ACCELERATING DATA RETRIEVAL USING INDEX PRIORITIZATION APPROACH

SHATHA ALI MOHAMMED AL-ASHWAL

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2019

ACCELERATING DATA RETRIEVAL USING INDEX PRIORITIZATION APPROACH

SHATHA ALI MOHAMMED AL-ASHWAL

DISSERTATION SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SOFTWARE ENGINEERING

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2019

UNIVERSITY OF MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Shatha Ali Mohammed Al-Ashwl

Matric No: WOC160025

Name of Degree: Master of Software Engineering

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

Accelerating Data Retrieval Using Index Prioritization Approach

Field of Study: Database

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be the owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright, whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:

Designation:

ACCELERATING DATA RETRIEVAL
USING INDEX PRIORITIZATION APPROACH

Abstract

The last few decades have witnessed a huge growth in the size of generated data; the total amount of information that can be saved by all of the world's technical devices is doubling about every 40 months since 1980s. From 2012 to the present, 2.5 exabytes (2.5×10^{18}) bytes of information are produced daily. Database systems have to adjust with this rapid data growth. The capabilities for storing the generated data are also available. The only concern now is how to retrieve the stored data when needed and in a timely and accurate manner.

Many researchers have studied different approaches in the aspect of data retrieval, producing different ways that serves different scenarios. However, the most common way to speed up data retrieval is indexing. There are multiple types of indexing databases, but the most used ones in relational databases are the B-Tree and Bitmap index. These types of indexes speed up query response time, but with a price on storage and performance, as indexes need to be stored and maintained after each delete and write operation. Moreover, these indexes depend on indexing an attribute or two, and not the whole record, which make them limited to a limited number of queries that contain these attributes in the 'where' clause.

This research proposed a covering index that depends on the priority of the records. It is known that data in a table are not in the same level of importance. Some records are more important than the others in a dataset. Some records need to be fetched in a timely manner,

while others do not need to be retrieved very fast. Each company knows the criteria of important records, so it can decide the ranking of the records.

Ranking of records can be done by using triggers or procedures. A procedure or trigger should be created to meet the company's definition or criteria of the priority of the records. Once the records are prioritized, they are sorted according to the rank field. When a query is run, the records are scanned in an order according to their rank; the higher a record in the rank, the first it is going to be scanned. The Priority index overcomes the limitations of the classic indexes, as it does not need maintenance in each write or delete operation. Maintenance can be scheduled and made at night or weekends. Moreover, it can be useful for a variety of bounded queries as it indexes the whole record and not a single attribute. In addition, it is faster than the common index when querying the highly ranked records. The size of Priority index is also smaller than the size of the common indexes.

This work required multiple experiments by running different types of queries on three tables; one indexed by B-Tree index, another one by Bitmap index, and the third by the proposed index. The outcome of the experiments show that Priority index is faster when retrieving highly ranked records, while the size of the Priority index is still smaller.

Abstrak

Beberapa dekad kebelakangan ini telah menyaksikan pertumbuhan yang pesat dalam saiz data yang dijana. Jumlah maklumat yang dapat disimpan oleh semua peranti teknikal di dunia meningkat dua kali ganda bagi setiap 40 bulan semenjak tahun 1980-an. Pada tahun 2012 sehingga kini, sebanyak 2.5 exabytes (2.5×10^{18}) bait maklumat dihasilkan setiap hari. Oleh itu, sistem pangkalan data perlu memenuhi spesifikasi bagi menyediakan keperluan seiring pertumbuhan data yang pesat ini. Keupayaan untuk menyimpan data storan yang dijana telah tersedia namun cabaran utama adalah bagaimana untuk mendapatkan semula data yang disimpan pada waktu dan mengikut ketetapan masa yang bertepatan.

Ramai penyelidik telah menjalankan kajian dengan menggunakan kaedah berbeza dalam aspek menjana pertanyaan dalam pemprosesan rekod data pada keadaan senario yang pelbagai. Kaedah yang biasa digunakan untuk mempercepatkan pemprosesan janaan data adalah pengindeksan. Terdapat pelbagai pangkalan data jenis pengindeksan. Walau bagaimanapun, hubungan pangkalan data jenis B-Tree dan indeks Bitmap sering digunakan, iaitu jenis indeks yang memberi tindak balas cepat kepada janaan pertanyaan data dengan ruang penyimpanan data dan prestasi indeks yang perlu kekal disimpan setelah operasi memadam dan menulis dilakukan. Indeks ini bergantung kepada pengindeksan satu atau dua atribut sahaja dan bukan pada keseluruhan rekod, menjadikan bilangan janaan pertanyaan terhad dan tertentu, dan mengandungi atribut-atribut yang termaktub pada klausa sahaja.

Kajian ini meliputi indeks yang bergantung kepada keutamaan rekod. Seperti yang diketahui, data di dalam jadual berada pada tahap yang tidak sama kepentingan keutamaan di mana sebahagian rekod data adalah lebih penting daripada set data yang lain. Sebahagian

rekod perlu diperoleh pada tempoh masa yang tepat, manakala sebahagian yang lain tidak perlu penjanaan segera. Setiap syarikat perlu mengetahui kriteria sesuatu rekod penting agar dapat menentukan kedudukan rekod berkenaan.

Kedudukan rekod boleh diperolehi melalui kaedah pencetus atau berdasarkan prosedur. Prosedur atau pencetus perlu dihasilkan bertepatan definisi atau kriteria keutamaan rekod syarikat. Setelah rekod disusun mengikut keutamaan, ia disusun mengikut medan susunan. Apabila janaan pertanyaan diberi keutamaan untuk pemprosesan, rekod pada medan susunan diimbas berdasarkan susunan kedudukan, iaitu semakin tinggi kedudukan pada susunan rekod maka ia akan diimbas terlebih dahulu. Kaedah Indeks keutamaan ini dapat mengatasi jenis indeks biasa yang digunakan kerana tidak memerlukan penyelenggaraan pada setiap operasi menulis atau memadam. Ia boleh diselenggara mengikut jadual dan ketetapan pada waktu malam atau hujung minggu. Selain itu, ia sangat berguna untuk pelbagai pertanyaan rekod tanpa had kerana menjana keseluruhan indeks bukan hanya pada satu atribut tunggal. Ia juga lebih cepat dijana berbanding indeks biasa apabila menjana rekod data pada kedudukan tinggi. Saiz jenis indeks keutamaan juga lebih kecil berbanding saiz indeks biasa.

Di dalam kajian ini, beberapa eksperimen telah dilaksanakan dengan pelbagai pertanyaan dijana ke atas tiga jenis jadual iaitu pada indeks jenis B-Tree, jenis indeks Bitmap serta indeks yang dicadangkan. Hasil eksperimen menunjukkan bahawa indeks keutamaan lebih cepat menjana rekod pada susunan kedudukan yang lebih tinggi dengan saiz indeks keutamaan yang tetap kecil.

ACKNOWLEDGEMENTS

Alhamdulillah, all praises to Allah for blessing me and giving me the strength to finish my master's degree at the University of Malaya. It is my privilege to express my gratitude to all those who have helped me during the completion of this work.

I express my deepest gratitude to my supervisors Dr. Maizatul Akmar Binti Ismail and Dr. Mumtaz Begum Binti Peer Mustafa for provoking ideas, encouragements, helpful insights, valuable assistance, useful comments as well as meticulous reading and editing of the draft of this thesis. This research would never be accomplished without their supervision.

I must express my very profound gratitude to my parents for their continuous encouragement and love that never stops. I would not be where I am today without their help and support.

I dedicate this thesis to my family; my beloved husband Niyazi Al-Ashwal who supported me during my studies and believed in me, and to my children Aiham and Farah for their love.

Last but not least, I would like to thank my siblings for being a special part of my life and a source of inspiration.

Contents

Chapter 1: INTRODUCTION

1.1 Overview

Humans have always stored data. The tally sticks are the earliest example of data storing. It was invented in C 18,000 BCE. People used to make marks into sticks or bones to keep track of trading activity and food supplies. In C 2400 BCE, the abacus was constructed in Babylon, which is a dedicated device for performing calculations. In addition, libraries appeared in this period, representing humans' first attempt at mass data storage. 1663 was the year of statistical Emergence. In that year in London, John Graunt made the first recorded experiment in statistical data analysis. By recording information about death, he theorized that he can design an early warning system for the bubonic plague ravaging Europe. In 1970, IBM mathematician Edgar F Codd presented his framework for a "relational database". The model provides the framework that many modern data services use today, to store information in a hierarchical format, which can be accessed by anyone who knows what they are looking for. Quoted from (Marr, 2015).

This teaches us that humans have always been storing data, and with advanced technology, people are able to store more data. The important issue nowadays is how to retrieve the data in a timely manner, and how to find answers for our questions in this pool of data.

1.2 Research Background

As a reason of the recent technologies such as internet and Smart phones, we are encountered by a huge size of data in many fields. Databases are increasing rapidly in size. These days, companies have started to keep data that they used to get rid-off in the past, the

kept data gives valuable information for the companies. For example, a company can predict a customer's behaviors from the pages the customer visits, the comments the customer writes, and the searches the customer does, which means that they need to store a lot of data and retrieve useful information when needed.

It is easy to store data as long as the capacity is available. Companies now find themselves having a huge size of data that most of the time, it is cheaper to keep all the data than deciding which to keep, but it can prioritize the importance of the data. A company can benefit from big data, to have better insights, understanding issues that can lead the company to increase its profits and find new opportunities in business.

Big data have five characteristics, which are volume, velocity, variety, veracity, and value (Philip Chen and Zhang, 2014). Those characteristics are what make managing big data a challenging task. Data comes in big size and in different types, such as texts, videos, images, and audios. It also expands in a high rate, and it contains a lot of noise.

Accelerating data retrieval is a crucial aspect in today's industry. There are multiple approaches that deal with accelerating data retrieval. One approach is indexing the data, which improves the speed of data retrieval operations on a dataset, at the cost of additional writes and storage space, to maintain the index data structure. Another way is dealing with the necessary data only using Materialized Views.

1.3 Research motivation

As a result of the progress of data storage devices, companies have started to store more data and never get rid of any data. The problem that rose was among all the stored data is it became difficult to find the important data the company needs. In addition, the big amount

of data slowed down data retrieval. Nowadays, it is easier to store data than to retrieve it. Moreover, it is sometimes cheaper to keep data than determining which data is worth keeping

In the last decade, many researchers undertook significant amount of work to improve the performance of database retrieval. There are multiple ways that speed up retrieval time, such as indexing, and materialized views. Each approach has limitations that can be expensive in regards to accuracy, space, and maintenance.

As data is growing rapidly, data retrieval is slowing down, and to speed up data retrieval we need to improve the existing approaches so that queries can run faster and companies can get timely accurate reports that lead to better decision making.

1.4 Statement of Problem

The last decade witnessed a massive switch to digital technology, digitizing information of all kinds, such as audios, books, photos and videos. Households then began to switch to digitization, and the information they held (documents, photos, videos, music, etc.) were switched over in turn. By the 2000s, the digitization process was complete, as digital television, e-books, digital cinema, and so on, opened up a new era with loads of digital data to store. In addition, multiple sensors provide a huge volume of information (Bounie and Gille, 2012). The growth of data and accumulation of complex data collections have become a challenge for information retrieval (Fasolin et al., 2013).

Querying big data is time consuming. Databases nowadays are increasing at a rapid rate as we are collecting more data from different sources, like users, electronic devices, and media. In addition, we have started to keep all the data, as sometimes it is cheaper to keep

all the data than finding which data is worth keeping. For example, a linear scan of a dataset of PB size takes days using a solid state drive with a read speed of 6 GB/s, and it takes years if a dataset is of EB size (Fan et al., 2015).

Indexing is the most common approach for speeding up query retrieval time. There are muliple types of indexing database, such as B-Tree, Bitmap, and Hash index. Each type is used for determined circumstances, but all index approaches are designed according to columns and not the entire query, which slows the query operations (Wu et al., 2017). Moreover, they increase the size of data needed to be stored, which means the database will increase more in size. In addition, maintaining an index is a high latency job because the database management system have to locate and update the index pages that are affected by tables changes (Yu and Sarwat, 2016).

1.5 Research Objectives

The aim of this research is to develop a database retrieval approach that speeds up the retrieval time in big data with fewer side effects than the existing approaches i.e. the proposed index will occupy less space and require less maintenance.

The following objectives were set to achieve the main aim of this research:

1. To develop a suitable index scheme approach to accelerate the query retrieval time in big data.
2. To evaluate and compare the proposed approach with the common available approaches in terms of retrieval time, space occupied and maintenance.

1.6 Research Questions

Objective 1: To develop a suitable index scheme approach to accelerate the query retrieval time in big data.

- How can the retrieval time of data be improved?

Objective 2: To evaluate and compare the proposed approach with the common available approaches in terms of retrieval time and space occupied.

- What are the factors in evaluating the performance of database index?

1.7 Scope of research

This research focuses on accelerating data retrieval time in relational database. A suitable approach is going to be developed; this approach depends on prioritizing the dataset records. Two different datasets will be downloaded from the internet, rebuilt and imported to the database. Four tables will be created. Each two tables are the same, containing the same data. One table of each dataset will be indexed by the proposed index scheme, the other table of each dataset will be indexed once using B-Tree index and then by using Bitmap index. Experiments will be conducted to compare the proposed index with the B-Tree index, and the Bitmap index which are the most used index schemes in relational database. In these experiments we will use bounded queries with different selectivity.

