IMPROVED STORAGE TECHNIQUE FOR METADATA MANAGEMENT AT THE APPLICATION LEVEL

A Thesis by

Sriharsha Dhanekula

B.Tech, DRK Institute of Science and Technology, JNTU 2009

Submitted to the Department of Electrical Engineering and Computer Science

and the faculty of the Graduate School of

Wichita State University

in partial fulfillment of

the requirements for the degree of

Master of Science

December 2012

IMPROVED STORAGE TECHNIQUE FOR METADATA MANAGEMENT AT THE APPLICATION LEVEL

The following faculty members have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Computer Networking.

Ravi Pendse, Committee Chair

John Watkins, Committee Member

Bob Minaie, Committee Member

DEDICATION

To my dad, Sambasiva Rao Dhanekula (Late).

ACKNOWLEDGEMENTS

I would like to wholeheartedly thank my advisor and professor Dr. Ravi Pendse for his invaluable insight, guidance, and constant encouragement throughout not only my thesis work but my studies at Wichita State University. I would also like to extend my appreciation to Mrs. Vidya Suryanarayana for her guidance, advice, and time. I would like to thank my committee members Dr. John Watkins and Dr. Bob Minaie for their precious time and suggestions. Thank you to my leads Amarnath Jasti and Vijay Ragothaman for all their support during my tenure at work. Finally, I would also like to thank God, my family, and friends for their constant encouragement and support during my research.

ABSTRACT

Documentation has a become part of everyday life for individuals in the corporate world. With the increase in required documentation, the size of the data is also growing, which requires maintenance and storage facilities. Due to the growth in data, metadata, data about data, has been created. Metadata helps users recover relevant data easily and is used throughout the industry. Though it may take up less space, metadata is also increasing to keep up with the needs of companies, which means additional costs are accumulating. Data can be classified into two types: active data and archived data. Active data is the data which is frequently used while archived data is the data which is used irregularly or infrequently. In order to improve database space management, this thesis proposes a mechanism where the metadata associated with low priority data is stored in a compressed format using Burrows-Wheeler's transform compression algorithm.

TABLE OF CONTENTS

TABLE OF CONTENTS (Contd..)

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Metadata is generally defined as data about data or information about information. Metadata is an organized form of information that describes, locates, or explains data. In addition, it helps the user easily retrieve, use, and manage the resource. Thus metadata helps to find or understand the data before accessing it. Metadata has been in use for quite some time. In the past it has been used in libraries in the form of indexing for books and as catalogs for the different documents. Based on the catalogs, a document in a huge library can be searched for and located easily. Metadata is also useful in museums for the description of various items on display. In general, the uses of metadata can fall under one of the following categories: resource discovery, organizing electronic resources, interoperability, digital identification, and archiving and preservation. Classification of metadata has been done based different concepts. Amongst them we can classify metadata as content-dependent or content-independent. Respective examples for the above classification are size in case and location. One more type of classification is based on purpose of usage of metadata. This includes resource discovery and document indexing.

1.2 Standards of Metadata

In order to have complete understanding and interoperability between various devices or machines, standards must be defined. Numerous independent bodies have been established at both the national and international level to set standards for various aspects in assorted computer fields.

1.2.1 Dublin Core

The international metadata standard is Dublin Core (DC). It has been by developed by ISO. This standard defines metadata as a simple and concise element set that enables professionals from different domains to describe their content in an interoperable manner. It has 15 core elements to describe the content: title, creator, subject, description, publisher, contributor, date, format, identifier, source, language, relation, coverage, and rights [1]. In order to make the usage understandable for all users, the rules have been kept very simple. These elements in Dublin core are all optional, repeatable and can be presented in any order [2].

Dublin Core has made improvements by introducing a set of qualifiers which was followed by controlled vocabularies. These improvements were made by narrowing down a few of the 15 core elements such as “table of contents,” a narrowing of “description;” and “issued,” narrowing “date.”Because of these improvements, it is known as “Quality DC.” A number of encoding schemes have been identified and registered under the Dublin Core Metadata Initiative (DCMI) community. These schemes were employed in order to control the values that are being entered in a metadata statement. One example of this is that english is represented by “eng.”. Dublin Core Metadata Element Set (DCME) is used for describing resources for discovery purposes all over the world.

1.2.2 MODS and MARC Family

MARC is an acronym for Machine-Readable Cataloging. MARC forms the fundamentals of MARC standards, which oversee the representation and communication of bibliographic and related information in machine readable form. MARC was developed by Henriette Avram in the 1960’s at the library of Congress [3]. This vast, valuable repository of information needs to be read, retrieved, processed, and used by new tools and technologies. Thus developments are being made in order to transform MARC formats into new interoperable formats, which have led to MODS.

MODS is an acronym for Metadata Object Description Scheme. This has been developed by the Library of Congress' Network Development and MARC standards office in the year 2002. This is an extensible mark-up language (XML) based bibliographic description schema intended to move selected data from existing MARC records. The latest version available in the market is MODS version 3.4 [4]. The features and relationship of MODS with MARC are as follows:

- MODS semantics originated with MARC. The element set includes a subset of MARC fields and generally inherit its semantics.
- Unlike MARC, MODS uses language based tags rather than numeric ones.
- MODS regroups elements from MARC format
- MODS uses attributes to refine elements
- MODS don't use any specific cataloging such as the Anglo-American Cataloguing Rules [5].

1.2.3 CDWA

CDWA stands for Categories for the Description of Works of Art. Digital collections and digital library projects for cultural objects and visuals have been a focus of many cultural heritage institutions of the world. The CDWA standard issues guidelines for the description of art objects and images also includes a discussion on issues involved in building art information systems. CDWA is a product of the Art Information Task Force (AITF) which was initiated in early 90's. The current CDWA is a frame-work rather than a set of elements designed for implementation. CDWA has 31 broad categories and more than 380 sub-categories. The purpose of CDWA is to make data more compatible and easily accessible for existing art information systems as well as developing new systems. This standard aims to present the content of databases by describing and accessing information about cultural objects and related visual resources [6]. The standard CDWA has provided a detailed mapping of important metadata standards elements such as MARC and DC. CDWA has become a basis for a number of derived metadata standards like Visual Resources Association Core 3.0 and 4.0, cdwa lite, object id, and more.

1.3 Issues with Metadata

As discussed in the abstract, metadata is quite large and growing at an exponential rate. Metadata is very important as it provides information about the data, data retrieval, and location of data, thereby helping improve performance. In the present day, the data (and its subsequent meta-data) is increasing at such a rate separate storage and maintenance for metadata may need to be considered. This would of course increase costs. In today’s market, customers pay only for the stored data, not the meta-data which makes their lives easier by providing quick access to relevant material. Thus the total cost for metadata maintenance is borne by the service provider.

Numerous solutions to address this issue have been proposed. These proposals can be categorized into two types: compression algorithms which compress the available metadata, and a new efficient methodology for the storage of meta-data. This thesis proposes an algorithm to compress the available Metadata, the objective of which is to verify if the proposed algorithm will result in efficient retrieval of metadata while improving performance.

1.4 Partitioning

Partitioning uses the policy of “Divide and Conquer” to improve the oracle maintenance [7, 8, and 9]. The concept of partitioning was first introduced in oracle 8.0 in 1997. This concept is considered to be one of the most successful and important functionalities in the history of oracle databases. Partitioning has numerous benefits which improve the functionality of databases for various applications. This is done by improving the manageability, performance and availability. One of the most important benefits of the partitioning technique is that it cuts costs. It uses the “Tiered Archiving” technique to store the relevant information, which saves server space. Partitioning is a simple and effective approach when the information lifecycle management is considered for huge environments.

Partitioning allows a table to be sub-divided into numerous smaller pieces, which are then called partitions. Partitions can have their own name and can also have separate storage characteristics depending upon requirements. One more major advantage in using partitions is that each partition can be managed either individually or collectively. This provides the administrator flexibility in managing the operations of the partitions. From an application perspective there is no difference between a partitioned table and non-partitioned table. There is no change or modification required in the respective Structured Query Language (SQL) commands to access the partitioned table. The database objects such as tables are partitioned using a variable called a partitioning key. These partitioning keys vary. The figure below shows a database table which is partitioned based on different months.

Maintenance operations can be performed on the required partitions rather than performing it on the entire database [7, 8 and 9]. If an administrator wanted to remove data, he could directly remove the respective partition which is relatively quick and more efficient than the process of deleting each and every row individually. Partitions provide independence; due to this characteristic they have very high-strategic advantage. Administrators can save each partition at a separate table space which would be useful in the event of a recovery process after a disaster. During a disaster the database could be recovered with the help of partitions compri-

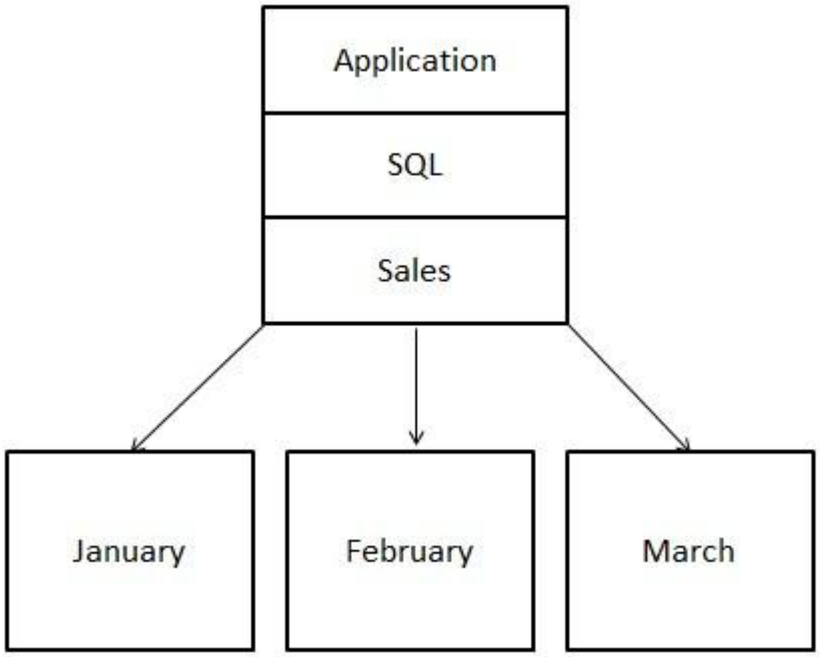


Figure 1 Partitioned Database Table

sing of active data, saving the recovery of inactive data for a later time. Thus partitioning helps reduce system down-time. There are various types of partitions that database supports. They are:

- Range Partitioning
- List Partitioning
- Interval Partitioning
- Hash Partitioning
- Composite Partitioning
- System Partitioning
- Reference Partitioning

Range partitioning is a type of partitioning where data maps to relevant partitions based on a range of column values, usually a date column. List Partitioning is a type of partitioning where the data maps to the relevant partition based on the list of discrete values. In the case of

Interval Partitioning, data is mapped to the respective partition based on the intervals. In Hash partitioning, the data will be mapped to the respective partition on the basis of a hashing algorithm. Composite partitioning is a hybrid type of partitioning which combines any of the above two methods. Research is also being conducted in this area.

