Penerbit UTHM

JSCDM

Journal homepage: http://penerbit.uthm.edu.my/ojs/index.php/jscdm

Journal of Soft Computing and Data Mining

A Case Study on B-Tree Database Indexing Technique

Ammar Aminuddin[1], Mohd Zainuri Saringat[1*], Salama A. Mostofa[1], Aida Mustapha[1], Mustafa Hamid Hassan[1]

[1]Faculty of Computer Science and Information Technology,
Universiti Tun Hussein Onn Malaysia, 86400 Johor, MALAYSIA

*Corresponding Author

DOI: https://doi.org/10.30880/jscdm.2020.01.01.004
Received 20th December 2020; Accepted 10th February 2020; Available online 1st March 2020

Abstract: In the last three decades, many organizations of various natures are used relational databases like health, education, businesses and in several applications. Human started collection information before computer system was designed. Presently, the data that collected by human in computer system named database system. The larger database the more number of joins will occur which ultimately leads to the increase in query retrieval time. This research is to implement a b-tree indexing technique on PostgreSQL database. The main objective of this paper is to implement and analyze the application of b-tree in database indexing technique. This case study may be useful for future research in developing new indexing techniques which will added new techniques together with existing techniques as to increase performance of the data query.

Keywords: Indexing database, B-Tree, PostgreSQL, performance analysis

1. Introduction

A good database has a unique identifier, known as the primary key. Typically, primary key is used for a set of personnel records such as personal identification or might be the social security number. However, designing a good database is still a big issue especially in designing optimum database because it is very hard to do consistency checking between system design and database design to fulfil user needs [1]. Criteria such as consistency, redundancy and performance always lead to a good database. As the data size grows bigger, the time taken to retrieve and search query are also increasing. There are several techniques in shortening the time taken to retrieve a query such as indexing techniques. To increase the performance of the model, it needs indexing techniques to increase the performance of queries and to retrieve the results fast [2].

In a nutshell, indexing is a significant technique for managing larger databases. If the right index structures are built on columns, the performance of queries, especially ad hoc queries will be greatly enhanced [3]. The most well-known indexing technique is the tree index. Trees are most commonly used to manage and organize large databases which support record insertion, deletion, and key range searches. B-Trees are the most widely used indexing method for large disk-based databases and for implementing file systems.

The rest of the paper is organized as follows. Section 2 introduces the theoretical foundation of B-Tree. Section 3 presents the related work. Section 4, the methodology will explain on how to conduct the study. Then, Section 5 the implementation and discussion for B-Tree indexing. Finally, Section 6 concludes the paper with future works.

*Corresponding author: zainuri@uthm.edu.my

2. Basic Concepts of B-Tree

2.1 B-Tree

Invented about 40 years ago and called ubiquitous less than 10 years later, B-Tree indexes have been used in a wide variety of computing systems from handheld devices to mainframes and server farms. Over the years, many techniques have been added to the basic design in order to improve efficiency or to add functionality. Examples include separation of updates to structure or contents, utility operations such as non-logged yet transactional index creation, and robust query processing such as graceful degradation during index-to-index navigation. This section will discuss the B-Tree technique which is commonly default index for most relational database systems. Imagine a B-Tree which has at least two nodes. The topmost level of the index is called the root. The lowest level is called the leaf node. All other levels in between are called branches. Both the root and branch contain entries that point to the next level in the index. Leaf nodes consisting of the index key and pointers pointing to the physical location example is row ids in which the corresponding records are stored. [3]. B-Tree is predominantly a single-dimensional indexing method. It is a multilevel hierarchical indexing method for data access from peripheral data stores with low operations cost. B-Tree has its genesis in graph theory. The Binary Search tree is the first well-defined method proposed in the literature for faster searching. In the late 1960s an intense competition was prevailing among the computer manufactures and independent research group to invent general-purpose data access methods. This resulted in the invention of B-Tree [4], an indexing mechanism with relatively low cost for insertion, deletion and updating. The implications of various treatments on diverse fine distinctions of the B-Tree are documented in the literature [5].

Instead of only fetching and comparing a single index key per search step like in binary search, B-Tree search compares and fetches blocks of multiple index keys per search step with the motivation of reducing hard disk seeks. The idea of grouping multiple items into nodes was originally used to reduce the number of hard disk seeks per index lookup in traditional database systems. It can, however, be used in main memory databases to reduce the number of page misses for an index lookup as well.