1.8 Expected research outcomes

The expected outcome is to develop an index approach that outperforms the common index approaches. The proposed approach is expected to have the following advantages over the common index approaches

- Faster retrieval time for higher priority records.
- Less space needed for the index.
- Easier maintenance, as it is not going to be needed during each write.

1.9 Significance of the research

This is an era of digital technology. We are faced with massive switch to digital technology, and databases have to store more data than it used to in the past.

This research should be of great benefit as database system applications are encountering huge numbers of records and companies are starting to keep all the data without getting rid of any.

The proposed approach should be beneficial in database system, whether they are using data warehouses or not, as this approach will help in accelerating the query retrieval time and will reduce the needed maintenance.

1.10 Thesis Organization

The rest of the dissertation is organized as follows:

Chapter (2): The second chapter reviews the relevant literatures of speeding up data retrieval in big data. This chapter is divided into two parts:

- The first part reviews the generally used approaches to enhance database retrieval time.
- The second part reviews the different types of index approaches and their limitations.

Chapter (3): This chapter is dedicated to the research methodology.

Chapter (4): This chapter is dedicated to the design and development of the proposed approach.

Chapter (5): This chapter contains the conducted experiments and evaluation of the proposed approach; it discusses the performance of the proposed approach, as well as the comparison against the common index types.

Chapter (6): This chapter contains the findings of our research, explains the limitations of the work, and provides suggestions for future work that can be carried from this research.

Chapter 2: LITERATURE REVIEW

2.1 Introduction

Nowadays, databases have to store huge amounts of data, as data is being generated at a rapid rate. Eric Schmidt, executive chairman of Google, said in a conference that "as much data is now being created every two days, as was created from the beginning of human civilization to the year 2003". Data growth has many reasons, such as the growth of the Internet, the use of advanced sensors, and the switch to digital technology. Database systems store the generated data to be used in the system to benefit the decision maker, the system owners, and the customers.

Hardware also is being improved, servers could now process zettabytes (trillion gigabytes) of data, which in turn encouraged the companies to keep all of the data and never get rid of any of them. Companies are keeping track of almost all of their transactions and all of their logs. Sometimes, it is cheaper to keep all the data than deciding which data is worth keeping.

As the collected data increases and its complexity grows, the challenge of retrieving data in a timely manner grows higher. (Fasolin et al., 2013), for example, a sequential scan of a PB dataset in size takes days using a compact drive with a read speed of 6 GB/s, and it takes years if a dataset is EB in size (Fan et al., 2015).

Currently, most big data analysis is for structured data that can be handled in relational database, but the problem that it faces is the query retrieval time (Durham et al., 2014).

Many researchers have worked in enhancing query operation time as a need to enhance the overall performance of the systems and database warehouses, where several issues are

illustrated. When volume and variety of data are involved, the required time for queries to retrieve the needed data is affected negatively, and so different solutions were discussed in different researches. (Durham et al., 2014) proposed the use of a new model that decreases the volume of handled data by the application, which in turn leads to a lower retrieval time. They rebuilt the big data applications to serve in this way using relational database. Other researchers preferred speeding up the query operation by using indexes, such as (Yu and Sarwat 2016; Chong et al., 2016); Goldstein et al., 1998; Wu et al., 2006). (Fan et al., 2015) explored the possibility of querying big data by accessing a limited amount of the data. They were solving the issue of time consumed when querying big data, and trying to speed up retrieval time with the least possible cost.

2.2 Relational Database

A relational database stores data in tables that have relations among them as records, and each record contains multiple fields. Data can be accessed, updated, or deleted by running SQL queries. Relational database have many positive characteristics, as they have been developed and used for decades; they are famous for their reliability, security, and handling complicated queries. (Fan et al., 2015 and Durham et al., 2014) suggested that applications should use relational databases when complicated queries have to be handled or have repetitive data analysis. Currently, most big data analysis is for structured data that can be handled in relational database, but they are facing query retrieval time issues as developers think of how the application would put the data in and ignore how they will retrieve the data (Durham et al., 2014).

2.3 Materialized views

Materialized views are used to speed up query retrieval time in the database system and in database warehouses. Materialized views are different from the regular views in many aspects as they are stored in true tables, which are not virtual ones, while regular views are stored in virtual tables. Materialized views can be indexed, while regular views cannot. Querying a materialized view is like querying a table (Sharma and Sood, 2014). Materialized views have a wide diversity of data management issues such as query enhancement, maintenance, and data integration (Pottinger and Halevy, 2001)

Materialized views reduce query retrieval time (Sohrabi and Azgomi 2017; Sohrabi and Ghods 2016; Zhou et al., 2007) by pre-computing all the aggregations and join operations, in contrast to the regular views that only compute aggregations and joins when a calling query is run. Materialized views can be used in many areas, but they are mostly used in systems that support decision making and in data warehouses. A critical issue in materialized views is that they should be updated regularly, so when a query is run against a materialized view, the results will not be accurate unless the materialized view is up-to-date (Sharma and Sood, 2014). Materialized view's drawback is the cost of storage and maintenance. Materialized views occupy true space, and they have to be refreshed regularly whenever there is a change in a base table. To sum up, materialized views speed up query execution time and they can be enhanced by indexing. Choosing a suitable index would improve the performance of a materialized view.

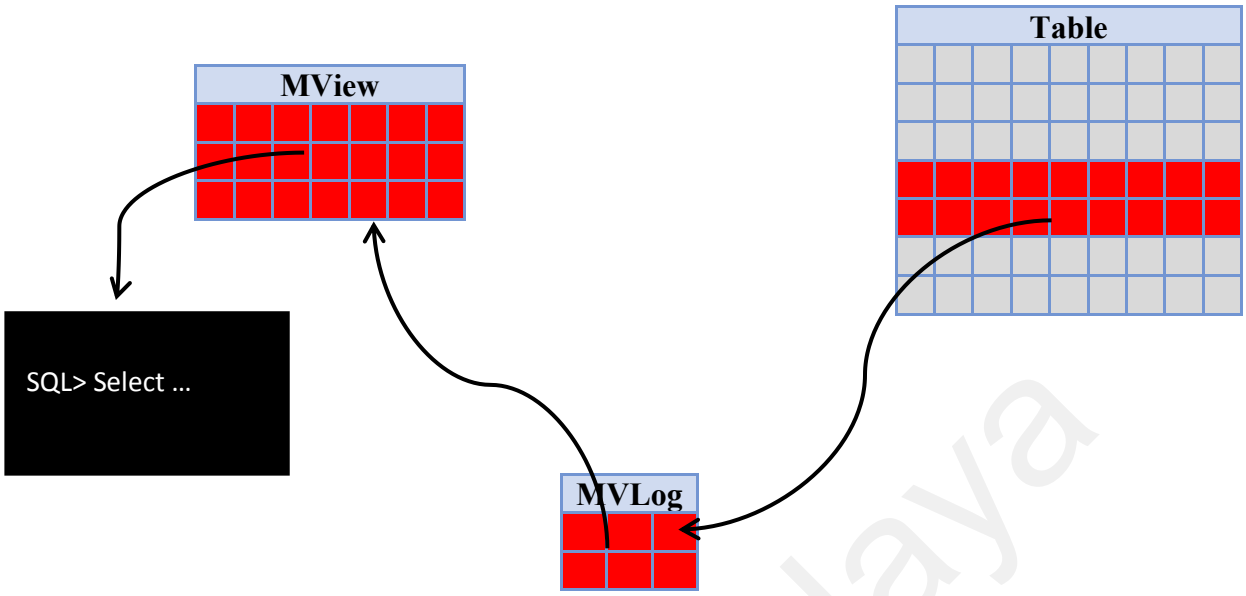


Figure 2.1: Materialized view mechanism

2.4 Scale Independence

Scale Independence means that even if the dataset Ds is big or grows bigger by the day, the subset dataset Dsq will remain almost the same because it is independent of the size of the dataset Ds. So, the retrieval time will still be small independent of the size of Ds dataset.

$Dsq \subset Ds$ and

$Q(Dsq) = Q(Ds)$

In this case, we can customize our query to only scan the needed records, such as using group-by and rollup clauses (Bellamkonda et al., 2013) i.e. we can use bounded query.

A study conducted by (Fan et al., 2015) used bounded evaluable queries to access small data of the big dataset, avoiding to scan the entire dataset. They used bounded queries, but when the queries are not bounded, they used two approaches to answer the query by scanning limited records. First, they searched for upper and lower envelopers that are bounded for the query. Second, they initiated the needed parameters for the query to

become boundedly evaluable. They studied the way of changing an unbounded query to a bounded one and illustrated the difficulty of different types of queries. The main take away from this study is that an unbounded query can be changed to a bounded one, and a bounded query outperforms an unbounded query. In summary, most queries can be a bounded evaluability, which means that we can decrease the number of records that a query has to scan if we try to enhance the used query by making it bounded.

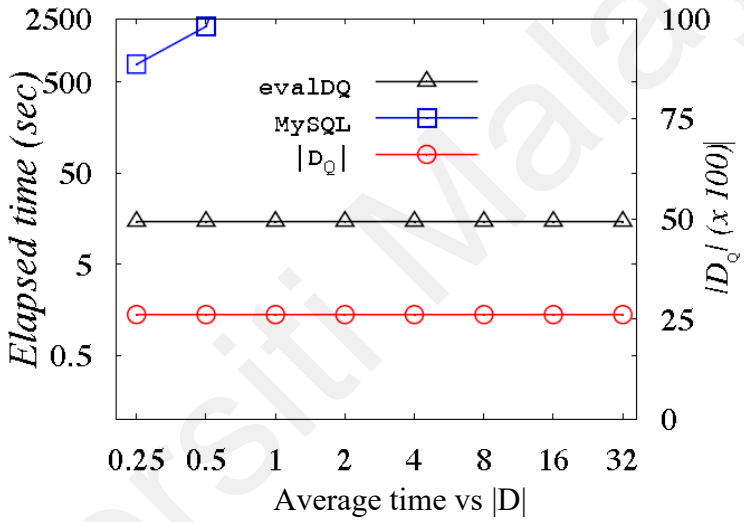


Figure 2.2: Comparison between generated bounded query plan vs mysql query plan by (Geerts 2016)

2.5 Prioritization

Prioritize definition in Oxford English dictionary is "1 decide the order of importance of a number of tasks. 2 treat something as being more important than other things." Prioritization is used in many things in life from the daily tasks to the goals in life. Prioritization has been used in multiple different fields helping in concentrating on what matters mores and starting with the important things and avoiding the distracting matters from disturbing. For instance, (Goldsmith et al.,

2014) used prioritization to develop a consumer product ingredient database for chemical exposure.

Software companies usually has multiple requirements when building a software product but they are often faced with limitations in resources (Bebensee et al., 2010) used binary priority list to develop a tool that prioritize software requirements which would ease choosing which requirement to implement.

(Garg and Datta, 2012) used prioritization in testing. Test cases were prioritized when doing regression tests so errors are detected early; in their work they proposed a novel automated test case prioritization technique that automatically finds any changes in the database, prioritizes test cases according to degree of relevance to the changes in the database, and executes them in priority order. (Narayanan and Waas, 2011) introduced a mechanism that prioritize queries in database systems according to their importance, the database administrators are also capable of changing the priority of a query, series of experiments were conducted to show the efficiency of this mechanism and achieved near optimal results.

2.6 Indexing

Indexing is a data structure approach that improves the speed of data retrieval time on a dataset at the cost of additional writes and storage space, to maintain the index data structure. Index performance is measured by retrieval time, index size, and the required maintenance. “A database index usually yields 5% to 15% additional storage overhead. Although the overhead may not seem too high in small databases, it results in non-ignorable dollar cost in big data scenarios” (Yu and Sarwat, 2016).

Database indexes need maintenance in each write or delete process. A DBMS has to reorder the index after inserting to, or deleting from an indexed table. DBMS has to locate and reorder affected indexes after any operation, which makes maintenance a time consuming job, which affects the performance of the overall system (Yu and Sarwat, 2016).

(Philip Chen and Zhang, 2014; Qin, 2016) made an experiment that compared between the most common index approaches, which are B-Tree, Bitmap index, Physical Data Block Range Index, and Logical Data Block Range Index. In the conducted experiments, they compared between the four indexes in terms of required time to build the index, space occupied by the index, query operation time in various selectivity, and index maintenance cost. We can conclude from their experiment that Bitmap index and B-Tree index are slow in query retrieval time, and that these indexes occupy large spaces, as compared to Logical Data Block Range Indexes. However, Logical Data Block Range Indexes are not suitable for hot database as all blocks may contain eligible records, which mean fewer records may be excluded. A hot database is a database that always has transactions.

Table 2.1: A comparison between the common index types

	Retrieval time	Space occupied	Maintenance time	Create time	Remarks
Bit map	When the selectivity is 0.01%, retrieval time is 100 ms	Using 300 GB of data, index size is 67MB	1900 ms when inserting 100 k new records	Using 300 GB of data, creation time is 1890s	Slow retrieval time and high cost for maintenance and storage
B-Tree	When the selectivity is 0.01%, retrieval time is 129 ms	Using 300 GB of data, index size is 2.4G	1600 ms when inserting 100 k new records	Using 300 GB of data, creation time is 4800s	Slow retrieval time and high cost for maintenance and storage
Physical DBRI	When the selectivity is 0.01%, retrieval time is 64 ms	Using 300 GB of data, index size is 88.3 MB	78 ms when inserting 100 k new records	Using 300 GB of data, creation time is 600 ms	In hot database it may break as all blocks may contain eligible records which means fewer records may excluded
Logical DBRI	When the selectivity is 0.01%, retrieval time is 87 ms	Using 300 GB of data, index size is 72.7 MB	50 ms when inserting 100 k new records	Using 300 GB of data, creation time is 602 ms	In hot database it may break as all blocks may contain eligible records which means fewer records may excluded

2.6.1 B-Tree index

The B-Tree is similar to a Binary tree search, but it is more complicated as it has multiple branches per node instead of two (Adamu et al., 2015). It is used because it keeps keys in sorted order for sequential traversing as relational database is slower at performing random seeks.