CHAPTER 2

LITERATURE SURVEY

2.1 Related Work

The demand for additional storage capacity for every organization is increasing at an exponential rate. Though disc storage cost per gigabyte (GB) has been decreasing constantly, the organizations are unable to find a cost-neutral solution with the amount of growth in data required for the organization [10]. Although all data is not active data it is important to maintain everything for organizational benefits in the future. Thus the concept of data compression is increasingly important. Numerous researchers still continue to choose this field for their current research due to its increasing importance. Many data compression algorithms have been proposed to date to meet the current industry requirements.

Data compression refers to the art of representing information using fewer bits than the original representation [11]. It is also known as source coding or bit-rate reduction. This is very useful as it helps save resources such as bandwidth, hard disc space, etc. Issues concerning de-compressing files or images also need to be considered, though. This requires extra processing capabilities which may be vital in some applications. Compression is classified into two types, lossy compression, and lossless compression. Lossy compression is a data encoding method which compresses the data by removing some of the data [12]. In general terms this method of compression is used for multimedia data such as images, audio, etc. A group of compression algorithms that replicate the exact output as the original input fall under the category of lossless compression algorithms [13]. The work done in this research takes a closer look at lossless compression algorithms.

Data that is being stored in the database tables continues to increase at an exponential rate for every business organization. This idea of data compression can be implemented on database tables to get the general benefits of data compression. Much research has been conducted in this area of database compression [14, 15, and 16]. Compression techniques tend to improve the central processing unit performance [16]. Many algorithms have been proposed by researchers to substantiate this requirement. Huang, Wang, and Xu [14] propose a data compression algorithm which falls in the category of lossless data compression. This algorithm was designed based on LZW and RLE algorithms. Huang et al use priority data for their research. In real-time the priority data was divided into three portions depending upon the characteristics of the data. The three partitions of the priority data were as follows:

- Data value,
- Time stamp, and
- Quality code

In the above paper, in-depth analysis has been made on every compression algorithm. Results of this paper show that the better algorithm to compress the data value of priority data was LZW algorithm, as LZW is based upon a dictionary model. Time stamp and the quality code of the priority data compress better with an RLE compression algorithm. The algorithm is works in such a way that it chooses the best fit by analyzing the characteristics of the respective data in the respective data package in order to achieve better results.

Zhen and Ren [15] have been successful in developing a new compression algorithm which is a hybrid algorithm of LZ78 and LZW algorithms. This algorithm has been proposed to the same priority data considered in "A Lossless data compression algorithm for real-time Database" [14].The results of this new compression algorithm prove that it has a better performance than the previous algorithm which moves the data to the respective compression algorithm depending upon the characteristic of the data. The results from this paper indicate that the compression rate has been doubled from the previous algorithm. Also the compression time and de-compression times have been lowered. Thus this algorithm improves the performance of the databases in real-time.

Research has also been conducted on the performances of the databases which work on the idea of database compression. In “Database compression techniques for Performance Optimization” [16], the author’s research was focused on the various compression schemes used for database compression and their performance on the databases. The compression algorithms discussed in this paper include null suppression, dictionary encoding, run length encoding, bit-vector encoding, heavy-weight compression schemes, hybrid columnar compression and online transaction processing table compression. This paper discusses all included compression algorithms and their performance on databases. Aghav has found that light weight compression techniques tend to improve the database performance better.

Metadata is another possibility that has been researched in hopes of solving the dilemma of increased storage needs. There are numerous advantages of metadata. One of the major drawbacks or disadvantages of Metadata is that the cost required to generate it and maintain it are extensive [17 and 18].

Yonghong, Hongvan, and Guilong [18] have conducted research on various methods of metadata generation. In this study, the authors explain the various traditional techniques involved in the generation of metadata. They also discuss the flaws of these traditional techniques and propose an algorithm which fixes these flaws. The traditional methods include the manual generation of metadata for databases, which includes three steps: collect, input, and transform into a database. The flaw in the traditional method, as stated in this research, is that the metadata generated by this method cannot update itself when there is a change in the source data. In this paper, the authors have designed an algorithm which generates the metadata dynamically based on the database triggers. Rules for automatic generation of metadata have been discussed. The information about various software's available for metadata generation in the present market has also been included in this paper. This research also shows that the relevant data can be easily accessible using metadata. To check the performance of the massive databases, a data access engine based on metadata has been proposed.

In another work, Song and Bao [19] have worked on the concept of partitioning for massive databases, yet another way in which researchers are attempting to lower storage usage. As discussed earlier, there many types of partitioning concepts such as range partitioning, hash partitioning, and list partitioning. In "Increased Partitioning Approach for Massive Data in real-time Data Ware House", the authors claim that, due to the traditional partitioning approach, there is an increase in burden to the respective database system; therefore, they claim that it is not suited for real-time data warehouse purposes. The authors have proposed three effective algorithms to speed up the current partitioning technology.

CHAPTER 3
ANTICIPATED TECHNIQUE

Metadata is very large and is growing at an exponential rate along with traditional stored data. Metadata is very important as it provides information about the data, data retrieval and location of data, thereby helping improve performance. Data and its subsequent metadata are increasing in volume to a point that separate storage and maintenance for metadata itself may need to be considered, which increases the costs associated with metadata. Customers pay only for stored data, not the metadata accompanying it. Thus the total cost for the metadata maintenance must be borne by the service provider.

To address this issue, numerous solutions have been proposed, which have been categorized into two types: compression algorithms and new, more efficient methodologies for the storage of metadata. We would like to propose an algorithm to compress the available metadata. Our main objective is to verify if the proposed algorithm will result in efficient retrieval of metadata while improving performance.

In business, everything is documented for various purposes. Every chunk of data that is being stored is of value and importance. Data in the present industry is stored by various means such as database tables, disc arrays, etc. Various efficient storage techniques have been proposed and are currently being implemented. In case of multi-national companies, universities, and hospitals, the stored data must be in an organized format. The most common practice to store this kind of data is the use of database tables. This kind of storage improves the efficiency and performance, and therefore is the most common. As the data stored in the database tables increase, various mechanisms must be used to improve the efficiency and the performance of the database. These mechanisms include generating metadata, and creating partitions among others. As we are already aware, metadata helps improve performance and efficiency. In order to substantiate our point in the growth of metadata, we continued our research work with oracle databases. Our goal is to propose a solution for this issue which in turn makes the business more profitable for both the customer and the service provider. We believe the best way to accomplish this is to compress the low priority metadata of the database and store it. Active metadata should not be compressed and stored due to the frequency of use, while historical metadata, while possibly important for future business needs, is used infrequently. For example, details of sales data for every month may not be useful daily but it is useful when one is calculating the performance over a period of time. Our idea is to compress the low priority metadata of the database and store it and retrieve it whenever required. With this space management would become more efficient and costs of maintenance borne by the service provider would be reduced.

3.1 Metadata Insertion into the Servers

In this section an algorithm is proposed which fits our requirement. This algorithm's goal is to separate the metadata generated for data which has lower importance from the metadata of higher importance. The proposed algorithm is defined in such a fashion that the high priority data is moved to one of the legs of the binary tree and the low priority is moved to the other leg in the binary tree architecture. Importance of certain data is decided depending upon various conditions like date, number of hits, etc. In the current proposed algorithm we consider the number of hits to be the relevant condition, which is the working principle for the algorithm. When the number of hits on a particular piece of data is more than a defined threshold value, the data is termed as high priority data. If the number of hits is less than the threshold value, the data is labeled lower priority data. The metadata of such data is compressed using the proposed BWT compression algorithm. Whenever there is a request for historical data, the output provided would be a general form of data as it will be presented after it is run through the decompression algorithm. This algorithm calculates the time required to fulfill our requirement and also tells us about the cost saved due to this process.

3.1.1 Proposed Algorithm:

*T= (60*60*24*K) (s) {K is the number of days which is defined by the user.}*

This function is a timer function based on the number of seconds and decrements every second.

```
if ( T !=0)
    if (read request to a file == True)
            H(i)++; { Increment the hit counter of the file}
        end if;
      else
        Display the Value of the Hit Count;
        H(i)=0;
end;
The threshold value is defined as the average value of number of hits for 'k' files.
Th = {H(1)+H(2)+......+H(k)}/ k ; where 'k' is the number of files.
if ( the number of hits are greater than average threshold value)
then the data is stored in the right leg(say) of the binary sub-tree ;
```

else

the meta-data of the respective file is compressed using BWT algorithm and stored in the left leg(say) of the binary sub-tree

3.1.2 Working of the Algorithm

The working principle of the algorithm is that metadata associated with data of lower importance needs to be compressed and stored. The determination of this condition is done based on the number of hits data acquires over a week's time. This is one of the most important parameters for the working of this algorithm. In the proposed algorithm this condition is defined as H(i). Whenever there is a request for any chunk of data (i), a hit counter is run which counts in increments by (1) when the request is made. These particular hit counters are then zeroed after a period of time. A threshold value is calculated by using the algorithm, denoted as H(t) in the proposed algorithm. This is also an important factor which plays a major role in the working of the algorithm. This value helps the algorithm categorize the data with needful specifications. This algorithm may be run at end of every week, 15 days, or monthly during the organization's maintenance hours. When this algorithm runs, it checks the required parameters discussed above and categorizes the data into high priority and low priority. The metadata of the data which has higher importance is stored in the one of the legs of the binary tree and the metadata of the data with lower importance is compressed using the compression algorithm and stored in the other leg of the binary tree. Every node in this architecture is a server and the threshold values at every server are calculated using the pseudo code. Assume that the servers are named from right to left at each and every leg as shown in the figure below. Threshold values are calculated in every server depending upon the architecture. The pseudo code below helps the user to calculate the threshold values at every node in the architecture. This algorithm was not tested in a real-time/practical scenario due to cost constraints, though it has been tested theoretically.