The example of a research process where all the process is shown in Figure 1, shows an example of a binary tree algorithm, where each of the nodes is been divided into branches.

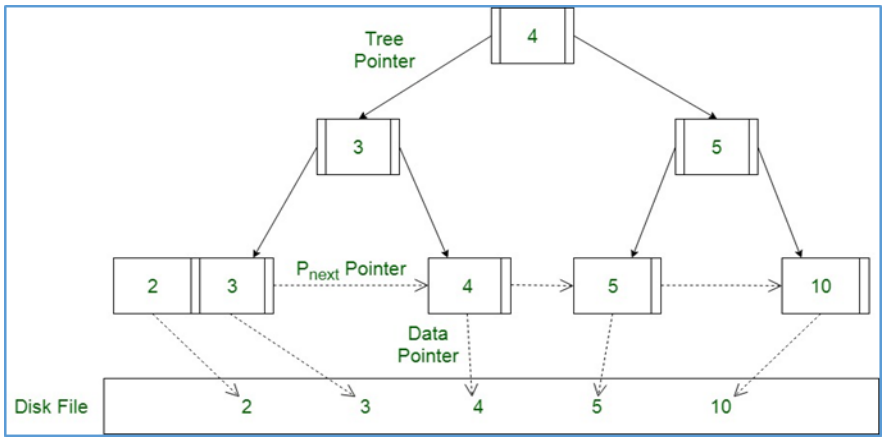

Fig. 1- Binary Tree Algorithm [2]

2.2 Variants of B-Tree

Variations of B-Trees are abounding. Some variations are due to the suggestions on implementation alternatives with respect to underflow and overflow conditions [5]. Other variations of B-Trees have concentrated on improvements in the secondary costs B*-tree [6] is a B-Tree in which each node is at least two-thirds full, instead of just half full.

In a B+-tree [6-8], all keys reside in the leaves. The upper levels, which are organized as a B-Tree, consist only of an index, a roadmap to enable the rapid location of the index and key parts. Virtual B-Tree uses a least recently used mechanism for B-Trees even when not using paging hardware [6]. When keys consist of a string of characters, an alternate method was suggested, namely the Prefix BT-tree [20]. This tree used the prefixes of the actual keys as separators in the index, to improve the tree components. Compression of keys and pointers based on displacement is also possible [2] B-Tree can also accommodate variable-length records.

The Binun. B-Tree makes B-Trees suitable for a one-level store [9]. Essentially, a Binary B-Tree is a B-Tree of order I; each node has 1 or 2 keys and 2 or 3 pointers. An extension of the Binary B-Tree that allows for both left and right links to point to sibling nodes, exhibits symmetry lacking in the Binary B-Tree. Hence, the name St.mi7tetric Birney B-Tree has been applied to such a data structure [10]. The Symmetric Binary B-Trees contain the well-known class of AVL trees as a subclass [11].

A 2-3 tree is a B-Tree of order and each node in it has 2 or 3 children [12]. The major intricacy of 2-3 is its sensitivity to the distributions of data. Its performance varied with the distributions of data. Higher orders of B-Tree skewed probability distributions of data and balancing factors also limit the B-Tree performance.

2.3 Structured Query Language (SQL)

SQL (Chamberlin & Boyce 1974) is a standard language for defining and manipulating the data in a relational database. In the relational model, the database is viewed as a set of relations; relationships between sets are denoted by relation values. The data can be retrieved by specifying a result table derived from one or more base relation. SQL statements are executed by a database manager which then transforms the specification of a result table into a sequence of internal operations that optimize the data retrieval. All executable SQL statements should be prepared before their execution. The result of preparation is the executable or operational statement. See figure 2. However, there are some issues in SQL which are taken for consideration in this work. They are listed below.

A. The SQL uses the crisp logic in the query process that causes crisp selection. It means that the record would not have been selected even if it is extremely close to the intent of the query criterion.
B. The SQL face another issue viz. „nested query" when it is subject to multi-query integration to satisfy the user's needs. The nested queries in standard SQL suffer from heavy performance penalty during execution since it contains complex query structure.
C. SQL is a powerful mechanism for advanced searchers and experienced developers. But the majority of users are not familiar with all its syntax which restricts the full utilization of SQL capability.