“B-Tree indexes satisfy range queries and similarity queries, also known as Nearest Neighbor Search (NNS), using comparison - operators (<, < =, =, >, > =). B-Tree algorithm consumes huge computing resources when performing indexing on Big Data” (Alvarez et al., 2015).

The performance of a B-Tree index is measured by the speed of retrieving data, size of index, and cost of maintenance.

Index operations, illustrations taken from wikipedia.

Write operation:

To insert a new record, the tree is searched to find the leaf node where the value should be added.

Steps to add a new value:

- If the node contains fewer than the maximum allowed number of elements, then there is room for the new value. Inserting the new value in the node keeps the node's elements ordered.
- If the node is full, it is evenly split into two nodes, so:

  A single median is chosen from among the leaf's elements and the new element.

  Values less than the median are put in the new left node and values greater than the median are put in the new right node, with the median acting as a separation value. The separation value is inserted in the node's parent, which may cause it to be split, and so on. If the node has no parent, in other words, if it is a root node, a new root is created above this node (increasing the height of the tree).

Deletion:

There are two strategies for deletion from a B-Tree; Locate and delete the value, then restructure the tree to retain its stable state, or restructure the tree before deleting. There are two special cases to consider when deleting an element:

1- The element in an internal node is a filter for its child nodes

2- Deleting an element may put its node under the minimum number of elements and children

The procedures for these cases are in the order below.

Deletion from a leaf node:

1. Search for the element to delete.
2. If the value is in a leaf node, simply delete it from the node.
3. If underflow happens, rebalance the tree.

Deletion from an internal node:

Each element in an internal node acts as a separation value for two subtrees, therefore we need to find a replacement for separation. Note that the largest element in the left subtree is still less than the filter. Likewise, the smallest value in the right subtree is still greater than the filter. Both of those elements are in the leaf nodes, and either one can be the new filter for the two subtrees.

Limitations of B-Tree:

- The index is built on one or more attributes and not the whole record, i.e. it speeds up limited number of queries.

- The index size grows rapidly when increasing the size of the dataset.
- Maintenance is expensive, as in each write or delete, there is a need to reorder the index.

ID	First name	Department
5	Ahmad	Finance
28	Ali	Finance
33	Sara	Finance
41	Arwa	IT
46	Aiham	Finance
47	Farah	IT
85	Nathir	IT
99	Waddah	Finance

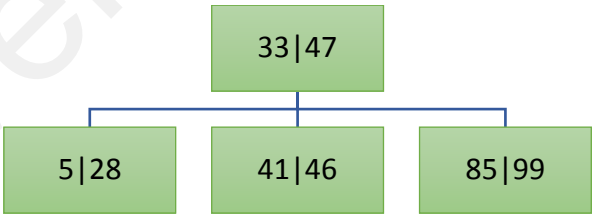


Figure 2.3: Example of B-Tree index

2.6.2 Bitmap index

Bitmap indexes are efficient for low-cardinality columns (Wu et al., 2006), which have a small number of distinct values. For example, sex column in a table can be

indexed by Bitmap as it has two values, either M or F. Bitmap indexes use bit arrays and answer queries by performing bitwise logical operations on these bitmaps. Bitmap indexes occupy a smaller space compared to B-Tree index as it only stores the record position and series of bits. Their disadvantage is that their query operation time slows down in high cardinality attribute, especially unique columns. The performance of a Bitmap index is measured by the speed of retrieving data, size of index, and the cost of maintenance.

Name	Sex
Hadil	F
David	M
Sam	M
Nasim	F
Amat	F
Layal	F
Amal	F

Bitmap Index
0 1
1 0
1 0
0 1
0 1
0 1
0 1

Figure 2.4: Example of Bitmap index

2.6.3 Covering index

A covering index is an index that contains all of the attributes needed to be fetched from a table through a query. Using a covering index speeds up the retrieval time as once a record is located, it is retrieved immediately. In the other index schemes, the

index is used just to locate the needed data, to resolve the query. After locating the needed data, the next step is to go to the location and fetch the required records. The maintenance of a covering index may be more expensive if it was not built appropriately. Choosing a general covering index affects the overall performance (Kaushik et al., 2002).

2.6.4 Recent work that tackles one or more database index challenges

As the data grows more and as more studies try to improve the common indexing approaches, some studies are dedicated to a certain field while other studies are useful in specific conditions, but all are trying to speed up the time for query operation.

2.6.4.1 Scientific big data

In the last few decades, scientific data is being generated at a rapid rate. As it grows in size, it becomes more difficult to organize in an efficient manner, so it becomes even slower to find a needed data. Bit map index can be useful in indexing scientific big data as it is efficient in indexing multidimensional data for speedy data retrieval time. It is more efficient in the scientific case as there are less write operations. Most of the time, scientific data are stored in distributed clusters, while the traditional Bitmap scheme is designed for single server. (Chong et al., 2016) improved the Bitmap index for scientific big data by introducing FastBit, in a distributed environment. They proved that this approach, 'FastBit', outperforms the classic Bitmap both in small and large data sizes by conducting experiments on astronomical big datasets.

2.6.4.2 Compressed Bitmap scheme

Compressing a Bitmap index squeezes its size, which is a great benefit as it reduces the size of an index. Word-Aligned Hybrid (WAH) is a compression Bitmap index which reduces the size of an index and improves the query operation time. This scheme was proposed by (Wu et al., 2006). It answered multidimensional queries by using compressed Bitmap indexes, as other indexes like B-Tree index cannot be used to answer multidimensional queries. This study proved that WAH compressed Bitmap index outperforms projection indexes, as they are three times faster in retrieving data and can be used in the case of high-cardinality columns. This scheme solved the size and high cardinality issues with the Bitmap index, but did not solve the problem of using Bitmap in a frequently updated dataset. (Fusco et al., 2010) used the principle of Bitmap compression scheme to develop a prototype using commodity hardware for network flow data, based on the on-the-fly compression and optimized Bitmap index, with an online LSH-based constructing scheme to save even more space.

2.6.4.3 Compressed B-Tree scheme

Similar to compressed Bitmap scheme, compressed B-Tree scheme reduces the space size dramatically. An example of a compressed B-Tree index is the compressed index proposed by (Goldstein et al., 1998). The proposed approach decreases the space size of the index, and gives the ability to decompress an individual field rather than the whole relation at a time. This approach indexes the low to medium cardinality fields of numbers, which can be used in decision support

systems, as the biggest part in these application is the fact tables which do not contain text columns, and the columns are of low and medium cardinality. The drawback of this approach is that it can only be used with number attributes that are not high cardinality. In other words, it cannot be used with text fields or with any type of fields that are high cardinality.

2.6.4.4 Hippo index

(Yu and Sarwat, 2016) proposed Hippo index which is a scalable and fast database indexing approach. Hippo stores disk page range instead of column pointer, which leads to smaller index size. When a query is run, Hippo weighs the page ranges and histogram-based page summaries to find out the pages that do not contain the answer for the query, and predicates and inspects the remaining pages. An experiment based on real datasets was conducted and the results showed that Hippo occupies less storage than B-Tree, but still has the same level of performance as B-Tree, meaning no change in query retrieval time. To sum up, Hippo index has many advantages as it reduces the size of the index and ease the maintenance without affecting the query operation.

Table 2.2: Summary of recent works improving common index

Title	Authors	Work significance
On-the-fly compression, archiving and indexing of streaming network traffic	Francesco et al., 2010	Used the principle of Bitmap compression scheme to develop a prototype for unique on-the-fly solution for archiving and indexing of network flow database
Performance Comparison of Index Schemes for Range Query of Big Data	Xiongpai QIN, 2016	A study of the common index schemes (B-Tree, Bitmap index, Physical Data Block Range Index, and Logical Data Block Range Index)
Accelerate Bitmap indexing construction with massive scientific data	Chong, Li, Chen, & Zhu, 2016	Improved Bitmap index for scientific big data by introducing FastBit, in a distributed environment
Compressing relations and indexes	Goldstein, Ramakrishnan, and Shaft, 1998	The proposed approach decreases the space size of the index, and gives the ability to decompress an individual field rather than the whole relation at a time. This approach indexes the low to medium cardinality fields of numbers.
Two birds, one stone: a fast, yet lightweight, indexing scheme for modern database systems	Yu, and Sarwat, 2016	Less space storage than B-Tree.
Optimizing Bitmap indices with efficient compression	Kesheng Wu, Ekow Otoo, and Arie Shoahani, 2006	Concentrates on the efficiency of using compressed Bitmap indices to answer multidimensional range queries. This approach is also effective in the case of high-cardinality attributes.

2.7 Summary of literature

There have been many studies in the field of accelerating query response time. Indexing is the most common approach that speeds up query operations. Indexing approaches speed up query retrieval time but they are designed according to columns and not the entire query, which slows the query operations (Wu et al., 2017). Moreover, they increase the size of data needed to be stored, and maintaining an index is a time consuming job because the database management system has to locate and update the index pages that are affected by table changes (Yu and Sarwat, 2016).

Prioritization is used in different fields to help to concentrating on the meaningful and important things, instead of wasting valuable time on the unnecessary. (Bebensee et al., 2010; Garg and Datta 2012; Narayanan and Waas 2011; Goldsmith et al., 2014) all used prioritization in their fields of study.

The most used index schemes are Bitmap index and B-Tree index. These two types have limitations. These limitations worsen in big data scenarios (Yu and Sarwat, 2016). Many researchers use these two indexes or one of them as a benchmark, such as (Yu and Sarwat, 2016; Wu et al., 2006; Chong et al., 2016).

It is noticed that some work have been dedicated to a significant field such as (Chong et al., 2016) that improved scientific data indexing using Bitmap. Others have concentrated on the size of index only (Fusco et al., 2010; Wu et al., 2006; Goldstein et al., 1998). (Yu and Sarwat, 2016) improved both, size of index and maintenance performance.

Index approaches can be improved if it gets the use of scale independence ideas, meaning that the index scans as less records as possible.

Chapter 3: RESEARCH METHODOLOGY

This chapter focuses on the research methodology adopted in this research to achieve the research objectives. The main objective is to propose an approach that speeds up the query operation time with fewer side effects, i.e. maintenance, and storage size.

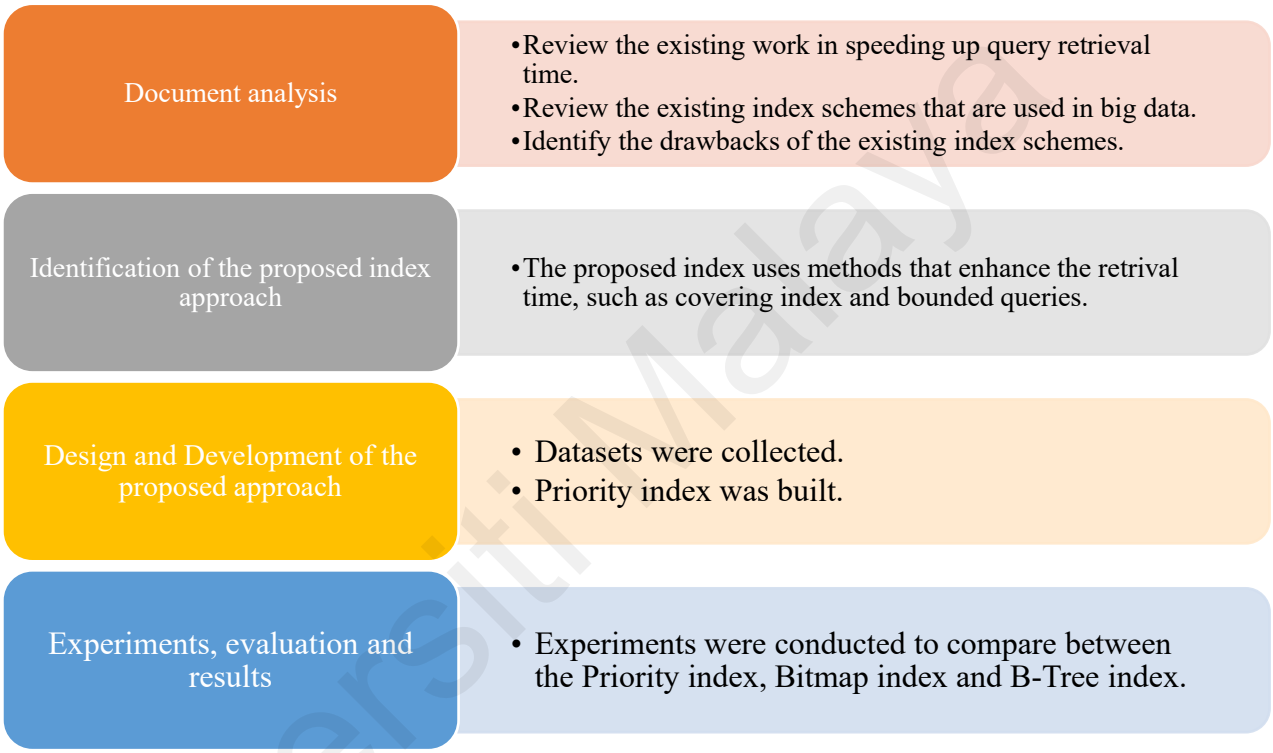


Figure 3.1: Research Methodology

3.1 Document analysis

The aim of the literature review of related researches is to obtain useful information in speeding up query operation time approaches, and to find the limitation of the used index schemes. We went through plenty of researches; some were dedicated to solving a single-issue, others proposed wider solutions.