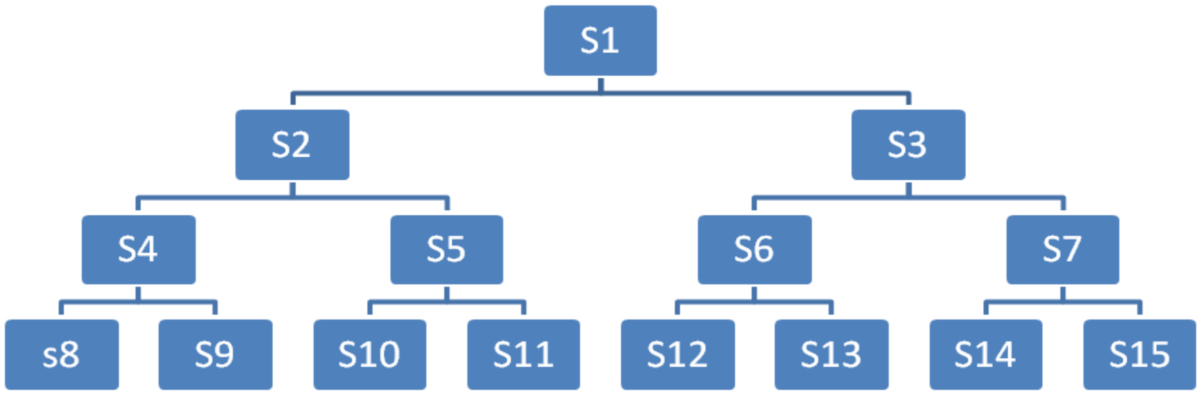


Figure 2 Binary tree architecture

3.1.2.1Pseudo Code

```
for (i=2:n; i=<n ; i++) ;  Initialize the value of the variable ‘i’ from 2 to n
For the condition when ‘i’ is less than or equal to the number of nodes in a binary tree structure

if (i%2==1) ; Whenever the remainder is one, when ‘i’ is divided by 2

    if the above condition holds true then

        th_i = th_(i-1/2) + th_root;

    end if loop;

else if the above condition doesn’t hold true then,

    m=i; Initialize the variable ‘m’ to the value of ‘i’
```

```
Check the value of the variable ’m’ and check whether if it is equal to 2
if ( m==2)
    if the above condition holds true then
    th_i = th_root/2;
    root_node = i; change the root node to the current node ‘i’
end if;
  else  if the above condition doesn’t hold true then
    Initialize a variable ‘f’ to the value of the variable ‘m’
    Check if it is not equal to 1
    for (f = m ; f!=1;  )
    if the above condition holds true then
        m= m/2;
        f = m%2;
        Check whether the value of the variable ’m’ is equal to 2
        if (m==2)
        if the above condition holds true then
            th_i = th_root/2;
            root_node = i; change the root node to the current node ‘i’
        end if;
        end for;
        else
        if the above condition doesn’t hold true then
        Check whether the value of variable f is 1
            if ( f==1)
            if the above condition holds true then
                $th\_i = th\_(i/2) – th\_root$;
            end if;
        end for;
    end for;
```

3.2 Compression Algorithms

Data compression refers to the process of encoding information using fewer bits than the original representation [11]. It is also known as source coding or bit-rate reduction. Compression is classified into two types, which are

- Lossy Compression and
- Lossless Compression.

3.2.1 Lossy Compression Algorithms

Lossy Compression is also known as perceptual coding. It is a data encoding method which compresses the data by removing some of the data [12]. In general terms this method of compression is used for multimedia data including images and audio. Examples of Lossy compression algorithms are fractal compression, cartesian perceptual compression, wavelet compression, block truncation coding, DV, H.261, H.263, speex, dirac, and musepack. All of these examples fall under different categories of lossy compression: image compression, audio compression, etc. Cartesian perceptual compression is a lossy compression used for image compression [20]. Fractal compression is also a lossy compression technique used for image compression. This is a new and patented technology. The speed of compression is very slow whereas decompression is fast [21]. Both are patented compression techniques. Wavelet compression is also an image compression technique. Here the compression is done based on wavelets. The functions that permit data analysis of images with respect to scales or resolutions are known as wavelets [22]. DV is another lossy Video compression scheme. This compression scheme is an intra-frame video compression scheme that employs the technique of Discrete Cosine Transform (DCT) for video compression. During the compression, audio is stored uncompressed. Speex is used for speech compression. This is an open source technology and patent-free technology designed for audio compression for speech [23].

3.2.2 Lossless Compression Algorithms

A group of compression algorithms that replicate the exact output as the original input fall under the category of lossless compression algorithms [13]. There are various lossless compression techniques such as LZW, huffman, LZ77, and context tree weighting. Lempel-Ziv-Welch (LZW) is a universal lossless data compression algorithm which was developed as an improvement of the previous LZ77 algorithm. Implementation of this compression technique is very simple and is useful during hardware implementations. LZ77 is the basic lossless compression algorithm developed and proposed by Lempel and Ziv in 1977.It is a sliding window-based compression algorithm. CTW (Context Tree Weighting method) is a prediction algorithm which has very close theoretical and practical values. However, the work done in this research takes a closer look at the Burrows-Wheeler Transform (BWT) lossless compression algorithm.

3.2.2.1Working of BWT

This compression technique falls under the category of lossless compression techniques. It is named after the two scientists who invented the algorithm in 1994, Micheal Burrows and David Wheeler. This compression algorithm is also known as block-sorting compression algorithm. Burrows-wheeler transform is considered to be one best data compression algorithms currently available due to its simplicity and effectiveness and has received considerable attention since its invention. The algorithm runs on the basis of the permutations of the input sequence known as Block-Sorting. This blocking sorting helps in grouping symbols that are similar. In the original proposal, the Block-sorting was followed by MTF transformation and an entropy coding stage. Figure 3 represents the block diagram for the compression algorithm in its basic stages.

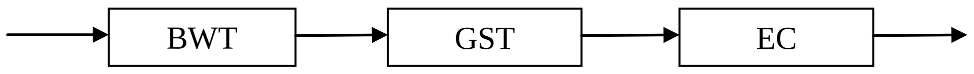


Figure 3 Block Diagram of Basic BWT Compression Algorithm

Several researchers have been working on improving this compression. Many changes have been suggested by various researchers and have been implemented to make it more effective. The most recent BWT compression algorithm contains all the approved changes embedded in it. The figure below represents the block diagram of the present BWT algorithm that is being used.

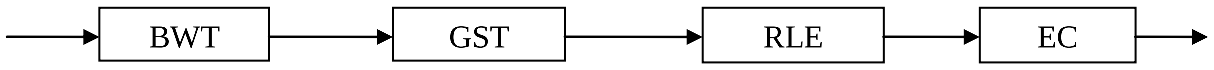


Figure 4 Block Diagram of Current BWT Compression Algorithm

Figure 4 is the BWT compression algorithm consists of four stages: BWT transformation, Global Structure Transformation (GST), Run-Length encoding (RLE), and entropy coding stage. This algorithm works based on the above block diagram and moves sequentially from left to right. Every stage of the algorithm has a transformation of the input data and the output is obtained, which becomes the input to the next stage. The first stage of the algorithm is the block-sorting stage, where the given data is sorted based on the similarity of the data. In this stage everything similar is grouped together. Numbers of symbols are kept constant during this stage. The next stage is Global Structure Transformation (GST). In this stage the local context of the symbols are changed to a global context. This is a very similar stage to Move-To-Front (MTF) that was used by the Burrows and Wheeler in their original publication. Here, MTF is a List Update Algorithm (LUA) where the symbols are replaced with the corresponding ranking values. Just like the first stage, the numbers of symbols are not altered in this stage. The output of the second stage is provided as the input of the third stage which typically uses Run-Length Encoding scheme to shrink the symbols. Here in this stage many other algorithms can be used to fulfill the required purpose, but previous research proved that RLE is the best fit for this purpose [24, 25]. The working of the RLE is based on replacing a long sequence of symbols by a shorter sequence. The working principle of RLE is based on three points: If l<t then it is unchanged, if l > t or l =t then the run is replaced, where ‘l’ is the length of the long sequence and t is the threshold. The output of the third stage is provided as the input of the fourth which is entropy coding. The process of converting of symbols into bits based on an estimated probability distribution is known as entropy encoding. In order to understand the process of BWT compression easily let take the example of a string which contains the characters ““cdtcecfcdtccdtcecfcdtc”. This particular is provided as the input to the BWT the algorithm. At the first stage of the algorithm the following string is converted or transformed using blocking-sorting technique. The obtained output is as follows:

BWT Input: 63 64 74 63 65 63 66 63 64 74 63 63 64 74 63 65 63 66 63 64 74 63

BWT Output: 63 74 74 66 66 74 74 65 65 63 63 63 63 63 63 63 63 64 64 64 64

The above BWT output is provided as the input to the second stage i.e. GST. Then output obtained is as shown:

GST Output: 63 74 00 67 00 02 02 00 67 00 02 00 00 00 00 00 00 00 67 00 00 00

The output obtained from GST is provided as the input to the RLE algorithm, and then output obtained will be:

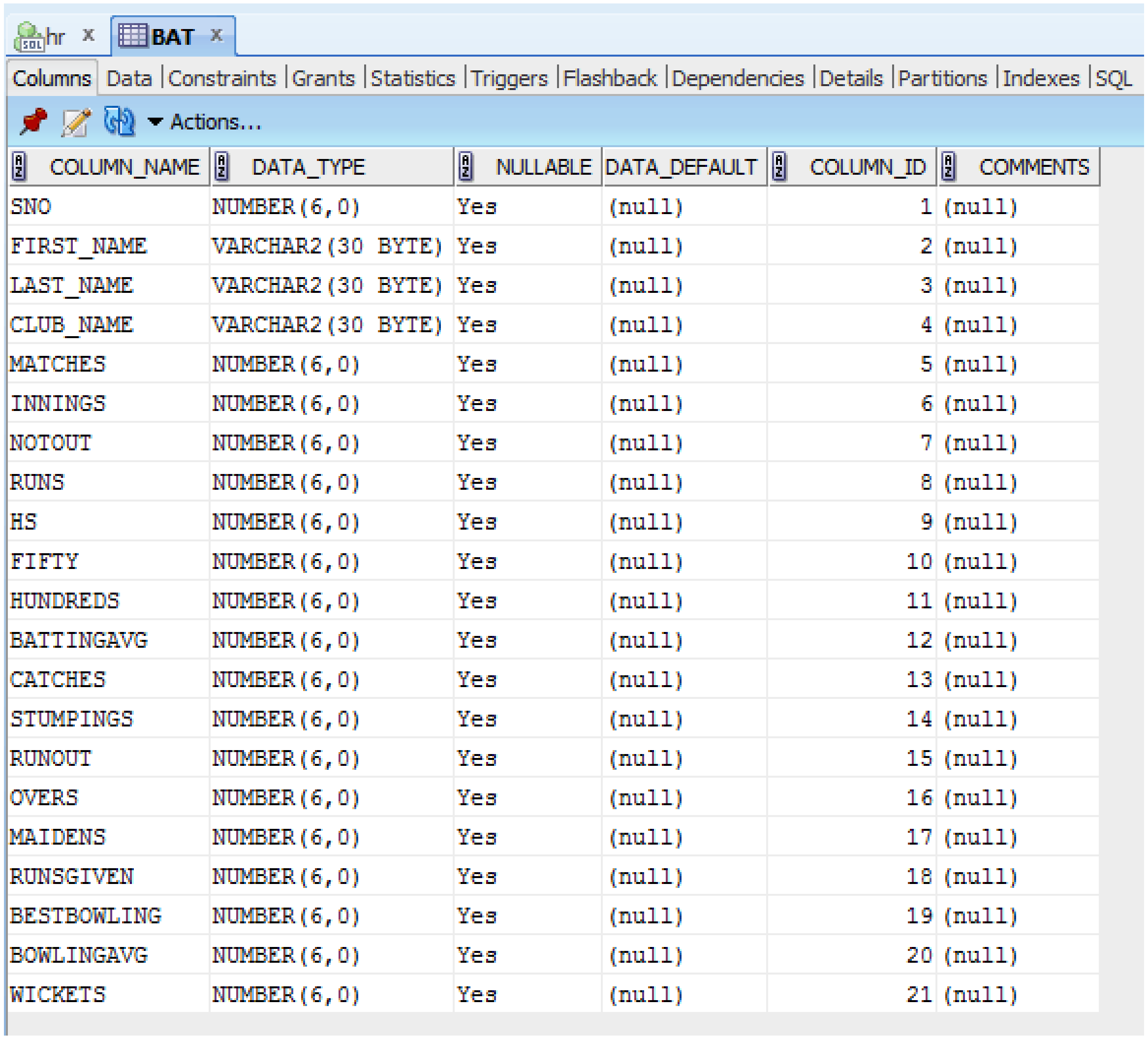
RLE Output: 65 74 00 67 00 04 04 00 69 00 04 00 00 00 69 00 00

This output from RLE is provided to an entropy coding unit. The EC unit then works on this string and produces an output using an arithmetic coding scheme. The output string looks like

EC Output: 00 0C 03 8B B3 6F 85 00 72 C8 E8 2C

The Burrows-wheeler Transform is a very simple though very effective compression algorithm. The compression rates of the Burrows–wheeler transform are better than other compression algorithms available. The speed of both compression and de-compression are very high. Also, due to the fact that it was block-sorting technique, it compresses massive databases better than other compression algorithms, leading to improved performance.