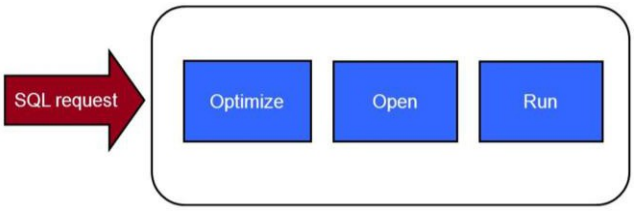

Fig. 2 - SQL process

In this article SQL Server vs PostgreSQL, we have seen Both SQL Server vs PostgreSQL are database management tools. They help in managing all data properly and efficiently. But when it comes to different features PostgreSQL is always at the upper hand. It is an advanced version of SQL and hence provides many additional features. All these features are for free, unlike SQL server. Also, it is cross-platform and can be used with any operating system.

Table 1 - SQL Server vs PostgreSQL

Comparison Item	SQL Server	PostgreSQL
Basic Difference	SQL Server is a database management system which is mainly used for e-commerce and providing different data warehousing solutions.	PostgreSQL is an advanced version of SQL which provides support to different functions of SQL like foreign keys, subqueries, triggers, and different user-defined types and functions.
Updateable Views	Views can be updatable even if 2 table views are updated. If the tables have different keys and the update statement does not involve more than one table, then it will be updated automatically. The user can also make use of triggers to update complex views.	Views in PostgreSQL can be updated but not automatically unlike SQL server. The user must write rules against different views to update them. Also, complex views can be easily created.
Computed Columns	SQL Server does provide computed columns but views are preferred over computed columns. Computed columns have very limited use as they are not capable of holding different roll-ups.	PostgreSQL does not provide computed columns. PostgreSQL, on the other hand, has functional indexes which work just like a view.
Replication	SQL server can replicate all sorts of data. This can be log shipping, mirroring, snapshot, and transactional and merge etc. and can even have	Replication in PostgreSQL is in the form of reports and is supposed to be least polished of the bunch. Although there are different third-party options to

	non-SQL Server windows-based subscribers.	choose from the ones that are free and not free. PostgreSQL 8.4 or a higher slated version can have built-in replication feature.
Support stored procedures and stored functions in different languages	SQL Server does support this feature. It can be done with any language which complies with CLR like VB, C#, Python, etc. TO get this done successfully user must first compile the code into all first.	Here there is no need to create a dull first. A user who has created the code can easily see what the code is doing. The server which is downside must host the language the environment is using.
Dynamic actions in SQL	SQL Server does not support this feature. But instead of this user can use the stored procedure and call these from select statements so it is much more limiting than PostgreSQL.	PostgreSQL does provide this feature and just by using select statements a user can perform really all operations and retrieve and do all other jobs easily.
Materialized Views	Yes, it provides the facilities to run materialized views. The functioning though varies depending on where the query is being run. It can be SQL Express, Workgroup, etc.	PostgreSQL does not provide the facility to run materialized views. Instead of this, they have a module called mat views which helps in rebuilding any materialized view.
Case sensitivity	By default SQL server is considered to be case insensitive but if a user wants to change the same they can do it by going down to the column level.	By default, PostgreSQL is case sensitive and it is difficult to make it insensitive. Changes can be made in it but they are not exposed and are not ANSI compliant hence making it a delirious job to use it on MS Access, PHP Gallery, etc. where SQL is regarded to be case insensitive.

2.4 APPLICATION OF SQL

Popescu et al. [13] proposed PRECISE system for Natural Language Interface (NLI) that links each question to SQL query System for finding the categories of questions. In PRECISE, each question is translated into sets of attribute-value pairs for attribute and a relation token for value token. This approach provides the correct answer for 80 % of the questions but questions with unknown words are not handled in PRECISE system. This is the major disadvantage of this system.

Li et al. [14] proposed NaLIX (Natural Language Interface to XML) which is the first generic interactive natural language interface to XML databases. This interface translates the complex English language sentence, with value joins, nesting and aggregation into an XQuery expression that can be evaluated against an XML database. A Database query language called Schema-Free XQuery that maps a query into exact database schema is utilized in NaLIX mainly for retrieving information in XML. In order to generate, validate and translating parse tree to an XQuery expression NaLIX uses syntax-based approach. This approach allows the user to write their own query or they can select the preloaded queries by loading the query template files with the help of an Interactive Query Formulation. NaLIX also provides the reuse facility by storing the whole query history or the set of selected queries in a new template file-text view, grid view and tree view are the three different query result display views.

