Georgescu, M. "Increasing Database Performance using Indexes", Database Systems Journal, vol. 2(2), 2011, pp. 13-22.

[6] M.K Gupta, "Comparative Study of Indexing Techniques In DBMS", Conference: Business Intelligence and Data Warehousing, p. 6, 2015

[7] Thamer Salah and Sabrina Tiun, Focused Crawling Of Online Business Web Pages Using Latent Semantic Indexing Approach, Journal of Engineering and Applied Sciences, vol. 11, no. 15, 2016, pp.9229-9234.

[8] J.M. Medina, C.D Barranco and O.Pons, "Indexing techniques to improve the performance of necessity-based fuzzy queries using classical indexing of RDBMS", Fuzzy Sets and Systems, Vol.351, 2018,vpp. 90–107.

[9] R.T. Hashim, "A Comparative Study of Indexing using Oracle and MS-SQL Server for Relational Database Management Systems", International Journal of Computer Science and Mobile Computing, Vol.7 Issue.12, 2018, pp. 341-350

[10] J. Mor, I.Kashyap and R.K. Rathy, "Analysis of Query Optimization Techniques in Databases", International Journal of Computer Applications 47(15), 2012, pp.6-12.

[11] C.Y.Chiu and S. Markchit, "Effective and efficient indexing in cross-modal hashing-based datasets", Signal Processing: Image Communication, Vol. 80, 2019.

[12] L. Toumi, A.MoussaouI and A.Ugur, "A linear programming approach for bitmap join indexes selection in data warehouses", 6th International Conference on Ambient Systems, Networks and Technologies, 2015

[13] N. Khushairi, N.A. Emran, and M.M.M Yusof, "Database Performance Tuning Methods for Manufacturing Execution System", World Applied Sciences Journal 30(30), 2014.

[14] P. H. Oliveira *et al*., "Employing Domain Indexes to Efficiently Query Medical Data From Multiple Repositories," in *IEEE Journal of Biomedical and Health Informatics*, vol. 23, no. 6, 2019, pp. 2220-2229.

[15] Oktavia, T. and Sujarwo, S. "Evaluation of Sub Query Performance in SQL Server", edited by Gaol, F.L.EPJ Web of Conferences, Vol. 68, 2014, pp. 33.

[16] C. Qi, "On index-based query in SQL Server database," *2016 35th Chinese Control Conference (CCC)*, Proceedings of the 35th Chinese Control Conference. 2016, pp. 9519-9523.

[17] K Sirohi, A. "Optimization of Dynamically Generated SQL Queries for Tiny-Huge, Huge-Tiny Problem", International Journal of Database Management Systems, Vol. 5 No. 1, 2013, pp. 53–68.

[18] D. Hao and L. Sun, "EIHJoin: An hash join with building index in bucket in column store data warehouse," IET International Conference on Smart and Sustainable City 2013 (ICSSC 2013), 2013, pp. 268-271

[19] F. S. Patel and D. Kasat, "Hashing based indexing techniques for content based image retrieval: A survey", International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), 2017, pp. 279-283.

[20] E.A.A. Abdelouarit, M. El Merouanib and A. Medouri, "The bitmap index advantages on the data warehouses", American Academic & Scholarly Research Journal, Vol. 6, No. 4, 2014, pp. 376-382

[21] F. Chen, and M. Li, "Efficient Algorithm Based on Itemset-lattice and Bitmap Index for Finding Frequent Itemsets", Systems Engineering - Theory & Practice, Vol. 28 No. 2, 2008, pp. 26–34.

[22] E. Wu and S. Madden, "Partitioning techniques for fine-grained indexing," 2011 IEEE 27th International Conference on Data Engineering, Hannover, 2011, pp. 1127-1138.

[23]K. Saleh Maabreh, "Optimizing Database Query Performance Using Table Partitioning Techniques," 2018 International Arab Conference on Information Technology (ACIT), 2018, pp. 1-4

[24] T. Truong and T. Risch, "Transparent inclusion, utilization, and validation of main-memory domain indexes", 27th International Conference on Scientific and Statistical Database Management (SSDBM), 2015.

[25] R. Shandilya, S. Sharma and S. Qamar, "A Domain Specific Indexing Technique for Hidden Web Documents", Communications in Information Science and Management Engineering, Vol.2 No.2, 2012 pp.37-41

[26] H.Lee and W. Lee, "Query Optimization for Web BBS by Analytic Function and Function-Based Index in Oracle DBMS," Sixth International Conference on Advanced Language Processing and Web Information Technology (ALPIT 2007), 2007, pp. 606-611.

[27] S.M.V. Jadhav, "An analysis of execution plans in query optimization", International Conference on Communication, Information & Computing Technology, 2012.

[28] Database SQL Tuning Guide, https://docs.oracle.com/database/121/TGSQL.

[29] G. Fritchey, "SQL Query Performance Tuning", in book: SQL Server Query Performance Tuning, 2014

[30] V.Bhagat and A. Gopal, "Roll of Relational Algebra and Query Optimizer in Different Types of DBMS", International Journal of Computer Science and Mobile Computing Vol. 5, Issue. 3, 2016, pp. 6 – 16

[31] M.A. Mohd Yunus, M. Z. Brohan, N. Mohd Nawi, E. S. Mat Surin, N.A. Md Najib and C.W.Liang, "Review of SQL Injection: Problems and Prevention", International Journal on Informatics Visualization Vol 2, No 3 – 2, 2018, pp.215-219

[32] https://opendata.socrata.com/Government/Payroll-Report-End-Date-2-20-2019/2dps-ayzy (accessed 15 November 2019).

[33] https://github.com/Oracle/db-sample-schemas (accessed 16 November 2019).