3.3 Methodology



COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
SNO	NUMBER(6,0)	Yes	(null)	1	(null)
FIRST_NAME	VARCHAR2(30 BYTE)	Yes	(null)	2	(null)
LAST_NAME	VARCHAR2(30 BYTE)	Yes	(null)	3	(null)
CLUB_NAME	VARCHAR2(30 BYTE)	Yes	(null)	4	(null)
MATCHES	NUMBER(6,0)	Yes	(null)	5	(null)
INNINGS	NUMBER(6,0)	Yes	(null)	6	(null)
NOTOUT	NUMBER(6,0)	Yes	(null)	7	(null)
RUNS	NUMBER(6,0)	Yes	(null)	8	(null)
HS	NUMBER(6,0)	Yes	(null)	9	(null)
FIFTY	NUMBER(6,0)	Yes	(null)	10	(null)
HUNDREDS	NUMBER(6,0)	Yes	(null)	11	(null)
BATTINGAVG	NUMBER(6,0)	Yes	(null)	12	(null)
CATCHES	NUMBER(6,0)	Yes	(null)	13	(null)
STUMPINGS	NUMBER(6,0)	Yes	(null)	14	(null)
RUNOUT	NUMBER(6,0)	Yes	(null)	15	(null)
OVERS	NUMBER(6,0)	Yes	(null)	16	(null)
MAIDENS	NUMBER(6,0)	Yes	(null)	17	(null)
RUNSGIVEN	NUMBER(6,0)	Yes	(null)	18	(null)
BESTBOWLING	NUMBER(6,0)	Yes	(null)	19	(null)
BOWLINGAVG	NUMBER(6,0)	Yes	(null)	20	(null)
WICKETS	NUMBER(6,0)	Yes	(null)	21	(null)

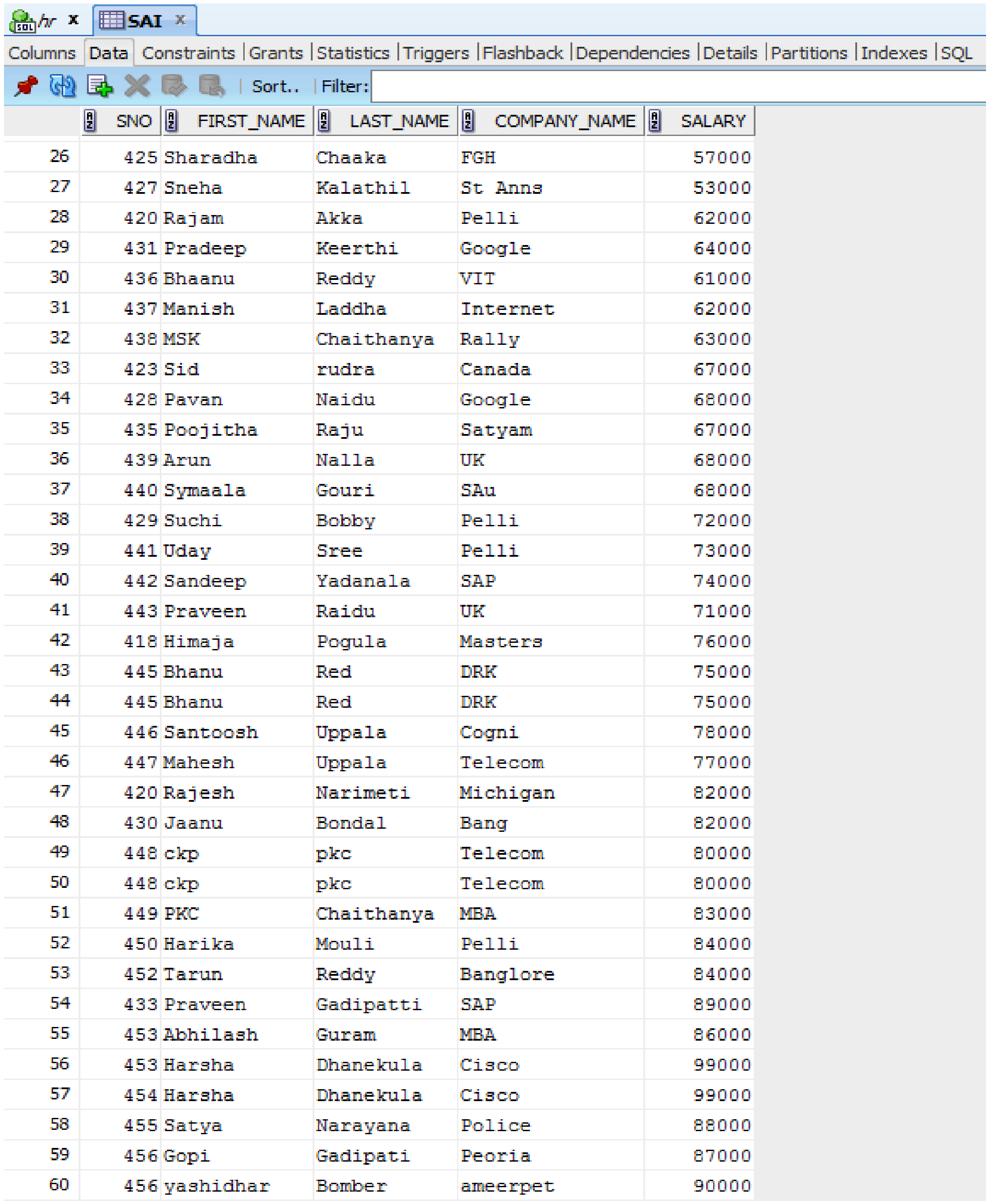
Figure 5 Screenshot of a database table with columns

Oracle 11g database was selected to be test-bed for our research. The two primary reasons for selecting oracle are:

1. free open source software and
2. real-time approach

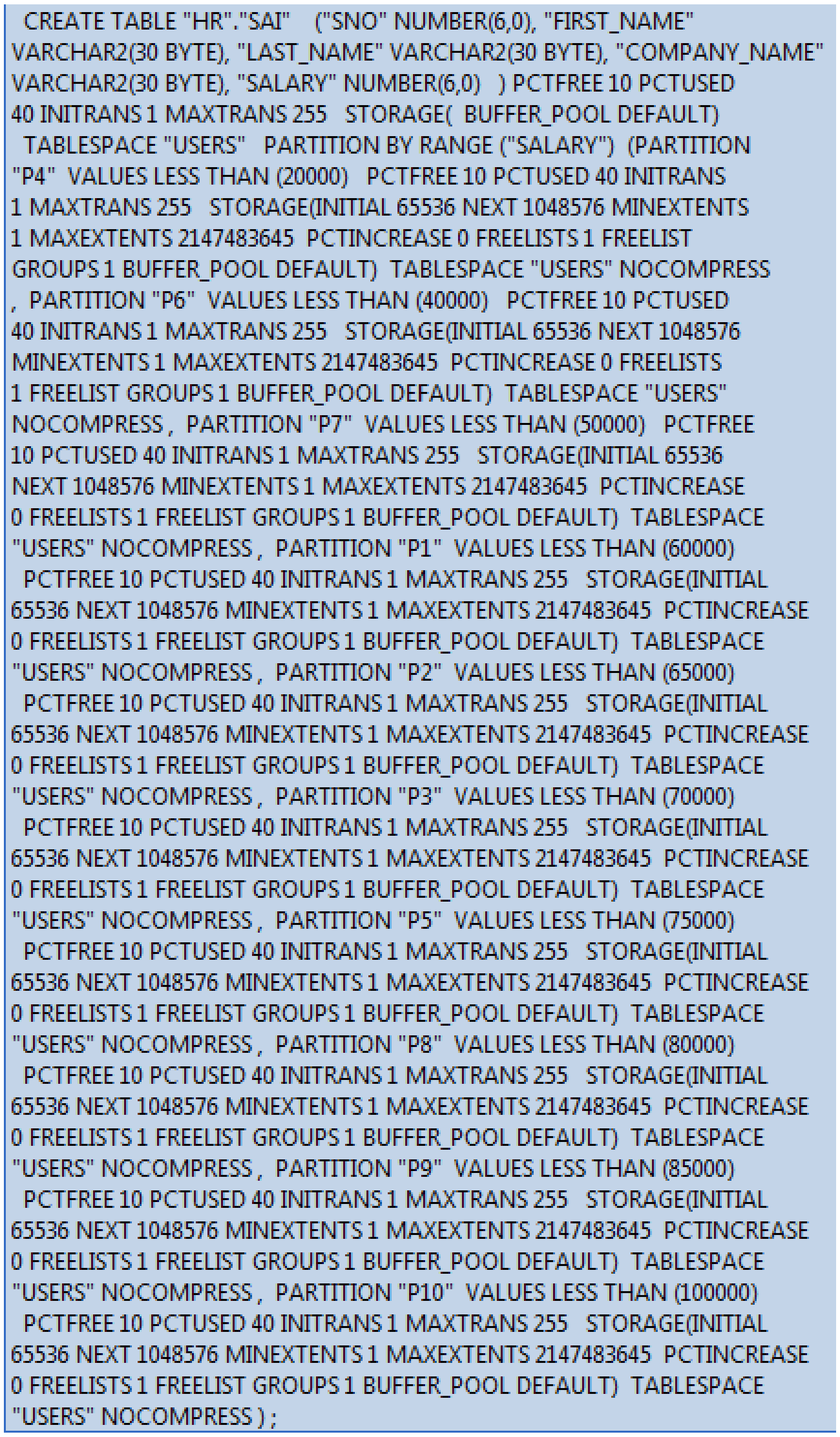
This allows for answers of usability in current practical applications. The below figures represent the various screen shots taken from the oracle database.

Figure 5 is the screen shot of a table named BAT with its respective columns and figure 6 is the screen shot of the data entered into the table named SAI.