Li et al. [15] constructed a Natural Language Interface for Relational databases (NaLIR). The author designed a data structure, called Query Tree for both “human understandable” and “RDBMS understandable”. Query tree act as an intermediate between a linguistic parse tree and a SQL statement. The NaLIR executes through three components: the conversion of natural language query to a query tree is performed by the first component, the second component is Interactive Query Mechanism that verifies the transformation interactively with the user and the transformation of query tree into SQL statement is performed by third component.

Kim et al. [16] have proposed new style of QA system architecture which gives a set of candidate query question for user’s information requirement and these candidate questions are automatically generated from significant sentences. Named Entity Recognition technology is used for query generation. This system analyzes the generation of question from a given sentence and isolates the meaningful sentence from generated questions. Manish et al. (2006) proposed an ENLIGHT (IntelligEnt Natural Language Interface) system for information retrieval. This system develops an interface that retrieves information in the form of reply to a non-formal query in NL. ENLIGHT is easy to use and satisfies the actual needs of the user. ENLIGHT utilizes the shallow parsing approach that facilitates the interface with quick response time and handles the semantic symmetry and ambiguous modification properly.

Ramasubramanian et al. [17] designed a Temporal Natural Language Interface (TNLP) to enable interaction and simplify query processing in the temporal database system. A Temporal Event Matching Language (TEML) is designed based on the concept of cursor for the purpose of pattern matching in query processing. The changes in the evolution are measured by the moves of cursor from one instant to another. The TEML act as a standalone language for querying temporal data and it is integrated with SQL for querying temporal databases. Therefore, this system recognizes the evolutionary nature of time, more naturally.

Minock et al. [18] introduced a C-Phrase system for Building Robust Natural Language Interfaces to Databases that enables users to obtain the contents of PostgreSQL database. This approach is a web-based NL front end to relational databases. It translates English queries to Codes tuple calculus and then translates to SQL. Queries may be paraphrased back to the user due to the generation of tuple calculus descriptions. Support of both select and update queries are obtained through the connection of PostgreSQL databases via ODBC and it is used in two primary cases such as personalized and legacy. The user can build the query dynamically to the personalized database using administrative tools in personalized case.

2.5 Related Works

In the study by Jung et al. [19] conduct the work analyses the hybrid indexing consists of tree and hash map for real- time update and query. The paper using hybrid indexing techniques combined both tree and hash map. They conclude that TPR*-tree which performs outstanding spatiotemporal predictive search queries.

Next, is the study by Bhagyashri Anand Jantkal et al. [20] performs study and purpose an algorithm of hybrid of B-Tree and hash map for optimizes search engine indexing. The advantages of the study found that the combined effect of B-Tree and hash map will reduce the number of hash map lookup.

Moreover, is the study by Priyanka Israni et al. [2] proposed work includes the OOD model which handle fuzziness in data to enhance the performance of the proposed model, an indexing technique using r-tree was introduced. The paper concludes that r-tree indexing creates minimum bounding rectangle which contains the group nearby objects.

The study made by Bhosale et al. [3] evaluate indexing techniques being used in academic and industrial applications. The paper using several indexing techniques such as bitmap index, B-Tree index, p-tree index and r-tree index. The evaluation resulted that p+tree and r*tree are more efficient for data warehouse.

The study made by Alhadi Klaib et al. [21] compare advance database indexing for wireless response system. In this work, several techniques are implemented including B-Tree and hash map index. The results show that the hash index may be less efficient than B-Tree for particular string processing applications. This paper is to implement and analyze the application of b-tree in database indexing technique. This case study may be useful for future research in developing new indexing techniques which will added new techniques together with existing techniques as to increase performance of the data query.

3. The Methodology

This section describes how the relevant data and information to address the objectives and research question were collected, presented and analyzed. The overview of the justification and related processes are presented.