3.1.1 Main insights

Indexing is the most common way to speed up queries operation time. Materialized views are also used to speed up query operation, but materialized views can be enhanced by indexing them.

Prioritization is used in many different fields to concentrate on the important.

Indexes have some limitations, such as they need to be reordered after each write or delete operation. Indexes are also built on one attribute, and not the whole record, which limit the number of queries that an index can speed up. Using Covering index speeds up the retrieval time, as when a record is found, it is fetched immediately as the index contains all the attributes that are needed to be fetched from a table. So once a record is located, it is fetched immediately, unlike other indexes that have two steps, first to locate the record and next to fetch it. Scale independence illustrates that not all records has to be scanned to get the query result.

3.2 Identification of the proposed index approach

Referring to the insights that were driven from the literature review, the scheme of the Priority index was determined, which is an index proposed by the author.

The main features of Priority index are:

1. It ranks the records, from higher to lower, according to priority.
2. It scans the records linearly, from a higher rank to a lower rank.

3. It is a covering index, i.e. once a record is located, it is returned immediately. The query engine does not need to look up the records, since all of the requested columns are available within the index.
4. It is not built on a specific attribute, which means it speeds up the query operation regardless of what is in or is not in the where clause.
5. The idea of Priority index was taken from scale independence discussed in chapter2.

3.3 Design and Development

3.3.1 Data collection

This work uses two existing datasets:

1. NYC Taxi and Limousine Commission: NYC yellow taxi dataset was downloaded from the official web pages of the NYC Taxi and Limousine Commission (NYC). Twelve CSV files were downloaded, each file represents the rides information of a month in the year 2017. This dataset was prioritized according to date, meaning that the more recent the data the higher the priority. This dataset was chosen because it was used in a published work (Yu and Sarwat 2016) that developed an index for big data. 36,000,000 records from this dataset were used sized around 2.6 Gigabytes. The fields of this dataset are: (VendorID, tpep_pickup_datetime, tpep_dropoff_datetime, Passenger_count, Trip_distance, PULocationID, DOLocationID, RateCodeID, Store_and_fwd_flag, Payment_type, Fare_amount, Extra, MTA_tax, Improvement_surcharge, Tip_amount, Tolls_amount, Total_amount).

2. Canadian International Merchandise Trade (CIMT): online database offers detailed trade data using the Harmonized System (HS) classification of goods (based on the 6-digit commodity level) ((CIMT) 2017 -2018). The fields of this dataset are: (HS_CODE, UOM_EXPORTS, COUNTRY, STATE, GEO, VALUE, QUANTITY, YEAR, MONTH). This dataset was chosen as it is suitable for comparing the proposed index with the common indexes, as the code column in this dataset have a variety of values. This means it has high selectivity, which in turn leads to an increase in the query response time for the common index schemes. Additionally, it can be prioritized according to the HS code, assuming that the dataset is being used by a company that mostly used specific goods. This dataset contains 99 files, 2,727,391 records, sized 128 Megabytes, each file is a chapter in the goods code. This dataset is prioritized according to the chapter of goods.

   It was supposed that this dataset was indexed for a company that is trading in toys and stationery goods, so usually their queries fetches items of these two types. Thus the stationery and toys records were prioritized, which are of chapters 95 and 48, i.e. records in these two chapters were given rank values.

3.3.2 Building the proposed index

Priority index is to be built in Oracle Database 12c, using Oracle SQL Developer Version 4.2.0.17.089. Index is to be built according to the priority of the records, and not a certain attribute, which speeds up a variety of queries. NYC Taxi dataset records would be prioritized by recent date and (CIMT) dataset records would be prioritized if goods are stationery or toys.

The records are to be sorted according to the priority of the records, and when a query is run, the query scans the records sequentially until it answers the query.

3.3.3 Baseline approaches

Bitmap and B-Tree indexes are the baseline approaches as they are the most common indexes that are used to evaluate other types of indexes; Bitmap and B-Tree indexes are already built in Oracle Database 12c, this research used the already built-in indexes as a baseline.

3.4 Experiment

Six experiments are to be conducted using multiple queries, to compare the performance of Priority index, Bitmap index, and B-Tree index. The comparison factors are the retrieval time and index size using the two datasets mentioned above in section 3.3.1.

3.4.1 Experimental design

Table 3.1: Experiments

Experiment	Dataset	Selectivity
E1	NYC dataset	0.000001
E2	NYC dataset	0.00001
E3	NYC dataset	0.0001
E4	CIMT dataset	0.000001
E5	CIMT dataset	0.00001
E6	CIMT dataset	0.0001

Table 3.2: Experiments using different selectivity on NYC dataset

Selectivity	0.000001	0.00001	0.0001
Priority Index			
B-Tree Index			
Bitmap Index			

Retrieval time average to be calculated from E1, E2, and E3

Table 3.3: Experiments using different selectivity on CIMT dataset

Selectivity	0.000001	0.00001	0.0001
Priority Index			
B-Tree Index			
Bitmap Index			

Retrieval time average to be calculated from E4, E5, and E6

Six experiments are to be conducted; three for each dataset each time the selectivity of the query will be changed - selectivity is the number of fetched records over the number of records -, selectivity would be 0.0001 in the first experiment, 0.00001 in the second experiment and 0.000001 in the third experiment. In these experiments the retrieval time of the 3 indexes will be compared. Retrieval time is affected by query selectivity and a good index should be fast even in high selectivity, it's noted that published work compare between indexes retrieval time in three different selectivity such as (Yu and Sarwat, 2016; Qin, 2016). Ten queries are to be run for each experiment; the queries are to be run three times, each time using a different index. The selectivity is to be changed in each experiment for each dataset, as shown in tables 3.2 and 3.3.

The objective of these experiments is to compute the average speed of retrieval time for each index scheme.

3.4.2 Experimental steps

1- Two tables are to be created using the NYC taxi dataset.

Table 3.4: Sample of the NYC dataset

VendorID	tpep_pickup_d	tpep_dropoff_dat	passenger	trip_dista	Ratecodel	store_and	PULocatio	DOLocatic	payment_	fare_amo	extra	mta_tax	tip_amou	tolls_amo	improvem	total_amo
1	01-04-17 0:00	01-04-17 0:15	1	1.8	1	N	158	113	2	10.5	0.5	0.5	0	0	0.3	11.8
1	01-04-17 0:00	01-04-17 0:16	3	3.7	1	N	87	158	2	14.5	0.5	0.5	0	0	0.3	15.8
2	01-04-17 0:00	01-04-17 10:20	1	0	1	N	264	193	1	0	0	0	0	0	0	0
1	01-04-17 0:00	01-04-17 0:37	2	3.3	1	N	230	4	1	23.5	0.5	0.5	4.95	0	0.3	29.75
1	01-04-17 0:00	01-04-17 0:02	1	0.7	1	N	142	143	2	4.5	0.5	0.5	0	0	0.3	5.8
1	01-04-17 0:00	01-04-17 0:07	1	2.2	1	N	170	263	1	8.5	0.5	0.5	2.45	0	0.3	12.25
1	01-04-17 0:00	01-04-17 0:08	1	0.7	1	N	100	48	1	7	0.5	0.5	1.65	0	0.3	9.95
1	01-04-17 0:00	01-04-17 0:02	1	0.4	1	N	166	166	2	4	0.5	0.5	0	0	0.3	5.3
2	01-04-17 0:00	01-04-17 0:10	1	1.23	1	N	113	68	1	8	0.5	0.5	1	0	0.3	10.3
2	05-04-17 7:52	05-04-17 7:55	1	0.55	1	N	68	68	1	4	0	0.5	0	0	0.3	4.8
2	26-04-17 15:45	26-04-17 16:10	6	3.72	1	N	88	246	1	18.5	1	0.5	3.04	0	0.3	25.29
2	26-04-17 16:12	26-04-17 16:28	6	2.29	1	N	246	114	1	12.5	1	0.5	1	0	0.3	15.3
2	26-04-17 16:32	26-04-17 23:46	6	1.81	1	N	113	170	2	10	1	0.5	0	0	0.3	11.8
2	01-04-17 0:00	01-04-17 0:46	1	5.67	1	N	162	261	2	33	0.5	0.5	0	0	0.3	34.3
2	01-04-17 0:00	01-04-17 0:31	1	18.21	2	N	132	234	1	52	0	0.5	14.64	5.76	0.3	73.2
2	01-04-17 0:00	01-04-17 1:19	6	7.27	1	N	100	52	1	50.5	0.5	0.5	12.95	0	0.3	64.75
2	01-04-17 0:00	01-04-17 0:06	6	1.06	1	N	230	142	1	6.5	0.5	0.5	1.56	0	0.3	9.36
1	01-04-17 0:00	01-04-17 0:09	2	1.8	1	N	181	61	2	9	0.5	0.5	0	0	0.3	10.3
2	01-04-17 0:00	01-04-17 0:12	1	1.71	1	N	170	79	1	9.5	0.5	0.5	2.16	0	0.3	12.96
2	01-04-17 0:00	01-04-17 0:26	1	3.08	1	N	144	25	1	18.5	0.5	0.5	3.96	0	0.3	23.76
1	01-04-17 0:00	01-04-17 0:18	2	3.6	1	N	164	140	1	15	0.5	0.5	3	0	0.3	19.3
1	01-04-17 0:00	01-04-17 0:18	1	3.5	1	N	211	97	1	15.5	0.5	0.5	3.36	0	0.3	20.16
2	01-04-17 0:00	01-04-17 0:24	5	4.31	1	N	113	141	1	18.5	0.5	0.5	3.96	0	0.3	23.76
2	01-04-17 0:00	01-04-17 0:13	5	3.54	1	N	75	116	1	13	0.5	0.5	2	0	0.3	16.3
1	01-04-17 0:00	01-04-17 0:20	1	5	1	N	125	263	1	17.5	0.5	0.5	3.75	0	0.3	22.55
1	01-04-17 0:00	01-04-17 0:35	3	10.3	1	N	234	228	1	34.5	0.5	0.5	8.3	5.76	0.3	49.86
1	01-04-17 0:00	01-04-17 0:05	2	0.7	1	N	161	162	2	5.5	0.5	0.5	0	0	0.3	6.8
2	01-04-17 0:00	01-04-17 0:05	1	0.6	1	N	79	113	1	5	0.5	0.5	1.89	0	0.3	8.19
2	01-04-17 0:00	01-04-17 0:08	1	0.7	1	N	148	232	1	7	0.5	0.5	1.66	0	0.3	9.96
2	01-04-17 0:00	01-04-17 0:18	1	3.79	1	N	142	79	1	15.5	0.5	0.5	5.04	0	0.3	21.84
2	01-04-17 0:00	01-04-17 0:10	4	1.93	1	N	158	48	2	9.5	0.5	0.5	0	0	0.3	10.8
2	01-04-17 0:00	01-04-17 0:05	3	1.41	1	N	234	161	1	6.5	0.5	0.5	1.56	0	0.3	9.36
2	01-04-17 0:00	01-04-17 0:11	1	2.02	1	N	79	233	1	10	0.5	0.5	2.26	0	0.3	13.56
2	01-04-17 0:00	01-04-17 0:06	1	0.96	1	N	48	186	2	6	0.5	0.5	0	0	0.3	7.3
2	01-04-17 0:00	01-04-17 0:20	1	3.38	1	N	79	255	1	16	0.5	0.5	3.46	0	0.3	20.76
1	01-04-17 0:00	01-04-17 0:15	1	4.7	1	N	264	264	1	15	0.5	0.5	2	0	0.3	18.3
2	01-04-17 0:00	01-04-17 0:07	1	1.15	1	N	230	186	1	6.5	0.5	0.5	1.56	0	0.3	9.36
2	01-04-17 0:00	01-04-17 0:06	1	1.3	1	N	100	50	2	6	0.5	0.5	0	0	0.3	7.3
2	01-04-17 0:00	01-04-17 0:02	2	0.53	1	N	141	263	1	4	0.5	0.5	1.06	0	0.3	6.36

2- Two tables to be created using CIMT dataset.