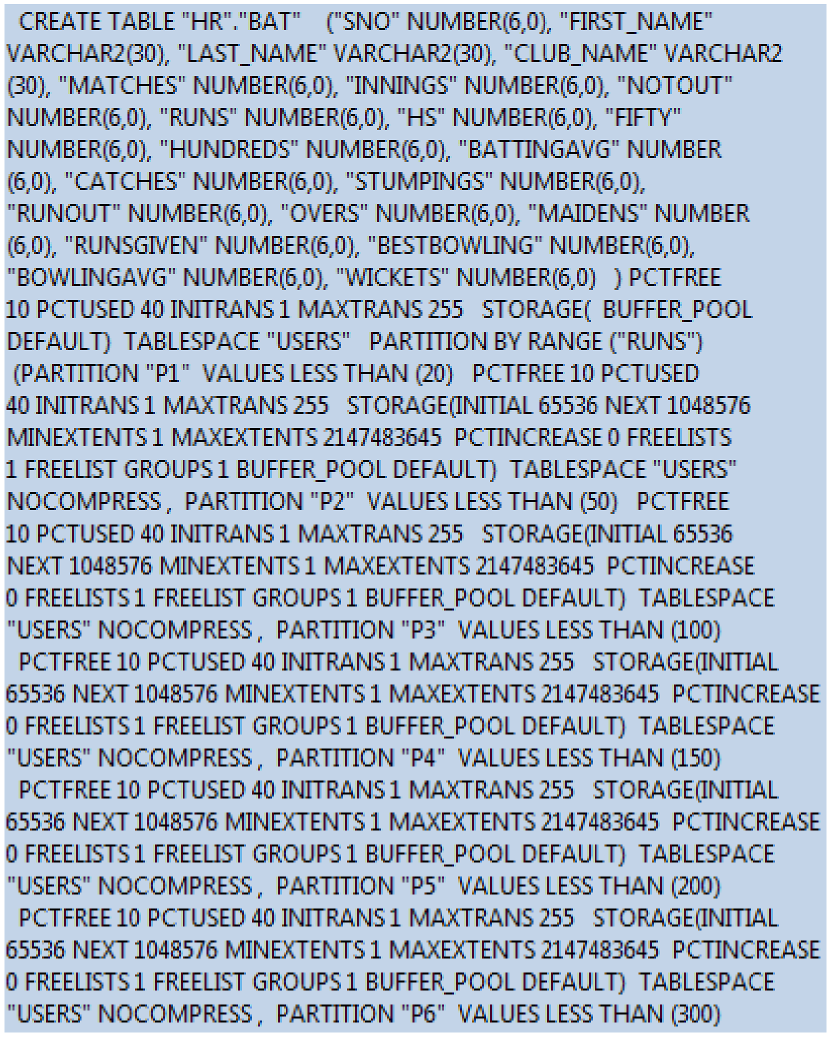
	SNO	FIRST_NAME	LAST_NAME	COMPANY_NAME	SALARY
26	425	Sharadha	Chaaka	FGH	57000
27	427	Sneha	Kalathil	St Anns	53000
28	420	Rajam	Akka	Pelli	62000
29	431	Pradeep	Keerthi	Google	64000
30	436	Bhaanu	Reddy	VIT	61000
31	437	Manish	Laddha	Internet	62000
32	438	MSK	Chaithanya	Rally	63000
33	423	Sid	rudra	Canada	67000
34	428	Pavan	Naidu	Google	68000
35	435	Poojitha	Raju	Satyam	67000
36	439	Arun	Nalla	UK	68000
37	440	Symaala	Gouri	SAu	68000
38	429	Suchi	Bobby	Pelli	72000
39	441	Uday	Sree	Pelli	73000
40	442	Sandeep	Yadanala	SAP	74000
41	443	Praveen	Raidu	UK	71000
42	418	Himaja	Pogula	Masters	76000
43	445	Bhanu	Red	DRK	75000
44	445	Bhanu	Red	DRK	75000
45	446	Santoosh	Uppala	Cogni	78000
46	447	Mahesh	Uppala	Telecom	77000
47	420	Rajesh	Narimeti	Michigan	82000
48	430	Jaanu	Bondal	Bang	82000
49	448	ckp	pkc	Telecom	80000
50	448	ckp	pkc	Telecom	80000
51	449	PKC	Chaithanya	MBA	83000
52	450	Harika	Mouli	Pelli	84000
53	452	Tarun	Reddy	Banglore	84000
54	433	Praveen	Gadipatti	SAP	89000
55	453	Abhilash	Guram	MBA	86000
56	453	Harsha	Dhanekula	Cisco	99000
57	454	Harsha	Dhanekula	Cisco	99000
58	455	Satya	Narayana	Police	88000
59	456	Gopi	Gadipati	Peoria	87000
60	456	yashidhar	Bomber	ameerpet	90000

Figure 6 Screenshot of database table with data



```
  CREATE TABLE "HR"."SAI"    ("SNO" NUMBER(6,0), "FIRST_NAME"
VARCHAR2(30 BYTE), "LAST_NAME" VARCHAR2(30 BYTE), "COMPANY_NAME"
VARCHAR2(30 BYTE), "SALARY" NUMBER(6,0)   ) PCTFREE 10 PCTUSED
40 INITRANS 1 MAXTRANS 255   STORAGE(  BUFFER_POOL DEFAULT)
  TABLESPACE "USERS"   PARTITION BY RANGE ("SALARY")  (PARTITION
"P4"  VALUES LESS THAN (20000)   PCTFREE 10 PCTUSED 40 INITRANS
1 MAXTRANS 255   STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS
1 MAXEXTENTS 2147483645  PCTINCREASE 0 FREELISTS 1 FREELIST
GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE "USERS" NOCOMPRESS
,  PARTITION "P6"  VALUES LESS THAN (40000)   PCTFREE 10 PCTUSED
40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL 65536 NEXT 1048576
MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE 0 FREELISTS
1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE "USERS"
NOCOMPRESS ,  PARTITION "P7"  VALUES LESS THAN (50000)   PCTFREE
10 PCTUSED 40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL 65536
NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE
0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE
"USERS" NOCOMPRESS ,  PARTITION "P1"  VALUES LESS THAN (60000)
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL
65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE
0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE
"USERS" NOCOMPRESS ,  PARTITION "P2"  VALUES LESS THAN (65000)
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL
65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE
0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE
"USERS" NOCOMPRESS ,  PARTITION "P3"  VALUES LESS THAN (70000)
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL
65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE
0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE
"USERS" NOCOMPRESS ,  PARTITION "P5"  VALUES LESS THAN (75000)
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL
65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE
0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE
"USERS" NOCOMPRESS ,  PARTITION "P8"  VALUES LESS THAN (80000)
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL
65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE
0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE
"USERS" NOCOMPRESS ,  PARTITION "P9"  VALUES LESS THAN (85000)
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL
65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE
0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE
"USERS" NOCOMPRESS ,  PARTITION "P10"  VALUES LESS THAN (100000)
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL
65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE
0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE
"USERS" NOCOMPRESS ) ;
```

Figure 7 Screenshot of metadata output from database table

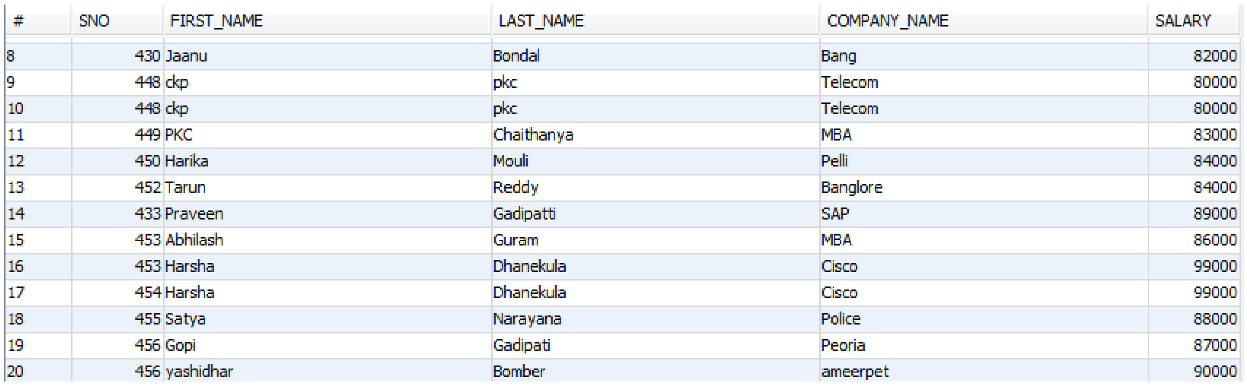


```
 CREATE TABLE "HR"."BAT"    ("SNO" NUMBER(6,0), "FIRST_NAME"
VARCHAR2(30), "LAST_NAME" VARCHAR2(30), "CLUB_NAME" VARCHAR2
(30), "MATCHES" NUMBER(6,0), "INNINGS" NUMBER(6,0), "NOTOUT"
NUMBER(6,0), "RUNS" NUMBER(6,0), "HS" NUMBER(6,0), "FIFTY"
NUMBER(6,0), "HUNDREDS" NUMBER(6,0), "BATTINGAVG" NUMBER
(6,0), "CATCHES" NUMBER(6,0), "STUMPINGS" NUMBER(6,0),
"RUNOUT" NUMBER(6,0), "OVERS" NUMBER(6,0), "MAIDENS" NUMBER
(6,0), "RUNSGIVEN" NUMBER(6,0), "BESTBOWLING" NUMBER(6,0),
"BOWLINGAVG" NUMBER(6,0), "WICKETS" NUMBER(6,0)   ) PCTFREE
10 PCTUSED 40 INITRANS 1 MAXTRANS 255   STORAGE(  BUFFER_POOL
DEFAULT)  TABLESPACE "USERS"   PARTITION BY RANGE ("RUNS")
 (PARTITION "P1"  VALUES LESS THAN (20)   PCTFREE 10 PCTUSED
40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL 65536 NEXT 1048576
MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE 0 FREELISTS
1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE "USERS"
NOCOMPRESS ,  PARTITION "P2"  VALUES LESS THAN (50)   PCTFREE
10 PCTUSED 40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL 65536
NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE
0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE
"USERS" NOCOMPRESS ,  PARTITION "P3"  VALUES LESS THAN (100)
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL
65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE
0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE
"USERS" NOCOMPRESS ,  PARTITION "P4"  VALUES LESS THAN (150)
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL
65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE
0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE
"USERS" NOCOMPRESS ,  PARTITION "P5"  VALUES LESS THAN (200)
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255   STORAGE(INITIAL
65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  PCTINCREASE
0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)  TABLESPACE
"USERS" NOCOMPRESS ,  PARTITION "P6"  VALUES LESS THAN (300)
```

Figure 8 Screenshot of metadata output from database table

Figure 7and figure 8 gives an example of the metadata output of the SAI and BAT database table respectively.

