ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 11, May 2014

Treble Phase Compression for better efficiency of Columnar Database using RL-Huffman based LZW Coding

Punam Bajaj, Puneet Bagga, Rajeev Sharma
Assistant Professor M.Tech Assistant Professor

Abstract*— Data compression is now essential for applications such as transmission and storage in data bases. In this paper, we propose a method to reduce data volume stored in Columnar Database with the help of Treble Compression technique which includes combination of three lossless compression techniques. The proposed technique includes Run Length Encoding, Lempel-Ziv Welch and Huffman data compression methods which are lossless in manner. Many factors depend on compression ratio such as storage space in Hard disk, Cache locality, and transmission over media. It is a method and system used in the column oriented database for transferring data from a digital data source to a digital data receiver with refined efficiency. The double compression test based on Run Length and Huffman encoding called as RL-Huffman gives better compression result than a single compression test, we determine the case in which the triple phased Treble compression will provide a better efficiency with the help of compression ratio. The goal is achieved by using combination of three techniques on a raw Column based Database to provide better efficiency than a single compression algorithm. Data compression is now essential for applications such as transmission and storage in data bases. In this paper, we propose a method to reduce data volume stored in Columnar Database with the help of Treble Compression technique which includes combination of three lossless compression techniques. The proposed technique includes Run Length Encoding, Lempel-Ziv Welch and Huffman data compression methods which are lossless in manner. Many factors depend on compression ratio such as storage space in Hard disk, Cache locality, and transmission over media. It is a method and system used in the column oriented database for transferring data from a digital data source to a digital data receiver with refined efficiency. The double compression test based on Run Length and Huffman encoding called as RL-Huffman gives better compression result than a single compression test, we determine the case in which the triple phased Treble compression will provide a better efficiency with the help of compression ratio. The goal is achieved by using combination of three techniques on a raw Column based Database to provide better efficiency than a single compression algorithm.

***Index Terms*—Column Store, Huffman Algorithm, Lempel-Ziv Welch, Run Length Encoding, Treble Compression.**

I. INTRODUCTION

Database system performance is directly related to the efficiency of the system at storing data on primary storage (e.g., disk) and moving it into CPU registers for processing. Column-store systems completely vertically partition a database into a collection of individual columns that are stored separately. By storing each column separately on disk, these column-based systems enable queries to read just the attributes they need, rather than reading entire rows from disk and discard unneeded attributes once they are in memory. A similar benefit is true while transferring data from main memory to CPU registers, improving the overall utilization of the available I/O and memory bandwidth. Overall, taking the column-oriented approach to the extreme allows for numerous innovations in terms of database architectures. [12] In this paper, we discuss optimization of column-stores with compression with higher compression level achieving high compression rate than a single or a double compression with a minimal elapsed time.

A. Columnar Database

A **columnar** or **column-oriented DBMS** is a database management system (DBMS) that stores its content by column rather than by row.

Fig 1. Storage layout of Columnar Database

This has advantages for data warehouse and library catalogues and other ad hoc inquiry system where aggregates are computed over large numbers of similar data items. [2][9] Faced with massive data sets, a growing user population, and performance-driven service level agreements, organizations everywhere are under extreme pressure to deliver analyses faster and to more people than ever before. That means businesses need faster data warehouse performance to support rapid business decisions and added applications, and better system utilization and as data volumes continue to increase driven by everything from longer detailed histories to the need to accommodate big data companies require a solution that allows their data warehouse to run more applications and to be more responsive to changing business environments. Plus,

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 11, May 2014

they need a simple, self managing system that boosts performance but helps reduce administrative complexities and expenses. Column Oriented DBMS provides unlimited scalability, high availability and self managing administration. [2]

B. Core difference between Columnar Database and row-oriented Database

The world of relational database systems is a two-dimensional world. Data is stored in tabular data structures where rows correspond to distinct real-world entities or relationships, and columns are attributes of those entities. There is, however, a distinction between the conceptual and physical properties of database tables. This aforementioned two-dimensional property exists only at the conceptual level. At a physical level, database tables need to be mapped onto one dimensional structure before being stored. This is because common computer storage media (e.g. magnetic disks or RAM), despite ostensibly being multi-dimensional, provide only a one dimensional interface. For example, a database might have this table.