The process can be divided into four (4) phases. Firstly, a table is created with B-Tree index on database design phase. Secondly, to load data sources with different size of data for example 10000, 100000 and 500000 records are prepared. Thirdly, the test is executed by recording and evaluating the searching time on each query. Fourth, is the analysis phase where all the data are gathered in the previous phase are analyzed and discussed. Finally, based on all the evidence and results, a conclusion can be made. The processes are shown in Figure 3.

Some examples of how your references should be listed are given at the end of this template in the ‘References’ section, which will allow you to assemble your reference list according to the correct format and font size.

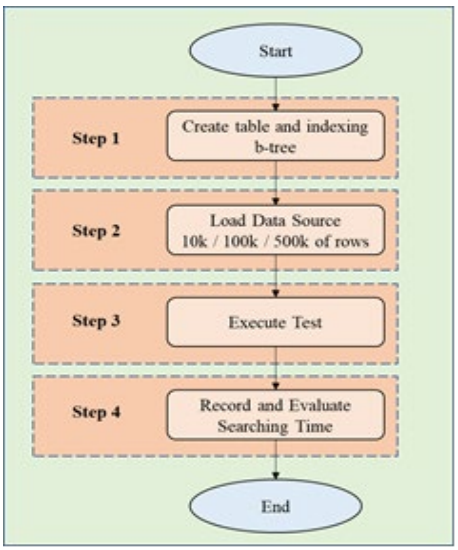

Fig. 3 - Flowchart of B-Tree indexing evaluation

4. The Implementation

This section shows the implementation of the B-Tree indexing technique. The technique then is implemented on PostgreSQL. Moreover, the process for the implementation will be explained and showed in details. Starting from how to create a database, create an index, load data into the database, query and get the result, and the end of the process that is making a comparative analysis based on the result [22], [23].

The example of research process where all the process is shown in Figure 1, where an overall overview of the research is presented and described. The result of test based on speeds that are estimated based upon a mathematical model of computation, or extrapolated from other experiments, will be clearly identified. The metric that will be using is in millisecond of each the process. After that, the result was made between both techniques based on query execution time. Both these techniques then will be implemented on different type of RDMS which is on PostgreSQL and MySQL. Next section will discussion A. Step 1: Create table & indexing. B. Step 2: Load data. C. Step 3: Execute test and Finally D. Step 4: Evaluating the result of the outcomes.

4.1 Create Table & Indexing Database

This section explains how to create a table and B-Tree index. A Table with twenty-two (22) columns are created using SQL statement as shown in Figure 4. If there is no indexing created, PostgreSQL will automatically assign a binary indexing technique once the table is created. The primary key is identified as ID and set to NOT NULL. Once a table is created, indexing can be implemented by using SQL statement syntax CREATE-INDEX-INDEX_NAME-ON-TABLE_NAME-USING-INDEX_TYPES. Based on Figure 5 it shows an example of how to create a simple index for PostgreSQL platform using SQL syntax. Then, each table on what the indexing techniques are used using \D TABLE_NAME is checked. Next section will discuss on how to load the data into the PostgreSQL database.

```
Postgres *# create table b_table10k(
Postgres (# ID int NOT NULL PRIMARY KEY ,
Postgres (# Case_Number varchar (64) ,
Postgres (# Date data,
Postgres (# Block varchar (64) ,
Postgres (# IUCR int,
Postgres (# Primary_Type varchar (64) ,
Postgres (# Description varchar (128) ,
Postgres (# Location_ Description varchar (64) ,
Postgres (# Arrest varchar (64) ,
Postgres (# Domestic varchar (64) ,
Postgres (# Beat varchar (64) ,
Postgres (# District varchar (64) ,
Postgres (# Ward varchar (64) ,
Postgres (# Comunity_Area varchar (64) ,
Postgres (# FDI_Code int ,
Postgres (# X_Coordinate varchar (16) ,
Postgres (# Y_Coordinate varchar (16) ,
Postgres (# Year int,
Postgres (# Updated_on data,
Postgres (# Latitude_varchar (64),
Postgres (# Longitude varchar (64),
Postgres (# Location varchar (64),
Postgres (# );
```

Fig. 4 - Creation Table Using SQL Statement

```
CREATE INDEX b_table ON b_table USING b tree (id);

CREATE INDEX
Time: 3800872.035 ms
```