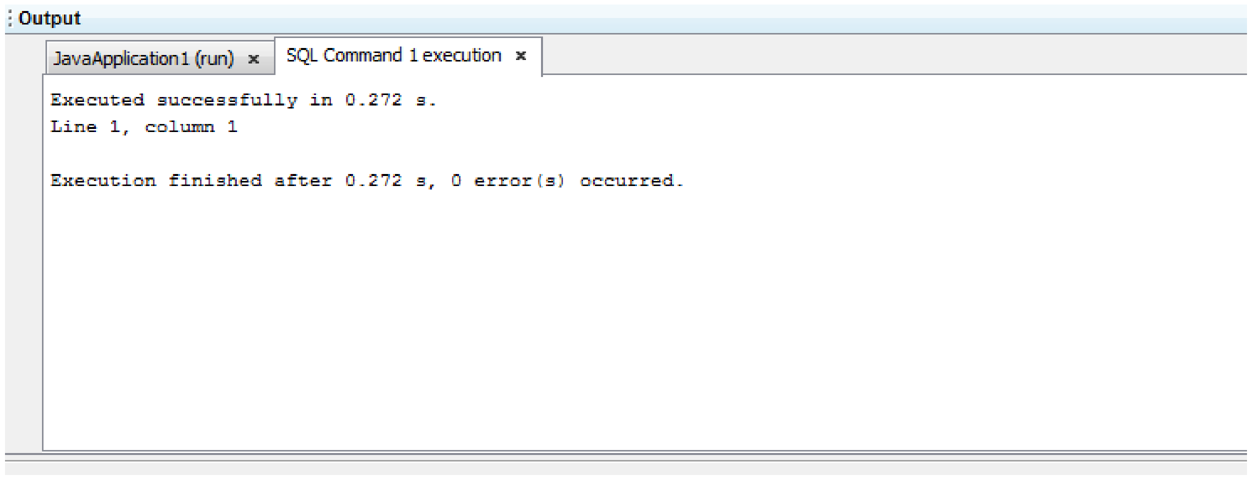
Databases were connected to in real-time via various applications such as netbeans, eclipse, and matlab. In order to have a real-time approach, we established the connectivity between oracle 11g and netbeans. Netbeans is an user interface which connects to the back-end database. After the connection was established successfully, we executed the same commands that were required to obtain the above outputs from the oracle. Figures 9 and Figure 10 show the respective outputs.



#	SNO	FIRST_NAME	LAST_NAME	COMPANY_NAME	SALARY
8	430	Jaanu	Bondal	Bang	82000
9	448	ckp	pkc	Telecom	80000
10	448	ckp	pkc	Telecom	80000
11	449	PKC	Chaithanya	MBA	83000
12	450	Harika	Mouli	Pelli	84000
13	452	Tarun	Reddy	Banglore	84000
14	433	Praveen	Gadipatti	SAP	89000
15	453	Abhilash	Guram	MBA	86000
16	453	Harsha	Dhanekula	Cisco	99000
17	454	Harsha	Dhanekula	Cisco	99000
18	455	Satya	Narayana	Police	88000
19	456	Gopi	Gadipati	Peoria	87000
20	456	yashidhar	Bomber	ameerpet	90000

Figure 9 Screenshot of Database Table Output with Data Using Netbeans

Figure9 is a screen shot taken from the Netbeans application. The output displays the table data for the table named SAI.

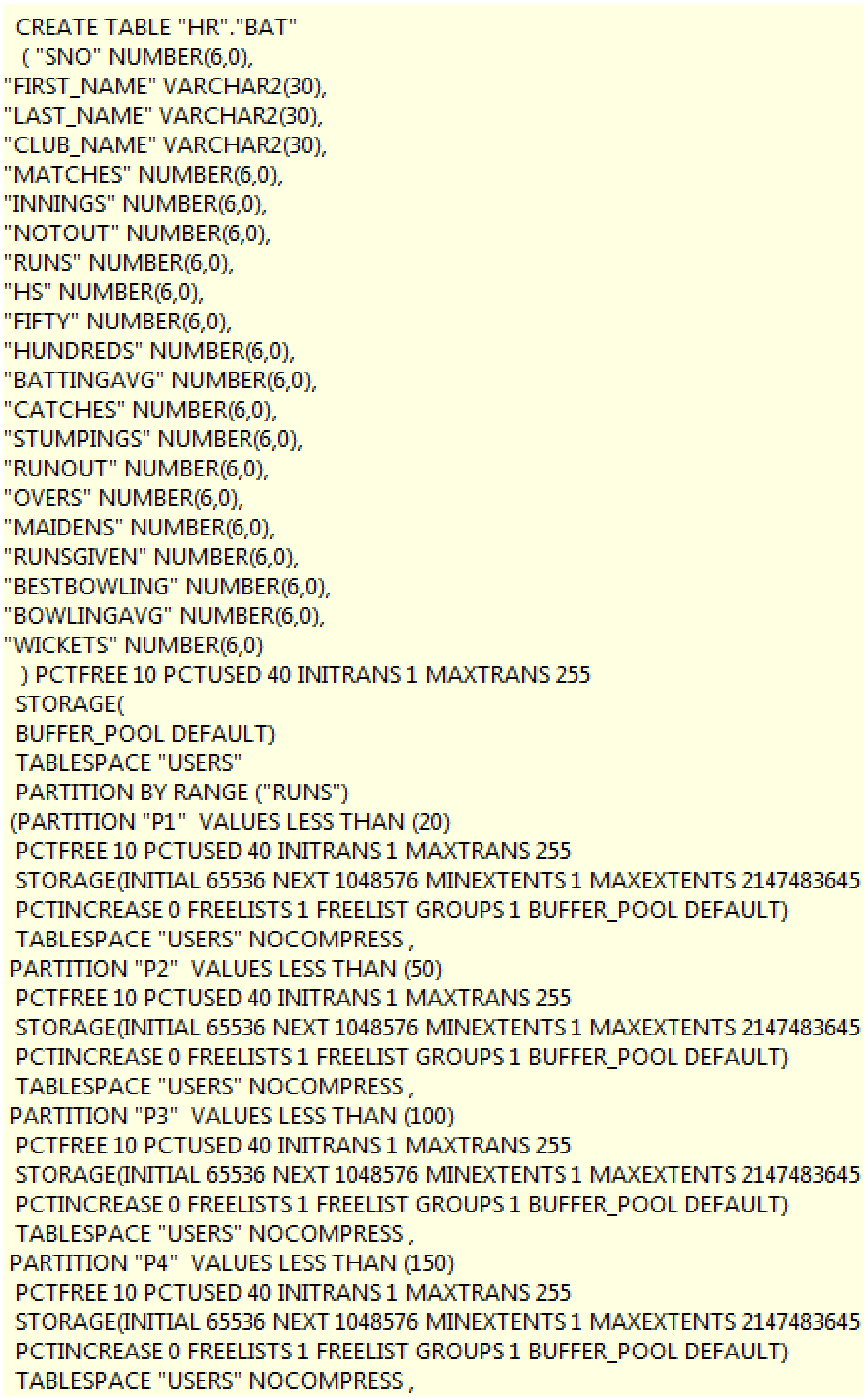


```
Executed successfully in 0.272 s.
Line 1, column 1

Execution finished after 0.272 s, 0 error(s) occurred.
```

Figure 10 Screenshot of time taken to retrieve metadata output of a database table using netbeans

Figure 10 is another screen shot taken from the netbeans application. The output displays the time taken to retrieve the metadata for the table named BAT.



```
 CREATE TABLE "HR"."BAT"
  ( "SNO" NUMBER(6,0),
"FIRST_NAME" VARCHAR2(30),
"LAST_NAME" VARCHAR2(30),
"CLUB_NAME" VARCHAR2(30),
"MATCHES" NUMBER(6,0),
"INNINGS" NUMBER(6,0),
"NOTOUT" NUMBER(6,0),
"RUNS" NUMBER(6,0),
"HS" NUMBER(6,0),
"FIFTY" NUMBER(6,0),
"HUNDREDS" NUMBER(6,0),
"BATTINGAVG" NUMBER(6,0),
"CATCHES" NUMBER(6,0),
"STUMPINGS" NUMBER(6,0),
"RUNOUT" NUMBER(6,0),
"OVERS" NUMBER(6,0),
"MAIDENS" NUMBER(6,0),
"RUNSGIVEN" NUMBER(6,0),
"BESTBOWLING" NUMBER(6,0),
"BOWLINGAVG" NUMBER(6,0),
"WICKETS" NUMBER(6,0)
  ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 STORAGE(
 BUFFER_POOL DEFAULT)
 TABLESPACE "USERS"
 PARTITION BY RANGE ("RUNS")
(PARTITION "P1"  VALUES LESS THAN (20)
 PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
 TABLESPACE "USERS" NOCOMPRESS ,
PARTITION "P2"  VALUES LESS THAN (50)
 PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
 TABLESPACE "USERS" NOCOMPRESS ,
PARTITION "P3"  VALUES LESS THAN (100)
 PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
 TABLESPACE "USERS" NOCOMPRESS ,
PARTITION "P4"  VALUES LESS THAN (150)
 PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
 TABLESPACE "USERS" NOCOMPRESS ,
```

Figure 11 Screenshot of Metadata Output from a Database Table Using Netbeans

Figure 11 is another screen shot taken from the Netbeans application showing the metadata for the table named BAT.

CHAPTER 4

EXPERIMENTAL SETUP AND RESULTS

4.1 Hardware

Processor: Intel Core 2 CPU T7200 @ 2GHz

RAM: 3GB

System Type: 64-bit operating system

OS: Windows 7 Professional

4.2 Software

The following software has been used for simulation purposes: Oracle 11g XE Database and Netbeans IDE (version 7.0.1).

4.3 Test Bed

The simulation was done to substantiate the assumed problem statement. In order to validate the research, we worked on the Oracle database as the backend application while the necessary data was entered through the front-end application netbeans. The connectivity between the oracle database and netbeans had to be established to get this set-up working. The connectivity was established after adding the java database connectivity driver and a new connection. Once the connectivity was established, the required changes of data entry, table alteration, etc can be completed using the Netbeans application. A database table was created with the required specifications. Partitions were also created to simplify the complexity of the database and increase the performance of the database. Data was entered into the database table. The respective metadata was recorded at every partition of the database table. Two more database tables were created in a similar fashion. Every database table created was different from the others. Data was entered into the created database tables depending upon the specifications of the table. Metadata for each database table was recorded separately at each partition. A graph was then plotted using the above values. Figure 12 represents the average of the three tables' general metadata.