EmpId	Lastname	Firstname	Salary
1	Wilson	Joe	40000
2	Yaina	Mary	50000
3	John	Cathy	44000

Fig 2.Two Dimensional Table

This simple table includes an employee identifier (EmpId), name fields (Lastname and Firstname) and a Salary .The database must coax its two-dimensional table into a one-dimensional series of bytes, for the operating system to write it to either the RAM, or hard drive, or both. A row-oriented database serializes all of the values in a row together, then the values in the next row, and so on.

1, Wilson, Joe, 40000;

2, Yaina, Mary, 50000;

3, John, Cathy, 44000;

A column-oriented database serializes all of the values of a column together, then the values of the next column, and so on.

1, 2, 3;

Wilson, Yaina, Johnson;

Joe, Mary, Cathy;

40000, 50000, 44000;

This is a simplification. Partitioning, indexing, caching, views, OLAP cubes, and transactional systems such as write ahead logging or multi version concurrency control all dramatically affect the physical organization. [3]

C. Advantages of Columnar Database

Following are some advantages of column stores:

- **Data compression**: Storing data from same attribute together increases locality and thus data compression ratio (when attribute is sorted). Bandwidth requirements are further reduced when transferring compressed data.
- **Improved Bandwidth Utilization:** In a column-store, only those attributes are accessed by a query needed to be read from the disk (or from RAM to cache), whereas in a row –store, all attributes of a particular row are also accessed which are not needed. Since, an attribute is generally smaller than the whole row in which data is accessed.
- **Improved Code Pipelining**: Attributed data can be iterated directly without indirection through a tuple interface and results in high IPC (instruction per cycle) efficiency.
- **Improved cache locality**: In spatial locality we have cache blocks which carry data which may not be needed like surrounding attributes in row store scheme. Therefore space of cache is wasted and reduces hit rate.[3]

II. DATA COMPRESSION

Compression is a technique used by many DBMSs to increase performance. Compression improves performance by reducing the size of data on disk, decreasing seek times, increasing the data transfer rate and increasing buffer pool hit rate. One of the most-often cited advantages of Column-Stores is data compression. Intuitively, data stored in columns is more compressible than data stored in rows. Compression algorithms perform better on data with low information entropy (high data value locality). Imagine a database table containing information about customers (name, phone number, e-mail address, e-mail address, etc.). Storing data in columns allows all of the names to be stored together, all of the phone numbers together, etc. Certainly phone numbers will be more similar to each other than surrounding text fields like e-mail addresses or names. Further, if the data is sorted by one of the columns, that column will be super-compressible. Column data is of uniform type; therefore, there are some opportunities for storage size optimizations available in column-oriented data that are not available in row-oriented data. Compression is useful because it helps reduce the consumption of expensive resources, such as hard disk space or transmission bandwidth. Info bright is an example of an open source column oriented database built for high-speed reporting and analytical queries, especially against large volumes of data. [2]

A. Conventional Data Compression Methods

Two type of compression exists in real world. One is Lossless compression and another is Lossy compression. Lossless and Lossy compression are terms that describe whether or not, in the compression of a file, all original data can be recovered when the file is uncompressed. [4]

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 11, May 2014

Lossless Compression vs. Lossy Compression

Lossless compression algorithms usually exploit statistical redundancy in such a way as to represent the sender's data more concisely, but nevertheless perfectly. Lossless compression is possible because most real-world data has statistical redundancy. For example, in English text, the letter 'e' is much more common than the letter 'z', and the probability that the letter 'q' will be followed by the letter 'z' is very small. Another kind of compression, called lossy data compression, is possible if some loss of fidelity is acceptable. For example, a person viewing a picture or television video scene might not notice if some of its finest details are removed or not represented perfectly (i.e. may not even notice compression artefacts). Similarly, two clips of audio may be perceived as the same to a listener even though one is missing details found in the other. Lossy data compression algorithms introduce relatively minor differences and represent the picture, video, or audio using fewer bits. Lossless compression schemes are reversible so that the original data can be reconstructed, while lossy schemes accept some loss of data in order to achieve higher compression. However, lossless data compression algorithms will always fail to compress some files; indeed, any compression algorithm will necessarily fail to compress any data containing no discernible patterns. Attempts to compress data that has been compressed already will therefore usually result in an expansion, as will attempts to compress Encrypted data. In practice, lossy data compression will also come to a point where compressing again does not work, although an extremely lossy algorithm, which for example always removes the last byte of a file, will always compress a file up to the point where it is empty. A good example of lossless vs. lossy compression is the following string -- 888883333333. What you just saw was the string written in an uncompressed form. However, you could save space by writing it 8[5]3[7]. By saying "5 eights, 7 threes", you still have the original string, just written in a smaller form. In a lossy system, using 83 instead, you cannot get the original data back (at the benefit of a smaller file size). [4]