Fig. 5 - Indexing Creation SQL Syntax

4.2 Load Data into Tables

In this section, the step for loading the data from the source. Firstly, comma-separated value (CSV) file is loaded into the created table as in step 1. Then, Linux script as to upload the dataset in CSV format into the created table. For example, a table name of b_table10k is created. Based on Figure 6 it shows that Linux scripting where a dataset is copied into a b_table10k table. Noted that the same process was implemented for B-Tree in several data sizes. Next section discusses on how to execute the test.

```
$ \COPY
b_table10k(ID,Case_Number,Date,Block,IUCR,Primary_Type,Desc
ription,Location_Description,Arrest,Domestic,Beat,District,
Ward,Community_Area,FBI_Code,X_Coordinate,Y_Coordinate,Year
,Updated_On,Latitude,Longitude,Location)

FROM '/home/postgres/datasource/Crimes10kRecords.csv'
DELIMITER ',' CSV HEADER;
```

Fig. 6- PostgreSQL Load Data 10K

4.3 Test

For execution, each of the tests must be implemented using B-Tree. Every query on each table must be scanned. By using simple SELECT statement of SQL it can identify every parameter including the cost, speed and row on each record. Figure 7 shows the example of PostgreSQL query scan and all its details.

```
postgres=# EXPLAIN ANALYZE SELECT * FROM b_table10k ORDER BY ID;
                                                          QUERY PLAN

------------------------------------------------------------------------------------------------
-----------------------------------------------
 Index Scan using b_table10k_pkey on b_table10k  (cost=0.00..591.21 rows=9999 width=178
) (actual time=0.012..4.027 rows=9999 loops=1)
 Total runtime: 4.344 ms
(2 rows)

postgres=#
```

Fig. 7- PostgreSQL query scan using select

The searching time for PostgreSQL on B-Tree is recorded. The SELECT query scan was executed and analyzed to the table of 10K, 100K and 500K rows. All the result was recorded. Next, will discuss the result for B-Tree.

4.4 Evaluating Result & Discussion

According to the analysis and result from Step 3, the query execution time was evaluated and projected in Table 1, where the results are compiled into tabular form. From the table, clearly shown that B-Tree time retrieval of query takes 1.851 milliseconds (ms) for 10K of rows. Then, for 100K of rows the time is 346.082 ms. Next, for 500K of rows the time taken is 1268.114 ms. This justifies from [24] that the larger the data is the slower the retrieval of the query time. Moreover, the result of this research shows that indexing techniques are significant in term of performance.

Table 2 - Summary of B-Tree analysis result

Table Name	Rows	Scan Operation	Response Time (MS)
b_table10k	10K	Full Scan	1.851
b_table100k	100K	Full Scan	346.082
b_table500k	500K	Full Scan	1268.114

5. Conclusion

For conclusion, this research had discussed the works of the indexing techniques with their algorithm. Through the experiment, this research implemented B-Tree index for PostgreSQL database. By analyzing the result of the outcomes, shows that B-Tree index is significant in database system in term of performance. This research recommends further investigation such as an improved algorithm from existing B-Tree where B-Tree prone to have slower time query as data grows larger. Hopefully, this case study may be useful for future research in developing new indexing techniques which will add new techniques together with existing techniques to increase the performance of the data query.

Acknowledgement

This project was supported by Universiti Tun Hussein Onn Malaysia.

References

[1] Saringat, M. Z., Ibrahim, R., Ibrahim, N., Herawan, T. (2010). On Database Normalization Using User Interface Normal Form, International Conference on Intelligent Computing, Springer, Berlin, Heidelberg. Strunk, W., Jr., & White, E. B. (1979). The elements of style (3rd ed.). New York: MacMillan.

[2] Israni,P., Israni,D. (2017). An Indexing Technique for Fuzzy Object Oriented Database Using R Tree Index. International Conference on Soft Computing and its Engineering Applications (icSoftComp). Changa, India: IEEE.

[3] Bhosale, P., Bidkar, N, Hirave, T., Kanase, P., Magar, M. (2013). Efficient Indexing Techniques on Data Warehouse. International Journal of Scientific & Engineering Research, Volume 4, Issue 5, May-2013Fachinger, J. (2006). Behavior of HTR fuel elements in aquatic phases of repository host rock formations. Nuclear Engineering & Design, 236, 54.