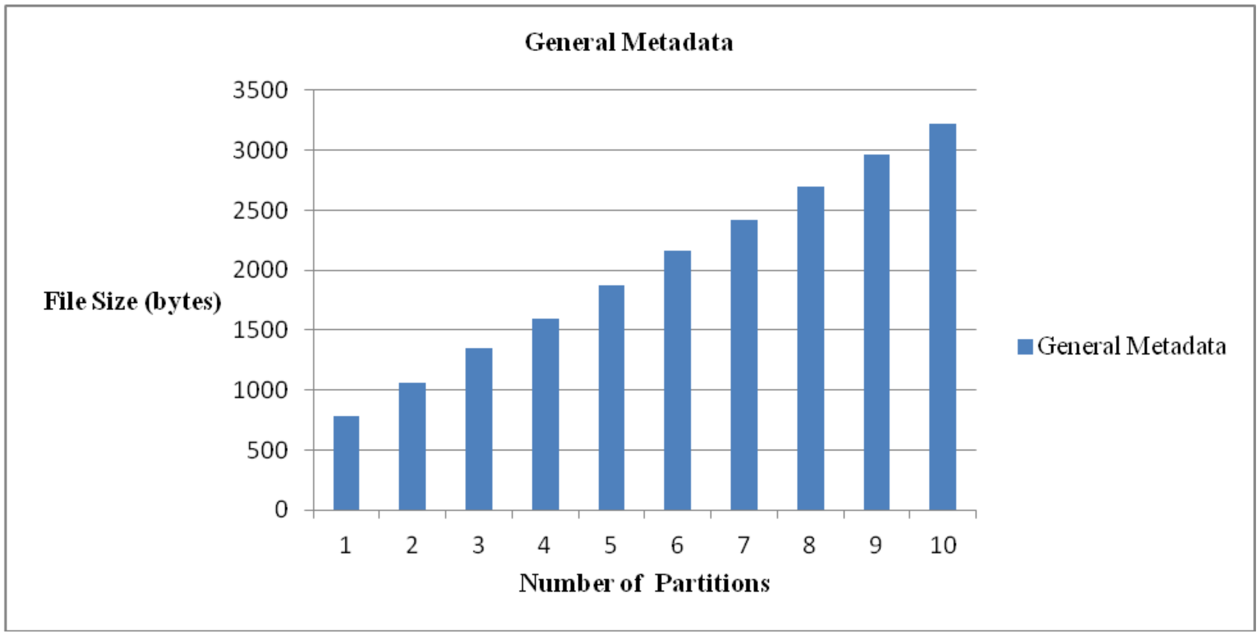


Figure 12 Graph of General Metadata

The x-axis of the above graph represents the number of the partitions in the database table while the Y-axis represents size of the metadata file in bytes. From the above graph it can be observed that the file size increased as the number of the partitions on the database table increased. This indicates that massive databases with a large number of partitions contain massive amounts of metadata, supporting our problem statement.

These metadata files were provided as the input to the burrows-wheeler transform algorithm and the respective outputs were recorded. A graph has been plotted using these values in comparison with the general value of the metadata. Figure 13 represents the graph of the average of compressed metadata in comparison with the average of general metadata for different tables.

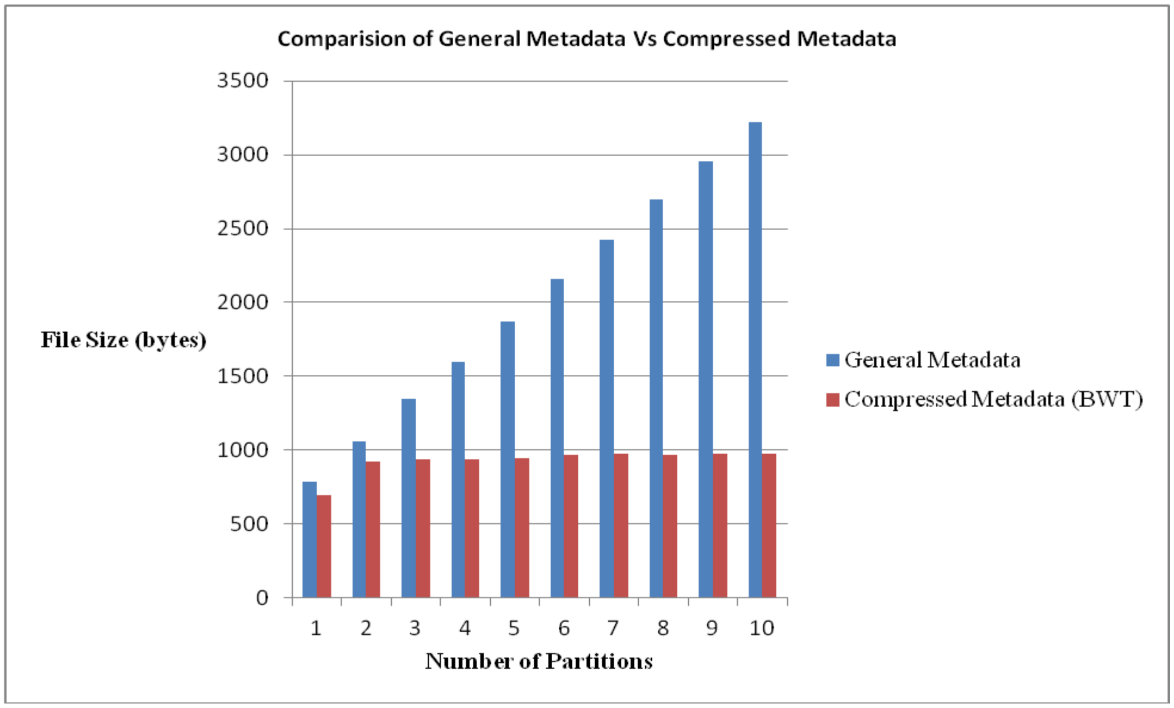


Figure 13 Graph Comparing General Metadata to Compressed Metadata

The X-axis of this graph represents the number of partitions on the various database tables while the Y-axis represents the corresponding size of the metadata file in bytes. The above graph compares the general metadata with the compressed metadata. The bars in blue indicate the general file size of the metadata files whereas the red bars indicate the compressed metadata files. From the above graph it is clearly evident that when the file size of the metadata is less for a table with one partition, the size of the compressed file is also very close to the actual size. As the size of the metadata file increases, the compression ratio improves. By looking at the last value (i.e. partition 10 or table with 10 partitions), the file size of the general metadata is very large when compared to the file size of the compressed metadata. Thus from the above graph we can conclude that the BWT compression algorithm used in this study works better for large files.

In order to check the performance on the real-time basis, we connected to Oracle using the Net-Beans application as well. After the connection was successfully executed, we recorded various times required to access the metadata of every database table for every partition directly from the Netbeans application. We also recorded the times required to de-compress the desired metadata. These de-compressed time values were recorded on a laptop. A graph (Figure 14) has been plotted with the recorded times. The below graph has two graphs embedded into it; one is the time taken to get the general metadata and other is the time taken to get the de-compressed metadata.

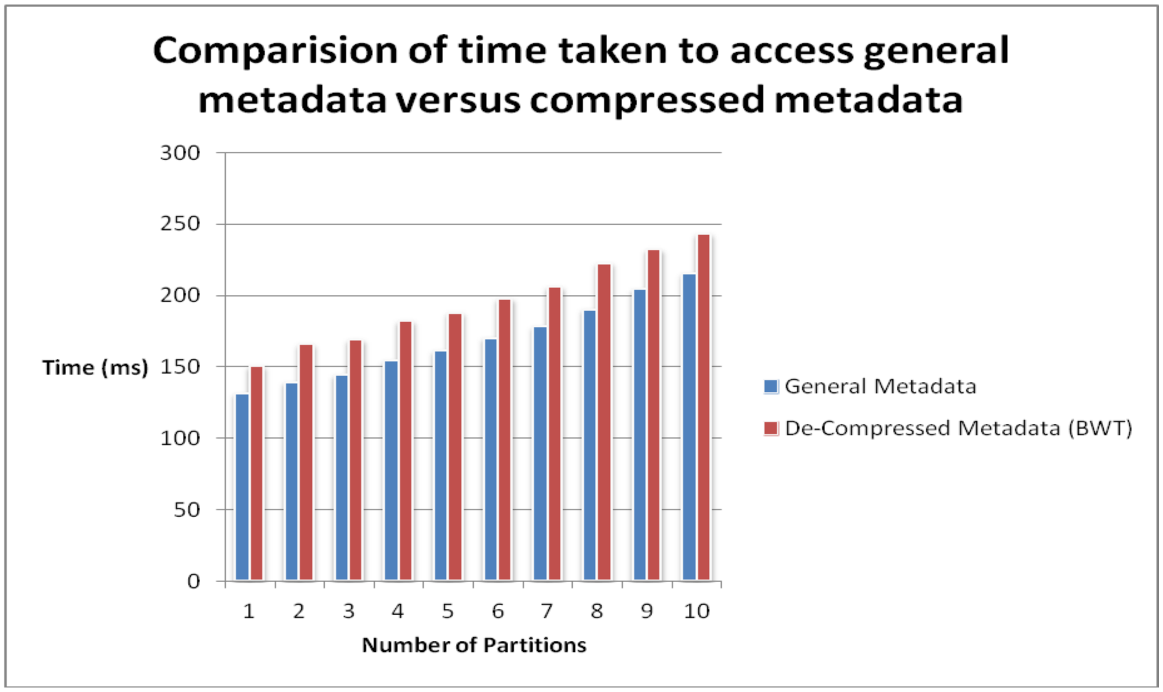


Figure 14 Graph comparing time to access general metadata versus compressed metadata

The X-axis of the graph is number of partitions of a database table and Y-axis is the time taken (in milliseconds) to get the metadata. The above graph compares the times taken to retrieve the general metadata with the compressed metadata. The bars in blue indicate the general time taken to retrieve the general metadata from the database and the bars in red indicate the time taken to retrieve the compressed metadata. The time taken indicated by the red bars also includes the time taken to decompress the compressed metadata.

As we already know, there are various compression algorithms available. In order to check the performance of our compression algorithm with other algorithms, the same procedure has been followed and graphs have been plotted using various other compression algorithms. The metadata files for every table that have been recorded at every partition are provided as input to the other algorithms (Huffman and Lempel-Ziv-Welch, etc). The compressed files are obtained as the outputs. The values of these output files have been recorded and the below graph has been plotted. Figure 15 shows the averages for different database tables and their compressed metadata file sizes versus the number of partitions.

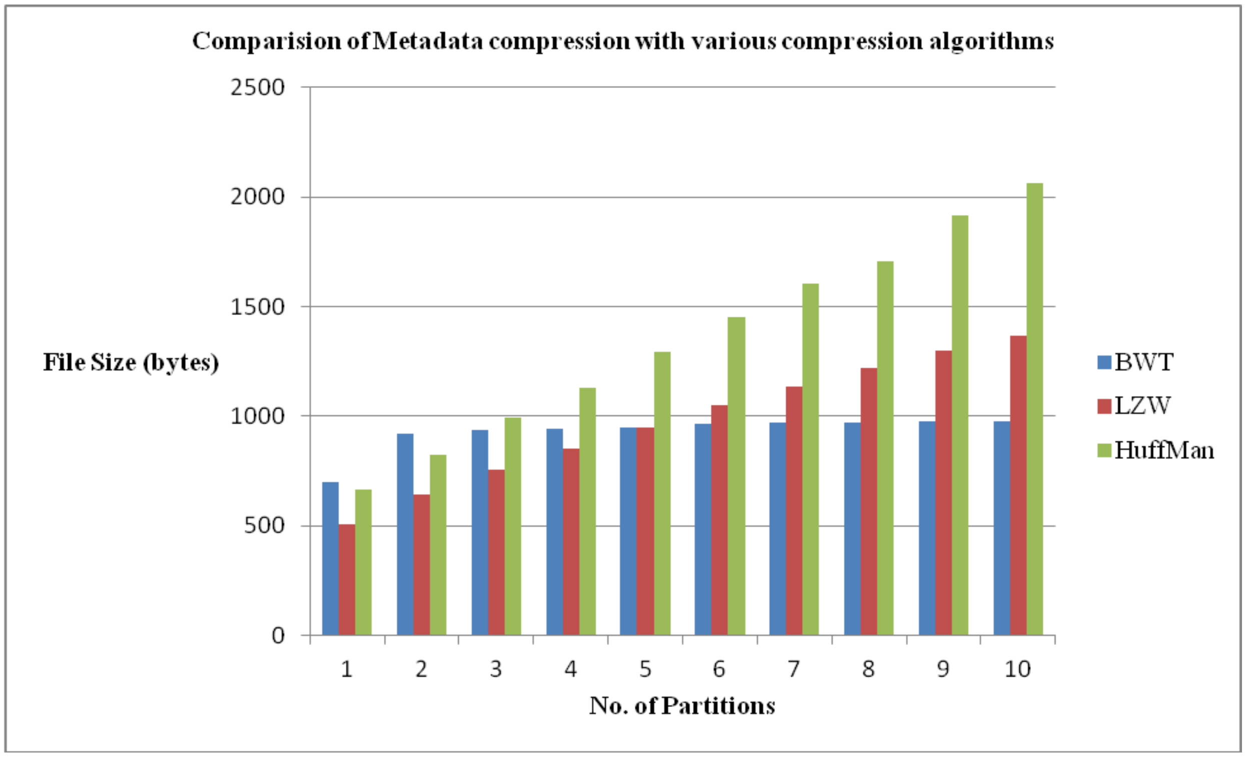


Figure 15 Graph comparing metadata compressions with various compression algorithms