Table 3.4: Sample of the NYC dataset

HS_CODE	UOM_EXPORTS	COUNTRY	STATE	GEO	VALUE	QUANTITY	YEAR	MONTH
60490	N/A	9	1038	1	15368	0	2017	10
60490	N/A	9	1038	1	74216	0	2017	11
60490	N/A	9	1038	1	6101	0	2018	2
60490	N/A	9	1038	35	11627	0	2017	7
60490	N/A	9	1038	35	60535	0	2017	8
60490	N/A	9	1038	35	93795	0	2017	9
60490	N/A	9	1038	35	15368	0	2017	10
60490	N/A	9	1038	35	74216	0	2017	11
60490	N/A	9	1038	35	6101	0	2018	2
60490	N/A	9	1039	1	85335	0	2017	4
60490	N/A	9	1039	1	82925	0	2017	5
60490	N/A	9	1039	1	6602	0	2017	6
60490	N/A	9	1039	1	18195	0	2017	7
60490	N/A	9	1039	1	11143	0	2017	8
60490	N/A	9	1039	35	85335	0	2017	4
60490	N/A	9	1039	35	82925	0	2017	5
60490	N/A	9	1039	35	6602	0	2017	6
60490	N/A	9	1039	35	18195	0	2017	7
60490	N/A	9	1039	35	11143	0	2017	8
60490	N/A	9	1040	1	8209	0	2017	8
60490	N/A	9	1040	1	19370	0	2017	11
60490	N/A	9	1040	35	8209	0	2017	8
60490	N/A	9	1040	35	19370	0	2017	11
60490	N/A	9	1041	1	2901	0	2017	2
60490	N/A	9	1041	1	5019	0	2017	3
60490	N/A	9	1041	1	11518	0	2017	4
60490	N/A	9	1041	1	5669	0	2017	5
60490	N/A	9	1041	1	11745	0	2017	6
60490	N/A	9	1041	1	17826	0	2017	8
60490	N/A	9	1041	1	2696	0	2017	11
60490	N/A	9	1041	1	4505	0	2018	1

3- One table of each dataset is to be indexed by Priority index.

4- The other tables are to be indexed, first by B-Tree index, then by Bitmap index.

5- The size of each index is to be measured.
6- Different queries with different selectivity are to be run to compare the retrieval time of each index in different selectivity.

3.5 Evaluation method

In this research, the evaluation factors are the retrieval time, index size, and maintenance, as they are the factors that assess the performance of indexes. These factors were used to evaluate indexes in many researches. (Philip Chen and Zhang 2014, Qin 2016; Yu and Sarwat, 2016) used retrieval time, index size and maintenance to evaluate performance of indexes in big data, while (Wu et al., 2006) used retrieval time, and index size only to evaluate index performance. The accuracy of Priority Index query results is measured by comparing the results obtained using Priority Index by the results obtained using the other two indexes.

3.6 Summary

The main objective of this research is to develop an index scheme that speeds up query retrieval time while occupying small space with less maintenance needed. The listed methods in this chapter were used to achieve the objectives of this work. The methodology adopted in this research led to creation of an index that outperforms the common indexes, as the new index speeds up the retrieval time while occupying less space and needing less maintenance.

CHAPTER 4: PRIORITY INDEX DESIGN AND DEVELOPMENT

This chapter is dedicated to the design and development of Priority index.

4.1 Development Tools and Environments

In this research, the following tools and environments were used:

- Oracle VM Virtual Box Manager 5.2.8.
- Integrated Development Environment (IDE): Oracle SQL Developer Version 4.2.0.17.089.
- Database: Oracle Database 12c.
- Operating System: Oracle Linux Server 7.3.
- Processor: Intel® Core™ i7-3630QM CPU @ 2.40GHz.
- Memory: 3.9 GiB

4.2 Oracle SQL Developer

"Oracle SQL Developer is the Oracle Database IDE. A free graphical user interface, Oracle SQL Developer allows database users and administrators to do their database tasks in fewer clicks and keystrokes. A productivity tool, SQL Developer's main objective is to help the end user save time and maximize the return on investment in the Oracle Database technology stack.

SQL Developer supports Oracle Database 10g, 11g, and 12c, and will run on any operating system that supports Java," (Oracle 2004).

4.3 What is Priority Index?

Priority index is a covering index; it overcomes the limitations of the commonly used indexes. It is dedicated to the whole query and not to a significant attribute, which means it is always useful regardless of which fields are in or are not in the where clause. It also decreases the size of index as it only adds an attribute to the original table that keeps the necessary rank values. In addition, it increases the query retrieval time for the highly ranked records as they are scanned first, and being a covered index saves time as once a record is located it is returned immediately. The query engine does not need to look up the records, since all the requested columns are available within the index.

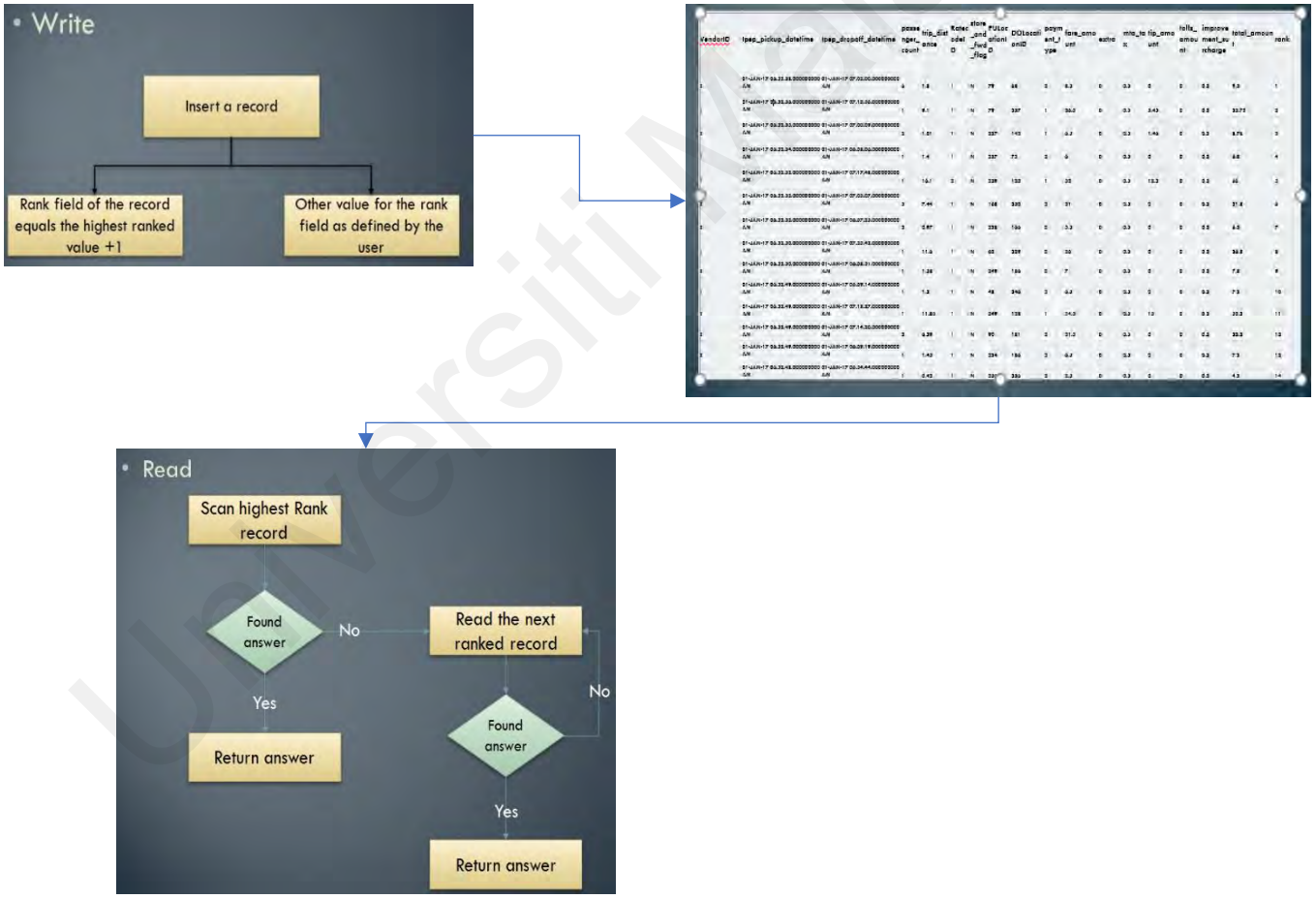


Figure 4.1: Read and Write operations in Priority index

4.4 When to use Priority Index?

This Priority index depends on the idea that not all data have the same importance, sometimes some data are never going to be used. Or at least some data are required to be retrieved fast, like the timely transactions, while other data are not needed in a timely manner.

Priority Index can be useful in the following cases:

- When having a huge table of data and the records are not of the same importance.
- When a big number of records are not queried in a big table.
- When there is a possibility to determine the criteria of the records that need to be queried in a fast manner.

Prioritization examples:

1- New data gets higher ranks, using a trigger that increases the value of the rank by one each time a record is added, in case the recent data needs to be fetched faster than the older data.

2- When special data is added, they get predefined ranks, using a trigger to insert a predefined value, in case there are some criteria and if they are met, the ranking of a record is calculated and inserted.

3- When a table has no more writes but is still used for query purpose, and we are not sure which are the important records and which are the useless ones, a suggestion is that when a field or fields are fetched by a query, a trigger increase their rank by one, so the more a record is queried the faster it will be retrieved.

4.5 Priority Index Operations

4.5.1 Write

Any number of records can be added easily without having to locate attributes or reconstruct the index. Records are added and if they need a value for rank, it is added either manually by the user or automatically by a predefined trigger, depending on the type of business.

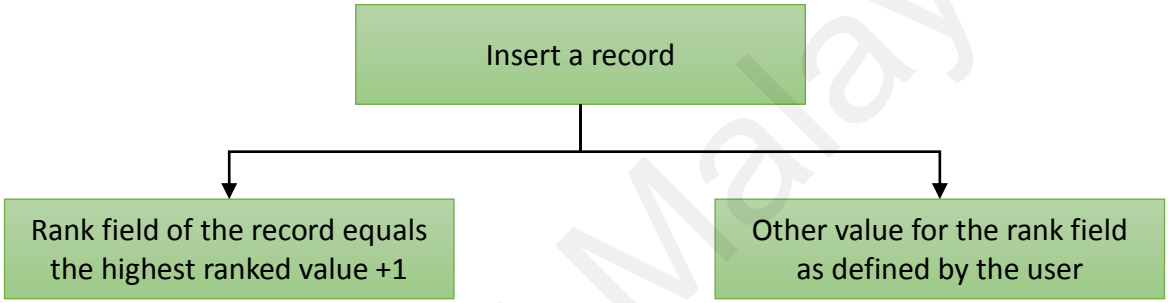


Figure 4.2: Write operation in Priority index

4.5.2 Maintenance

All types of index need maintenance, which is a cost on the database management system. The Priority index maintenance job is to reorder the records in the indexed table in a descending manner according to the value of the rank field.

The Priority index needs reordering, in case of adding new records (write) but it does not have to be immediate. Maintenance can be done at times when there are fewer transactions, or it can be scheduled according to the needs of the work. This is an advantage, as the write process will be fast, and as there will not be any type of reconstructing the index while writing. It also improves the performance of the database management system as maintenance can be done overnight after the day's work is over. Delaying the maintenance will not affect the accuracy of the returned

data, so it does not have to be in each write. In case of delete, there is no need to do any type of maintenance.

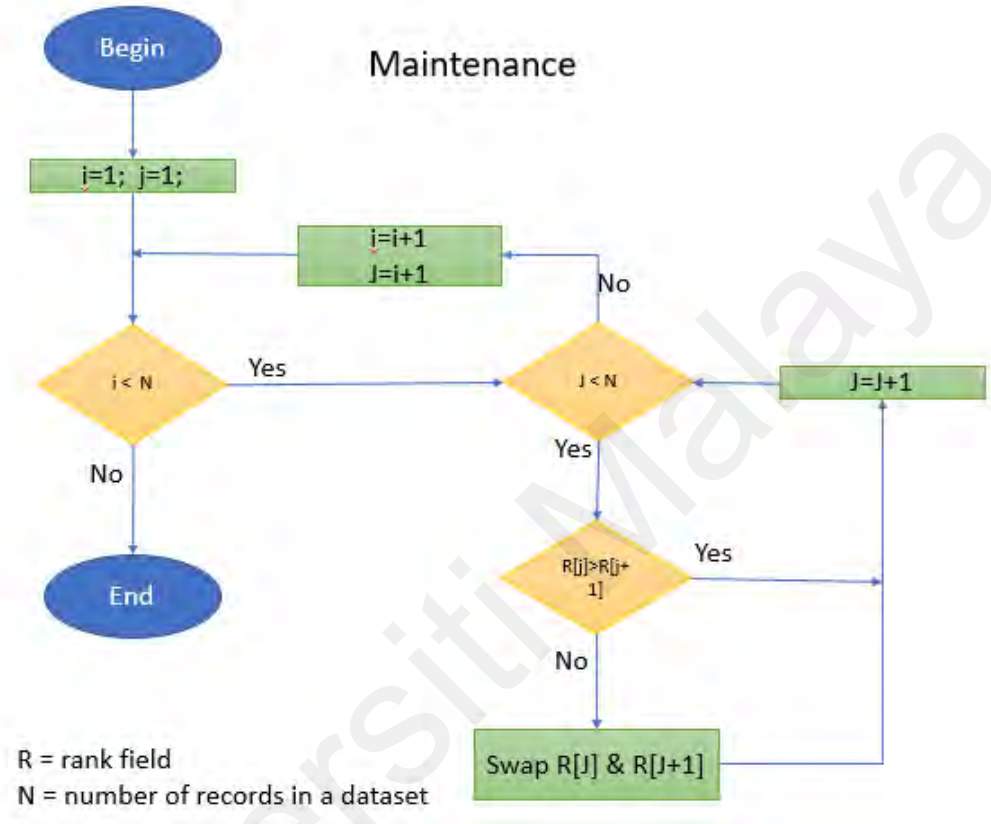


Figure 4.3: Maintenance operations in Priority index

4.5.3 Read

The data is going to be scanned sequentially; sequential read is faster than random read. As the maintenance job is to reorder the records in a descending manner according to the value of the rank field, the records with higher value for the rank attribute will be scanned first. The query will stop scanning the table when it is answered. For the read to be efficient, bounded query must be used. If not, the

reading process will scan the whole table which takes O(N) time, which is time consuming in the situation of big data.