[4] Horowitz, E., Sahni, S., & Anderson-Freed, S. (1976). Fundamentals of data structures (Vol. 1982). Potomac, MD: Computer science press.

[5] Knuth, D. E. (1973). The Art of Computer Programming, Volume 3, Sorting and Searching, Adison-Wesley. Reading, Mass.

[6] Francis, S. F. (2008). Amelioration of r*-tree indexing principles for multidimensional query processing.

[7] Andersson, A., Fagerberg, R., & Larsen, K. S. (2018). Balanced binary search trees. In Handbook of Data Structures and Applications (pp. 151-170). Chapman and Hall/CRC.

[8] Winblad, K. (2018). Dynamic Adaptations of Synchronization Granularity in Concurrent Data Structures (Doctoral dissertation, Acta Universitatis Upsaliensis).

[9] Bounif, L., & Zegour, D. E. (2019). Toward a Unique Representation for AVL and Red-Black Trees. Computación y Sistemas, 23(2), 435-450.

[10] Domicolo, C., Zhang, P., & Mahmoud, H. (2019). The degree Gini index of several classes of random trees and their poissonized counterparts---an evidence for a duality theory. arXiv preprint arXiv:1903.00086.

[11] Francis, S. F. (2008). Amelioration of r*-tree indexing principles for multidimensional queryprocessing.

[12] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT press.

[13] Popescu, A. M., Etzioni, O., & Kautz, H. (2003, January). Towards a theory of natural language interfaces to databases. In Proceedings of the 8th international conference on Intelligent user interfaces (pp. 149-157). ACM.

[14] Li, Y., Yang, H., & Jagadish, H. V. (2005, June). NaLIX: an interactive natural language interface for querying XML. In Proceedings of the 2005 ACM SIGMOD international conference on Management of data (pp. 900-902). ACM.

[15] Li, F., & Jagadish, H. V. (2014). Constructing an interactive natural language interface for relational databases. Proceedings of the VLDB Endowment, 8(1), 73-84.

[16] Kim, M. K., & Kim, H. J. (2008, September). Design of question answering system with automated question generation. In 2008 Fourth International Conference on Networked Computing and Advanced Information Management (Vol. 2, pp. 365-368). IEEE.

[17] Ramasubramanian, P & Kannan, A 2005, „Intelligent Natural Language Query Interface for Temporal Databases“, Web Journal of Formal, Computational & Cognitive Linguistics, University of Kazan, Russia, pp. 1-12.

[18] Minock, M. (2010). C-Phrase: A system for building robust natural language interfaces to databases. Data & Knowledge Engineering, 69(3), 290-302.

[19] Jung, S., Kim, C.S., Lee, J., Hong, B. (2018). Hybrid Indexing Consisting of TPR*-Tree and Hash Map for Real-Time Update and Query of Tactical Moving Objects, IEEE International Conference on Big Data and Smart Computing (BigComp). Shanghai, China

[20] Jantkal, B. A, Deshpande, S. L. (2017). Hybridization of B-Tree and HashMap for Optimized Search Engine Indexing. International Conference on Smart Technologies for Smart Nation (SmartTechCon). Bangalore, India: IEEE

[21] Klaib, Alhadi and Lu, Joan (2013). Development of Database Structure and Indexing Technique for the Wireless Response System. In: Proceedings of the Third International Conference on Advanced Communications and Computation. Infocomp. IARIA, Lisbon, Portugal, pp. 110-116. ISBN 978-1-61208-310-0.
[22] Gunasekaran, S. S., Mostafa, S. A., & Ahmad, M. S. (2014, November). Using the internet as a Collective Intelligence platform in harnessing issues on climate change. In Proceedings of the 6th International Conference on Information Technology and Multimedia (pp. 130-135). IEEE.
[23] Ghani, M. K. A., Jaber, M. M., Mostafa, S. A., Mustapha, A., & Abed, M. (2018). Proper Software Engineering Process in Developing an Integrated Telehealth System. International Journal of Engineering & Technology, 7(3.20), 441-450.
[24] Vyawahare, H.R., Karde, P. P., Thakare, V. M. (2018). A Hybrid Database Approach Using Graph and Relational Database. 2018 International Conference on Research in Intelligent and Computing in Engineering (RICE). San Salvador, El Salvador.