The X-axis of these graphs represents the number of partitions on the various database tables while the Y-axis represents the corresponding size of the metadata file in bytes. The above graph compares the various compression algorithms with the BWT compression algorithm. This graph is plotted based on the different compression ratios of three different compression algorithms. The additional algorithms used for this purpose were LZW and Huffman. The bars in blue indicate the compressed file size of the metadata using BWT, red bars indicate the compressed file size of the metadata using LZW, and green bars indicate the compressed file size of the metadata using Huffman compression algorithm. From the above graph we observe that the LZW and Huffman are better compression algorithms than the BWT algorithm on files of small sizes, but as the size of the file increases the performance of BWT algorithm is better than the LZW and Huffman algorithms. As we are working on larger databases with huge file sizes, BWT would be the best fit.

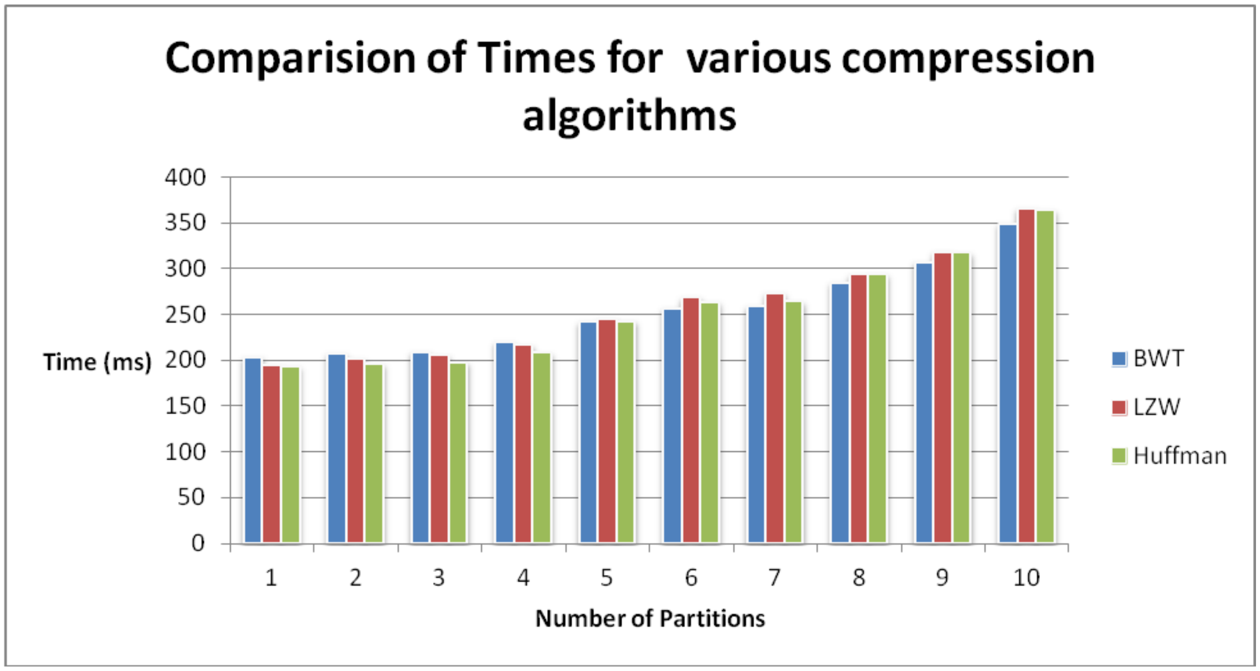


Figure 16 Graph comparing the times required for various algorithms

We also wanted to check the performance of the BWT algorithm with LZW and Huffman with respect to time. In order to check this performance, we reestablished the connection from Netbeans to the Oracle database. We executed the respective commands to get the metadata from the database table for every partition. We then recorded the times required to de-compress the desired metadata files. These de-compressed values were recorded on a laptop. Figure 16 has been plotted with the recorded times.

The X-axis of the above graph is number of partitions of a database table and Y-axis is the time taken to get the metadata. The above graph compares the various times taken by different compression algorithms with the time taken by the BWT compression algorithm. The bars in blue indicate the time taken to retrieve the metadata using BWT, red bars indicate the time taken to retrieve the metadata using LZW, and green bars indicate the time taken to retrieve the metadata using Huffman compression algorithm. The time taken indicated by the above bars also includes the decompression times of the respective compression algorithms.

CHAPTER 5

CONCLUSION

This research aims to establish a means to compress the application level metadata of low priority data present in database tables, and then store it. With the increase in data stored in the database tables, the corresponding metadata has also increased tremendously over the past decade, thereby increasing the cost for the storage of the data. Metadata improves the performance of the database but the maintenance of the metadata is very expensive. Our research work is intended to propose a solution which can be used in the current industry. In our research work we have created database tables and replicated the present day scenario using Oracle 11g database software. We have continued the research on compression algorithms that other researchers began. The research was carried out in order to find a better compression algorithm better suiting the issue at hand. Results prove that the BWT algorithm is very effective when reducing the database application level metadata for large partitions. Simulations done in the research compared various compression algorithms such as LZW and Huffman with BWT in regard to constraints such as compression ratio and speed of the algorithm. Results show that the BWT algorithm performs better when compared to the LZW and Huffman algorithms in terms of the compression ratio, de-compression time, and increase in the partition sizes. When low priority data is required for various reasons, the time taken to do so would be more than the time taken to fetch the general active data because the respective metadata must be de-compressed before the historical data can be retrieved or located. The time required for de-compression depends upon the algorithm. Our research makes use of BWT algorithm for this purpose and the results show that there is a very minimal increase in time taken, on the order of a few milliseconds. This is considered to be the flaw of this research. As this technique will lower the usage of other resources such as bandwidth, processing time, power, and costs, the limitations of this technique are negligible.

REFERENCES

REFERENCES

1. The Dublin Core metadata standard website http://www.dublincore.org/. Accessed: 09/11/2011.

2. Metadata Standards and Metadata Registries: An Overview, Bruce E. Bargmeyer http://www.bls.gov/ore/pdf/st000010.pdf. Accessed: 08/25/2011.

3. Introduction about MARC http://www.loc.gov/marc/umb/. Accessed: 09/19/2011

4. MODS Standards http://www.loc.gov/standards/mods/. Accessed: 09/19/2011

5. Marcia Lei Zeng and Jain Qin, *Metadata*: Neal Schuman, 2008, page 27.

6. CDWA- http://www.getty.edu/research/publications/electronic_publications/cdwa/index.htmll. Accessed: 08/18/2011.

7. Partitioning in Oracle Database 11g, Oracle White Paper http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-partitioning-11gr2-2011-12-1392415.pdf. Accessed: 09/21/2011.

8. Oracle Database VLDB and Partitioning Guide http://docs.oracle.com/cd/E11882_01/server.112/e25523/toc.htm. Accessed: 09/22/2011.

9. Oracle Partitioning Data Sheet. http://www.oracle.com/technetwork/database/enterprise-edition/overview/partitioning-11g-datasheet-128345.pdf?ssSourceSiteId=ocomen. Accessed: 09/24/2011.

10. ZBD: Using Transparent Compression at the Block Level to Increase Storage Space Efficiency:http://www.ics.forth.gr/~bilas/pdffiles/makatos-snapi10.pdf. Accessed: 09/24/2012.

11. Sayood, Khalid, *Introduction to Data Compression*, Second edition, Morgan Kaufmann, March 2000, page 1.

12. Sayood, Khalid, *Introduction to Data Compression*, Second edition, Morgan Kaufmann, March 2000, page 5.

13. Sayood, Khalid, *Introduction to Data Compression*, Second edition, Morgan Kaufmann, March 2000, pages 4-5.

14. H. Wenjun, W. Wenjun, X. Hui, “A Lossless Data Compression Algorithm for Real-time Database”, Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress, vol. 2, pp. 6645-6648, 2006.

15. Z. Chenggang, R. Baoqiang, “Design and Realization of data Compression in Real-time Database”, Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference, pp. 1, 2009.

16. S. Aghav, “Database Compression Techniques for Performance Optimization”, Computer Engineering and Technology (ICCET), 2010 2nd International Conference, vol. 6, pp. v6-714 – v6-717, 2010.

17. A. Rosenthal, E. Sciore, “Administering Propagated Metadata in Large, Multi-Layer database systems”, Knowledge and Data Engineering Exchange, 1999. (KDEX '99) Proceedings. 1999 Workshop, pp. 11-16, 1999.

18. L. Yonghong, D. Hangyan, D. Guilong, “A Metadata Generation Method for Massive Data in Distributed Databases”, Information Science and Engineering (ICISE), 2010 2nd International Conference, pp. 3431-3434, 2010.

19. J. Song, Y. Bao, “NPA: Increased Partitioning Approach for Massive Data in real-time Data Ware House”, Information Technology Convergence and Services (ITCS), 2010 2nd International Conference, pp. 1-6, 2010.

20. Technical Overview of CPC, http://www.cartesianinc.com/Tech/tech-overview.html. Accessed: 10/07/2011.

21. An Introduction to Fractual Image Compression by Texas Instruments Europe http://www.ti.com/lit/an/bpra065/bpra065.pdf . Accessed: 09/26/2011.

22. Wavelet Image Compression by Myung-Sin Song http://www.siue.edu/~msong/Research/article.pdf . Accessed: 09/27/2011.

23. Speex: A Free Codec for Speech, http://www.speex.org/. Accessed: 10/02/2011.

24. Improvements to Burrows-Wheeler Compression Algorithm by Juergen Abel http://docis.info/docis/lib/goti/rclis/dbl/soprex/%282000%2930%253A13%253C1465%253AITBCA%253E/www.data-compression.info%252FJuergenAbel%252FPreprints%252FPreprint_After_BWT_Stages.pdf. Accessed: 09/30/2011.

25. A Block-Sorting Lossless Data Compression algorithm by M.Burrows and David Wheeler http://marknelson.us/1996/09/01/bwt/. Accessed: 09/21/2011.