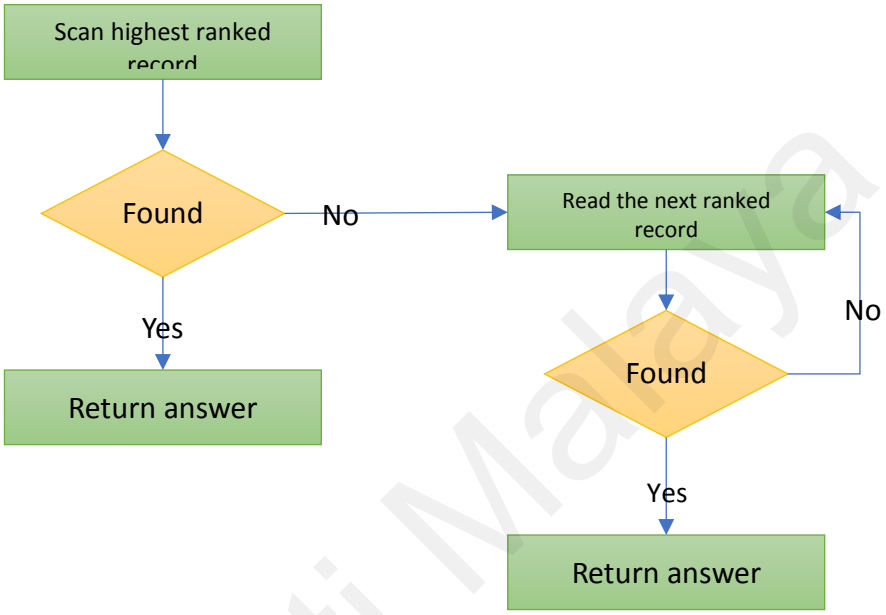


Figure 4.4: Read operations in Priority index

4.6 Data collection

A search for a suitable dataset was conducted for the proposed index and NYC Taxi and Limousine Commission (TLC) dataset was chosen. The dataset of yellow taxi was downloaded from the official web pages of the NYC Taxi and Limousine Commission. Twelve CSV files were downloaded, each file represents the ride information in a month in the year 2017 in NYC. The taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts. The data used in the datasets were collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology

providers authorized under the Taxicab & Livery Passenger Enhancement Programs (NYC).

Table 4.1: Data dictionary of NYC Yellow Taxi dataset (NYC 2018)

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. 1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
PULocationID	TLC Taxi Zone in which the taximeter was engaged
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged
RateCodeID	The final rate code in effect at the end of the trip. 1= Standard rate 2=JFK 3=Newark 4=Nassau or Westchester 5=Negotiated fare 6=Group ride
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server. Y= store and forward trip N= not a store and forward trip

Payment_type	A numeric code signifying how the passenger paid for the trip. 1= Credit card 2= Cash 3= No charge 4= Dispute 5= Unknown 6= Voided trip
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.
MTA_tax	$0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	$0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.
Tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.

4.7 Implementation

4.7.1 Table creation

Two tables were created using Oracle Sql Developer 'NYC', and 'NYC_PRIORITY'. Each table contains the following fields with the associated data type

"VendorID" NUMBER,

"tpep_pickup_datetime" TIMESTAMP (6),

"tpep_dropoff_datetime" TIMESTAMP (6),

"passenger_count" NUMBER(*,0),

"trip_distance" NUMBER,

"RatecodeID" NUMBER(*,0),

"store_and_fwd_flag" CHAR(1 BYTE),

"PULocationID" NUMBER,

"DOLocationID" NUMBER,

"payment_type" NUMBER(*,0),

"fare_amount" NUMBER,

"extra" NUMBER,

"mta_tax" NUMBER,

"tip_amount" NUMBER,

"tolls_amount" NUMBER,

"improvement_surcharge" NUMBER,

"total_amount" NUMBER.

‘NYC_PRIORITY’ has an extra field, which is ‘rank’, with data type number, for the purpose of Priority index.

4.7.2 Importing

4.7.2.1 Importing dataset files to ‘NYC’ table:

36,000,000 records from the NYC yellow taxi dataset were imported. The first 3,000,000 records in each file of the 12 datasets were imported to ‘NYC’ table, one file at a time creating a large table of 36,000,000 records, sized around 2.6 Gigabytes.

4.7.2.2 Importing dataset files to ‘NYC_PRIORITY’:

Records are imported to ‘NYC_PRIORITY’ table using an insert statement, which inserts all records from ‘NYC’ table that was created previously, and inserts the data to it. The insert statement sorts the records before inserting them; they are sorted according to pick up date and time value (tpep_pickup_datetime) attribute.

4.7.3 Indexing

- ‘NYC’ table was indexed first using B-Tree index. The index was on the attribute "tpep_pickup_datetime" as it is suitable for B-Tree index, as it is a high-cardinality column i.e. has variety of values.
- After running the needed queries on ‘NYC’ table and taking the needed measurements, B-Tree index was dropped and ‘NYC’ was indexed using Bitmap index. The index was on the attribute "tpep_pickup_datetime".
- ‘NYC_PRIORITY’ table was indexed using the Priority index described in the following.

4.6.3.1 Priority Index

As the chosen dataset is the data of taxi rides, The priority of this dataset was to be the date, i.e. the more recent the data the higher the rank. In most cases, there will not be a need to query an old ride in a timely manner. For example, if a problem happened during a ride, the data of that ride can be fetched in a timely manner. A passenger can appeal regarding a recent ride, or if a passenger reported something was lost or a problem happened during a ride, the information should be available in a timely manner. Data that is months old will not be needed in a timely manner.

As the records in NYC_PRIORITY table were already sorted, as the records were inserted into the table in a sorted manner, the remaining work was to fill the rank attribute.

The attribute rank was filled using a code that inserts values, starting with a value equal to the number of records, and it subtracts 1 as it fills the previous records. So the rank field will be filled with numbers, starting by the number of records, and subtracting one for each record until it reaches the last record with a value of zero.

Table 4.2: Part of table NYC_PRIORITY

VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	fare_amount	extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_amount	rank
2	01-JAN-17 06.53.58	01-JAN-17 07.02.00	6	1.8	1	N	79	68	2	8.5	0	0.5	0	0	0.3	9.3	17
1	01-JAN-17 06.53.56	01-JAN-17 07.12.56	1	9.1	1	N	79	257	1	26.5	0	0.5	5.45	0	0.3	32.75	16
2	01-JAN-17 06.53.55	01-JAN-17 07.00.09	3	1.01	1	N	237	142	1	6.5	0	0.5	1.46	0	0.3	8.76	15
1	01-JAN-17 06.53.54	01-JAN-17 06.58.06	1	1.4	1	N	237	75	2	6	0	0.5	0	0	0.3	6.8	14
1	01-JAN-17 06.53.53	01-JAN-17 07.17.48	1	16.1	2	N	229	132	1	52	0	0.5	13.2	0	0.3	66	13
2	01-JAN-17 06.53.53	01-JAN-17 07.05.07	5	7.44	1	N	168	250	2	21	0	0.5	0	0	0.3	21.8	12
2	01-JAN-17 06.53.53	01-JAN-17 06.57.35	2	0.97	1	N	238	166	2	5.5	0	0.5	0	0	0.3	6.3	11
1	01-JAN-17 06.53.50	01-JAN-17 07.25.43	1	11.6	1	N	62	239	2	36	0	0.5	0	0	0.3	36.8	10
2	01-JAN-17 06.53.50	01-JAN-17 06.58.51	1	1.58	1	N	249	186	2	7	0	0.5	0	0	0.3	7.8	9
1	01-JAN-17 06.53.49	01-JAN-17 06.59.14	1	1.2	1	N	48	246	2	6.5	0	0.5	0	0	0.3	7.3	8
2	01-JAN-17 06.53.49	01-JAN-17 07.18.37	1	11.86	1	N	249	138	1	34.5	0	0.5	15	0	0.3	50.3	7
2	01-JAN-17 06.53.49	01-JAN-17 07.14.30	2	6.29	1	N	90	181	2	21.5	0	0.5	0	0	0.3	22.3	6
2	01-JAN-17 06.53.49	01-JAN-17 06.59.19	1	1.45	1	N	234	186	2	6.5	0	0.5	0	0	0.3	7.3	5
2	01-JAN-17 06.53.48	01-JAN-17 06.54.44	1	0.42	1	N	255	256	2	3.5	0	0.5	0	0	0.3	4.3	4
1	01-JAN-17 06.53.45	01-JAN-17 06.56.43	1	1	1	N	236	262	1	5	0	0.5	1	0	0.3	6.8	3
2	01-JAN-17 06.53.45	01-JAN-17 07.10.08	5	3.84	1	N	48	75	2	15	0	0.5	0	0	0.3	15.8	2
2	01-JAN-17 06.53.43	01-JAN-17 07.02.22	1	1.99	1	N	37	255	2	8.5	0	0.5	0	0	0.3	9.3	1

4.8 Implementing Priority index for a second example

4.8.1 Dataset

The Canadian International Merchandise Trade (CIMT) online database offers detailed trade data using the Harmonized System (HS) classification of goods (based on the 6-digit commodity level) ((CIMT) 2017 -2018).

This dataset contains nine columns (HS_CODE, UOM_EXPORTS, COUNTRY, STATE, GEO, VALUE, QUANTITY, YEAR, MONTH).

4.8.2 Table creation

Two tables were created using Oracle Sql Developer HR.CANADA_TRADE_PR, and HR.CANADA_TRADE. Each table contains the following fields with the associated data type

HS_CODE NUMBER(6,0),
UOM_EXPORTS VARCHAR2(5 BYTE),
COUNTRY NUMBER(6,0),
STATE_NO NUMBER(6,0),
GEO NUMBER(3,0),
AMOUNT NUMBER(9,0),
QUANTITY NUMBER(9,0),
"YEAR" NUMBER(4,0),
"MONTH" NUMBER(2,0),
“RANK” NUMBER(6)

4.8.3 Importing

Data of The Canadian International Merchandise Trade (CIMT) was imported to HR.CANADA_TRADE table using oracle sql developer, 99 files were imported one file at a time to create a table of 2,727,391 records, sized 128 Megabytes.

4.8.4 Inserting the rank value

It was supposed that the company that will use the Priority index is a company that is interested in stationery and toys. The company will rank all the toys goods with one, and rank the stationery goods with two. We used the following code:

DECLARE

```
i number;
l number;
BEGIN
select count (*) INTO l from HR.CANADA_TRADE_PR;
 FOR i IN 1..l
 LOOP
UPDATE HR.CANADA_TRADE_PR
  SET "RANK" = 2
  where HR.CANADA_TRADE_PR.HS_CODE LIKE '95%';
  UPDATE HR.CANADA_TRADE_PR
  SET "RANK" = 1
  WHERE HR.CANADA_TRADE_PR.HS_CODE LIKE '48%';
 END LOOP;
END;
```

4.8.5 Inserting dataset files to HR.CANADA_TRADE_PR

Records are imported to 'HR.CANADA_TRADE_PR' table using an insert statement that inserts all records from 'HR.CANADA_TRADE' table that we created previously and inserts the data into it. The insert statement sorts the records before inserting; they are sorted according to rank field.

4.8.6 Indexing

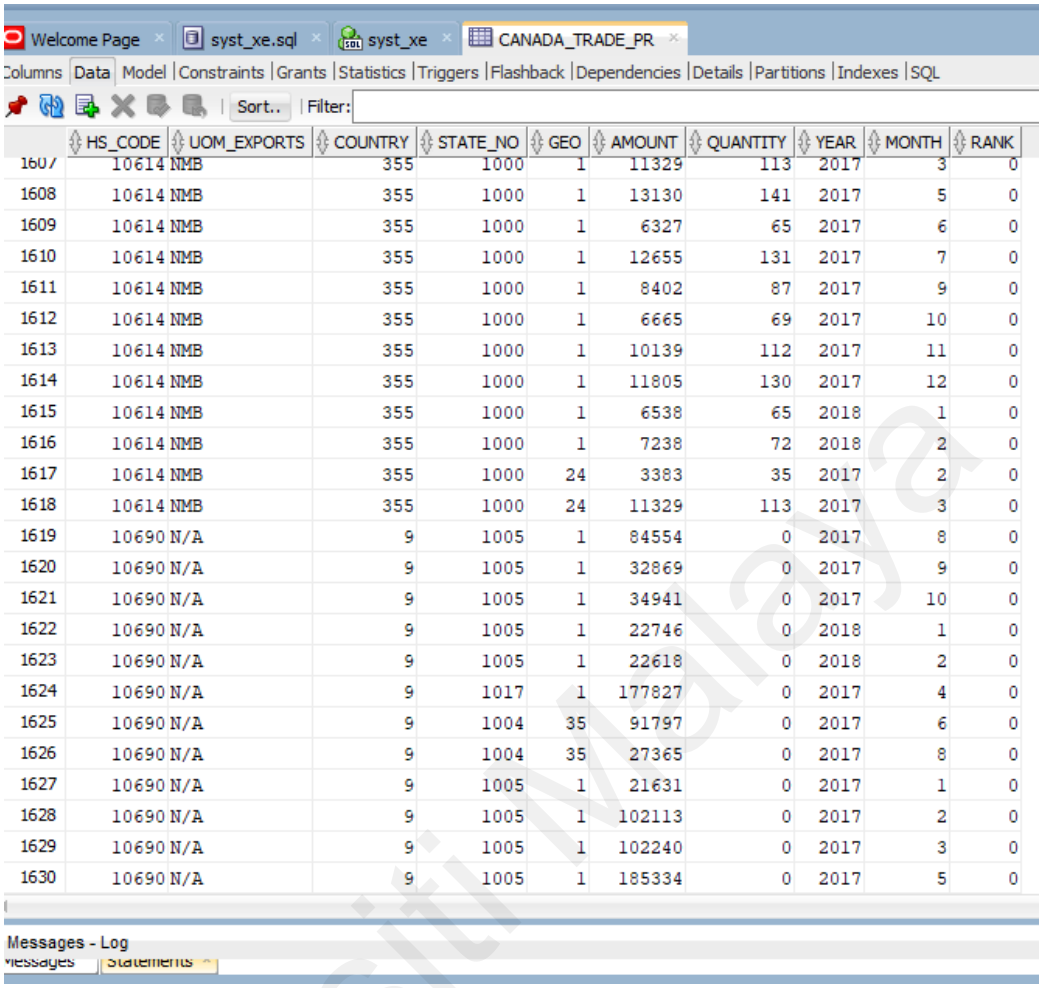
- HR.CANADA_TRADE table was indexed first using B-Tree index, the required queries were run against the table and all needed measurements were taken. Then this index was dropped and the table was indexed by Bitmap index.

- HR.CANADA_TRADE_PR table was indexed using the Priority index described in the following.

4.6.3.1 Priority Index

As the dataset is the detailed trade data, it was supposed that the company that will use this dataset is a company that works with stationery and toys, so the company is mostly interested in goods that codes start with 95 and 48.

As Rank value was already set in HR.CANADA_TRADE table using the code above, the data was inserted into table HR.CANADA_TRADE_PR using an insert statement that inserts records from HR.CANADA_TRADE to HR.CANADA_TRADE_PR ordered by Rank.



	HS_CODE	UOM_EXPORTS	COUNTRY	STATE_NO	GEO	AMOUNT	QUANTITY	YEAR	MONTH	RANK
1607	10614	NMB	355	1000	1	11329	113	2017	3	0
1608	10614	NMB	355	1000	1	13130	141	2017	5	0
1609	10614	NMB	355	1000	1	6327	65	2017	6	0
1610	10614	NMB	355	1000	1	12655	131	2017	7	0
1611	10614	NMB	355	1000	1	8402	87	2017	9	0
1612	10614	NMB	355	1000	1	6665	69	2017	10	0
1613	10614	NMB	355	1000	1	10139	112	2017	11	0
1614	10614	NMB	355	1000	1	11805	130	2017	12	0
1615	10614	NMB	355	1000	1	6538	65	2018	1	0
1616	10614	NMB	355	1000	1	7238	72	2018	2	0
1617	10614	NMB	355	1000	24	3383	35	2017	2	0
1618	10614	NMB	355	1000	24	11329	113	2017	3	0
1619	10690	N/A	9	1005	1	84554	0	2017	8	0
1620	10690	N/A	9	1005	1	32869	0	2017	9	0
1621	10690	N/A	9	1005	1	34941	0	2017	10	0
1622	10690	N/A	9	1005	1	22746	0	2018	1	0
1623	10690	N/A	9	1005	1	22618	0	2018	2	0
1624	10690	N/A	9	1017	1	177827	0	2017	4	0
1625	10690	N/A	9	1004	35	91797	0	2017	6	0
1626	10690	N/A	9	1004	35	27365	0	2017	8	0
1627	10690	N/A	9	1005	1	21631	0	2017	1	0
1628	10690	N/A	9	1005	1	102113	0	2017	2	0
1629	10690	N/A	9	1005	1	102240	0	2017	3	0
1630	10690	N/A	9	1005	1	185334	0	2017	5	0

Figure 4.5: Part of table CANADA_TRADE_PR

4.9 Summary

This chapter explains the process of developing the Priority Index. The datasets were imported to the database and records, were ranked, and sorted according to their importance. In read operation, records are scanned sequentially, so the important data is scanned first. Maintenance for Priority index is not needed in each operation. It can be scheduled to be done at any preferred time. Tables in the database were indexed by Priority Index, B-Tree Index, and Bitmap index.

CHAPTER 5: EXPERIMENTS, EVALUATION AND RESULTS

This chapter illustrates the experiments conducted and the evaluation of Priority index in terms of retrieval time and space occupied. It compares the Priority index with Bitmap index and B-Tree index. In addition, it discusses the obtained results.

5.1 Experiments and Evaluation

The indexed tables created in chapter 4 are used in 6 experiments to calculate the average retrieval time for each index scheme.

In this research, the evaluation factors are retrieval time, index size, and maintenance, to measure the performance of the proposed and baseline approaches in accelerating the retrieval time in big data.

5.1.1 Retrieval Time

To compare the retrieval time for Bitmap index and B-Tree index, a field to build the index on should be the same; a field with higher selectivity increases the index retrieval time, i.e. high-cardinality attribute.

- Selectivity = number of distinct values over number of records.
- "tpep_pickup_datetime" field was used to create the Bitmap and B-Tree index as this field has a high variety of values, which means it has high selectivity, thus making it a good choice for speeding up the retrieval time in index. The selectivity is 9095084/36000000 = 0.25

```
SELECT COUNT(*) FROM HR.NYC;
 -- The result of the statement is 36000000


SELECT COUNT( DISTINCT(TPEP_PICKUP_DATETIME)) FROM HR.NYC;
--The result of the statement is 9095084
```

Figure 5.1: Queries to get the needed information to calculate selectivity for NYC dataset

- "hs-code" field was used to create the Bitmap and B-Tree index as this field has a variety of values which means it has high selectivity to increase the query response time. The selectivity of index is 4830/2727391 = 0.002

```
SELECT COUNT( DISTINCT (HS_CODE)) FROM HR.CANADA_TRADE;
 -- The result: 4830


SELECT COUNT(*) FROM HR.CANADA_TRADE;
--The result of the statement is 2727391
```

Figure 5.2: Queries to obtain the needed information to calculate selectivity for CIMT dataset

5.1.2 Index size

The following queries are used to determine the size of the Bitmap and B-Tree indexes.

```
SELECT BYTES, SEGMENT_NAME FROM USER_SEGMENTS
WHERE SEGMENT_NAME = 'NYC_BTREE_IDX';

SELECT BYTES, SEGMENT_NAME FROM USER_SEGMENTS
WHERE SEGMENT_NAME = 'NYC_BITMAP_IDX';

SELECT BYTES, SEGMENT_NAME FROM USER_SEGMENTS
WHERE SEGMENT_NAME = 'CIMT_BTREE_IDX';

SELECT BYTES, SEGMENT_NAME FROM USER_SEGMENTS
WHERE SEGMENT_NAME = 'CIMT_BITMAP_IDX';
```

Figure 5.3: Queries to determine the size of Bitmap and B-Tree indexes

The size of Priority index is calculated by calculating the size of the rank attribute which were added for the sake of indexing using Priority index. The size of the Priority index is the size of the table, with the rank attribute minus the size of the original table.

5.1.3 Maintenance:

Maintenance was not measured in experiments as it is not done in each insert or delete operation in Priority Index.

5.2 Results

5.2.1 Index Size

By running queries in figure 5.3 to calculate the size of Bitmap index and B-Tree index, and by calculating the size of rank attribute to get the size of Priority index, the following results were found.

Table 5.1: Comparison between the indexes in terms of index size

Index\ Dataset	NYC Taxi	CIMT
Priority	171 Megabytes	0.09412 Megabytes
B-Tree	752 Megabytes	150 Megabytes
Bitmap	336 Megabytes	80 Megabytes

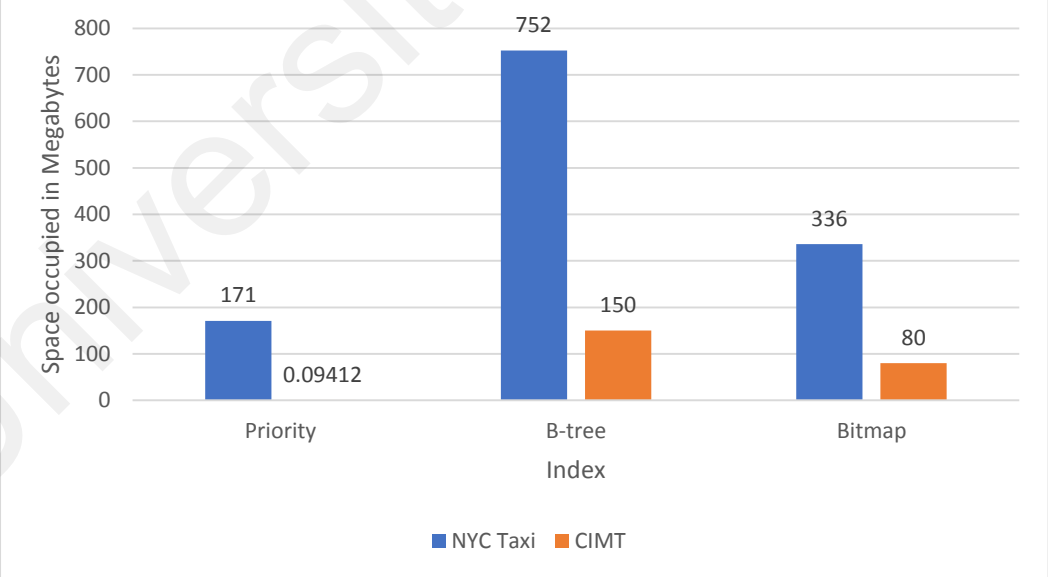


Figure 5.4: A comparison between the indexes (Priority, B-Tree and Bitmap) in terms of index size

Table 5.1 and Figure 5.4 compares between Priority index, B-Tree index and Bitmap index in terms of index size. It is noted that Priority index is small in size compared to B-Tree and Bitmap index. The size of Priority index is the size of the rank field only. Which means the size of the index will always be smaller than Bitmap and B-Tree index even if the dataset contained huge number of records.

It is also noticeable that Priority index size is very small for CIMT dataset and the reason is that there are only 2 values inserted to the rank column, which are "1" and "2".

20681 records have the value 1 and 78006 records have the value 2. Value 1 and 2 occupy 1 byte only, i.e. the size of Priority index for CIMT dataset is 20681 bytes + 78006 bytes. Even if the records are increased in this dataset, the size of the index will only increase by one byte, if the inserted record is of high priority. If the inserted records are not of high priority, there will be no effect in the size of the Priority index.

5.2.2 Retrieval Time

Queries with different selectivity were used to measure the query response time. Selectivity here is the number of fetched records over the number of records, as shown in figure 5.5, and 5.6 respectively.

```
SELECT FARE_AMOUNT FROM HR.NYC WHERE TPEP_PICKUP_DATETIME > '01-NOV-17 06.53.43.000000000 AM' AND ROWNUM <= 3,600;

-- Selectivity is 0.0001

SELECT FARE_AMOUNT FROM HR.NYC_PRIORITY WHERE "TPEP_PICKUP_DATETIME" > '01-DEC-17 06.53.43.000000000 AM' AND ROWNUM <= 360;

-- Selectivity is 0.00001

SELECT FARE_AMOUNT FROM HR.NYC WHERE TPEP_PICKUP_DATETIME > '01-DEC-17 06.53.43.000000000 AM' AND ROWNUM <= 36;

-- Selectivity is 0.000001
```

Figure 5.5: Example of the used queries in the experiment for NYC dataset

```
SELECT * FROM (SELECT AMOUNT FROM HR.CANADA_TRADE_PR WHERE (HS_CODE BETWEEN 480000 AND 489999) OR (HS_CODE BETWEEN 950000 AND 959999)) WHERE ROWNUM <= 272;

-- Selectivity is 0.0001


SELECT * FROM (SELECT AMOUNT FROM HR.CANADA_TRADE WHERE (HS_CODE BETWEEN 480000 AND 489999) OR (HS_CODE BETWEEN 950000 AND 959999)) WHERE ROWNUM <= 27;

-- Selectivity IS 0.00001


SELECT * FROM (SELECT AMOUNT FROM HR.CANADA_TRADE_PR WHERE HS_CODE BETWEEN 480000 AND 489999) WHERE ROWNUM <= 2;

-- Selectivity is 0.000001
```

Figure 5.6 Example of the used queries in the experiment for CIMT dataset

The outputs of the queries were the same for all the three indexes when using the same query. The differences were in the retrieval time which means that the proposed index query results are accurate.

Table 5.2: NYC taxi dataset response time

Index \ Selectivity	0.000001	0.00001	0.0001
Priority	0.0152	0.068	0.778
B-Tree	0.245	0.262	0.607
Bitmap	0.239	0.265	0.642

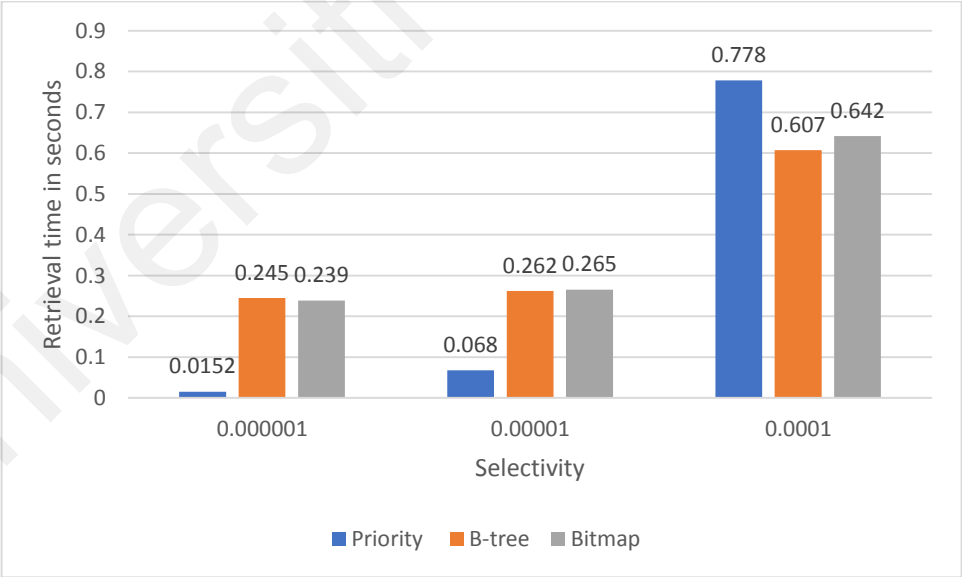


Figure 5.7: NYC taxi dataset response time

Table 5.2 and figure 5.7 illustrate a comparison between Priority index, B-Tree index and Bitmap index in terms of query retrieval time for NYC taxi dataset in selectivity 0.000001, 0.00001 and 0.0001. Retrieval time is measured in seconds.

Table 5.3: CIMT dataset response time

Index \ Selectivity	0.000001	0.00001	0.0001
Priority	0.0101	0.05	0.573
B-Tree	0.25	0.301	0.67
Bitmap	0.29	0.3	0.579

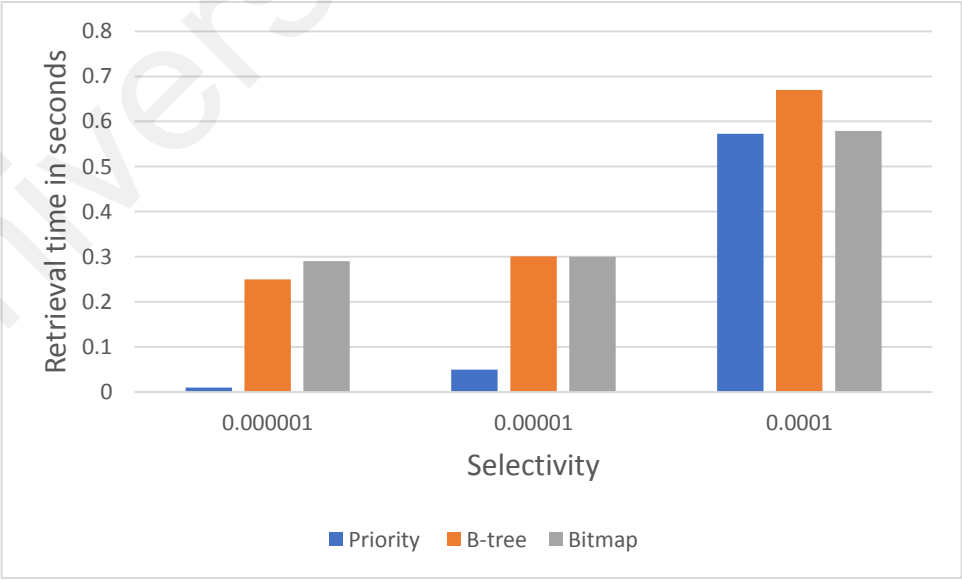


Figure 5.8: CIMT dataset retrieval time

Table 5.3 and figure 5.8 illustrate a comparison between Priority index, B-Tree index and Bitmap index in terms of query retrieval time for CIMT dataset in selectivity 0.000001, 0.00001 and 0.0001. Retrieval time is measured in seconds.

5.3 Discussion

5.3.1 Size

Priority index occupy a small size compared to Bitmap index and B-Tree index. It is known that the common indexes increase the table size from 5% to 15% (Yu and Sarwat 2016) which is a big space when data is in Gigabytes; on the other hand, Priority index only adds to the original size of a table, size of a column of a number datatype.

5.3.2 Retrieval time

It is noticed that Priority index is faster in retrieving records compared to Bitmap and B-Tree index, especially in the small selectivity. Priority index retrieval time is not affected by the size of the dataset as long as the required records are among the high priority records; Priority index starts scanning the important records, so it reaches the important records in a timely manner regardless of the size of the dataset.

5.3.3 Maintenance

Maintenance in Priority index can be scheduled and does not have to be done after each operation; on the other hand, maintenance in the common indexes has to be done after each operation, which is a high cost to the operating system.

5.4 Summary

In this chapter, the performance of the Priority index was evaluated in terms of retrieval time and space occupied compared to B-Tree index and Bitmap index. It was obvious that Priority index occupies a small space compared to the speed of retrieval time. It is also noted that Priority index does not increase much in size when the dataset is increasing, but, Priority retrieval time slows down when selectivity is high, which makes Priority index a good fit when querying big data and trying to retrieve small size of important data.

CHAPTER 6: CONCLUSION

This chapter summarizes the major findings of this research, towards the development of an index scheme. This chapter revisits the objectives of this research, as well as the steps taken to achieve those objectives. Furthermore, this chapter also discusses the contributions and work limitations, as well as suggestions that can be carried out in future.

6.1 State of the art

Speeding up retrieval time is important to system applications as operations can be faster and reports can be timely generated. Speeding up query retrieval comes with a cost to Database Management Systems. Using index is a common solution to speed up retrieval time, but an index comes with a price of extra space and extra maintenance, a faster index is a bigger index in size. There is a need to have an index that increases the speed of retrieval time without leading to much increment in size and maintenance.

6.2 Research objectives revisited

6.2.1 First objective

The first objective was to develop a suitable index scheme approach to accelerate the query retrieval time in big data. This objective was met by developing the Priority index approach. The performance of Priority index was analyzed, and it was clear that Priority index is quicker in retrieval time than Bitmap and B-Tree indexes, especially when retrieving small amount of high priority records.

6.2.3 Second objective

The second objective was to evaluate the proposed approach with the common available approaches in terms of retrieval time and space occupied. This objective was achieved by conducting experiments using two different datasets that were indexed using Bitmap index, B-Tree index and Priority index. The sizes of the different indexes were measured. And the retrieval time were calculated using different selectivity. The results showed that Priority index outperforms the other index schemes in terms of space occupied and retrieval time.

6.3 Contribution

The main contribution of this research is the design of the Priority index, which outperforms B-Tree and Bitmap index in retrieval time and space occupied. Priority index occupies a smaller size compared to the speed of query operation, in addition, Priority Index is not affected by cardinality. The maintenance of the Priority index can be scheduled and does not have to be in each write and delete operation, as this does not affect the accuracy of the retrieved data.

6.4 Interpretations of Results and Insights

- Priority index depends on the whole record and not on a single attribute, i.e. it is always useful regardless of what the where clause includes.
- Priority index depends on ranking the records of a dataset, and its retrieval time is small when the high ranked records are queried. It is suitable for indexing big data when the records are not of the same importance, especially in low selectivity.

- The reason behind the high speed of query operation in Priority index is that the important data are scanned first, so when the query result is among the highly ranked data, it is retrieved in a very short time. In addition, Priority index is a covering index, i.e. once a record is located it is returned immediately.
- Priority index needs less maintenance, as it can be scheduled and does not have to be in each write or delete.
- Priority index performance is not affected by cardinality, whether the dataset contains high or low cardinality attributes that makes no change in the performance of Priority Index.

6.5 Limitation of work

The limitation of the Priority index is that it is useful in bounded query only, i.e. if the query is not a bounded query, the query will scan the whole dataset, which means a slow retrieval time.

The used database is the oracle express version, which has limitation in size as it is a free database, and this is the reason behind not creating huge sized tables.

6.6 Recommendations for future works

A recommendation for future work that can be done using the Priority index approach is to create a module that uses the Priority index to index big datasets, and develop an approach that helps in building bounded queries instead of using unbounded queries, i.e. a module that has the ability to convert queries to bounded queries, so the index can answer any query in high speed.

References

(CIMT), C. I. M. T. D. (2017 -2018). "Canadian International Merchandise Trade Database (CIMT)." Retrieved 15-08-2018, 2018, from https://open.canada.ca/data/en/dataset/b1126a07-fd85-4d56-8395-143aba1747a4.

Adamu, F. B., et al. (2015). "A Survey On Big Data Indexing Strategies." SLAC-PUB **6 pages** (December 2015): 6.

Alvarez, V., et al. (2015). A comparison of adaptive radix trees and hash tables. 2015 IEEE 31st International Conference on Data Engineering.

Bebensee, T., et al. (2010). Binary Priority List for Prioritizing Software Requirements, Berlin, Heidelberg, Springer Berlin Heidelberg.

Bellamkonda, S., et al. (2013). "Adaptive and big data scale parallel execution in oracle." Proc. VLDB Endow. **6**(11): 1102-1113.

Bounie, D. and L. Gille (2012). Info Capacity| International Production and Dissemination of Information: Results, Methodological Issues and Statistical Perspectives.

Chong, G., et al. (2016). Accelerate bitmap indexing construction with massive scientific data. 2016 5th International Conference on Computer Science and Network Technology (ICCSNT).

Durham, E. E. A., et al. (2014). A model architecture for Big Data applications using relational databases. 2014 IEEE International Conference on Big Data (Big Data).

Fan, W., et al. (2015). Querying Big Data by Accessing Small Data. Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. Melbourne, Victoria, Australia, ACM**:** 173-184.

Fasolin, K., et al. (2013). Efficient Execution of Conjunctive Complex Queries on Big Multimedia Databases. 2013 IEEE International Symposium on Multimedia.

Fusco, F., et al. (2010). "NET-FLi: on-the-fly compression, archiving and indexing of streaming network traffic." Proc. VLDB Endow. **3**(1-2): 1382-1393.

Garg, D. and A. Datta (2012). Test Case Prioritization Due to Database Changes in Web Applications. 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation.

Geerts, F. (2016). "Scale Independence: Using Small Data to Answer Queries on Big Data." 61.

Goldsmith, M. R., et al. (2014). "Development of a consumer product ingredient database for chemical exposure screening and prioritization." Food and Chemical Toxicology **65**: 269-279.

Goldstein, J., et al. (1998). Compressing relations and indexes. Proceedings 14th International Conference on Data Engineering.

Hougaard, R., et al. (2016). Priorities. One Second Ahead: Enhance Your Performance at Work with Mindfulness. New York, Palgrave Macmillan US**:** 39-46.

Kaushik, R., et al. (2002). Covering indexes for branching path queries. Proceedings of the 2002 ACM SIGMOD international conference on Management of data. Madison, Wisconsin, ACM**:** 133-144.

Marr, B. (2015). "A brief history of big data everyone should read." Retrieved 22/07/2018, 2018, from https://www.weforum.org/agenda/2015/02/a-brief-history-of-big-data-everyone-should-read/.

Narayanan, S. and F. Waas (2011). Dynamic prioritization of database queries. 2011 IEEE 27th International Conference on Data Engineering.

NYC. "Taxi trips." Retrieved 25-07-2018, 2018, from http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.

NYC (2018). "NYC data dictionary." Retrieved 18/07/2018, 2018, from http://www.nyc.gov/html/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf.

Oracle (2004). "What is SQL Developer?". Retrieved 18/07/2018, 2018, from http://www.oracle.com/technetwork/developer-tools/sql-developer/what-is-sqldev-093866.html.

Philip Chen, C. L. and C.-Y. Zhang (2014). "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data." Information Sciences **275**: 314-347.

Pottinger, R. and A. Halevy (2001). "MiniCon: A scalable algorithm for answering queries using views." Vldb Journal **10**(2-3): 182-198.

Qin, X. (2016). Performance comparison of index schemes for range query of big data. 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD).

Sharma, A. and M. Sood (2014). Utilizing materialized views to formulate business intelligence. 2014 International Conference on Parallel, Distributed and Grid Computing.

Sohrabi, M. K. and H. Azgomi (2017). "TSGV: a table-like structure-based greedy method for materialized view selection in data warehouses." Turkish Journal of Electrical Engineering and Computer Sciences **25**(4): 3175-3187.

Sohrabi, M. K. and V. Ghods (2016). "Materialized View Selection for a Data Warehouse Using Frequent Itemset Mining." Journal of Computers **11**(2): 140-148.

Wu, H., et al. (2017). A Performance-Improved and Storage-Efficient Secondary Index for Big Data Processing. 2017 IEEE International Conference on Smart Cloud (SmartCloud).

Wu, K., et al. (2006). "Optimizing bitmap indices with efficient compression." ACM Trans. Database Syst. **31**(1): 1-38.

Yu, J. and M. Sarwat (2016). "Two birds, one stone: a fast, yet lightweight, indexing scheme for modern database systems." Proc. VLDB Endow. **10**(4): 385-396.

Zhou, J., et al. (2007). Lazy maintenance of materialized views. Proceedings of the 33rd international conference on Very large data bases. Vienna, Austria, VLDB Endowment**:** 231-242.