III. REQUIRED DATA COMPRESSION TECHNIQUES

A. RLE Coding

Run-length encoding (RLE) is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs: for example, simple graphic images such as icons, line drawings, and animations. [1] It can be explained with an example. Consider a screen containing plain black text on a solid white background. There will be many long runs of white pixels in the blank space, and many short runs of black pixels within the text. Let us take a hypothetical single scan line, with B representing a black word and W white blank space of page:

WWWWWWBWBBBWWW

If we apply the run-length encoding (RLE) data compression algorithm to the above hypothetical scan line, we get the following:

6[W]1[B]1[W]3[B]3[W]

B. Huffman's coding

Huffman codes give an efficient encoding for a list of symbols to be transmitted, when we know their probabilities of occurrence in the messages to be encoded. More likely symbols should have shorter encodings; less likely symbols should have longer encodings. If we draw some variable-length code as a code tree, we'll get some insight into how the encoding algorithm should work: To encode a symbol using the tree, start at the root and traverse the tree until you reach the symbol to be encoded—the encoding is the concatenation of the branch labels in the order the branches were visited. The destination node, which is always a "leaf" node for an instantaneous or prefix-free code, determines the path, and hence the encoding. So B is encoded as 0, C is encoded as 110, and so on. Decoding complements the process, in that now the path (codeword) determines the symbol, as described in the previous section. So 111100 is decoded as: 111→ D, 10→ A, 0→ B.

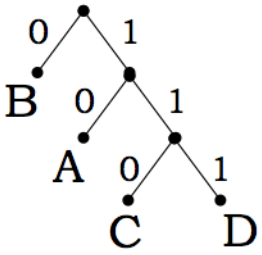

Fig 3.Code (Variable-length) shown in the form of a code tree.

Looking at the tree, we see that the more probable symbols (e.g., B) are near the root of the tree and so have short encodings, while less-probable symbols (e.g., C or D) are further down and so have longer encodings. David Huffman used this observation while writing a term paper in 1951 to devise an algorithm for building the decoding tree for an optimal variable-length code. Huffman's insight was to build the decoding tree bottom up, starting with the least probable symbols and applying a greedy strategy. Here are the steps involved, along with a worked example based on the variable-length code. The input to the algorithm is a set of symbols and their respective probabilities of occurrence. The output is the code tree, from which one can read off the codeword corresponding to each symbol.

C. LZW Coding

This method was developed originally by Ziv and Lempel, and subsequently improved by Welch. As the message to be encoded is processed, the LZW algorithm builds a string table that maps symbol sequences to/from an N-bit index. The

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 11, May 2014

string table has 2N entries and the transmitted code can be used at the decoder as an index into the string table to retrieve the corresponding original symbol sequence. The sequences stored in the table can be arbitrarily long. The algorithm is designed so that the string table can be reconstructed by the decoder based on information in the encoded stream—the table, while central to the encoding and decoding process, is never transmitted. This property is crucial to the understanding of the LZW method. When encoding a byte stream,2 the first 28 = 256 entries of the string table, numbered 0 through 255, are initialized to hold all the possible one-byte sequences. The other entries will be filled in as the message byte stream is processed. The encoding strategy works as follows and is shown in pseudo-code form in Figure 3-4. First, accumulate message bytes as long as the accumulated sequences appear as some entry in the string table. At some point, appending the next byte b to the accumulated sequence S would create a sequence S + b that's not in the string table, where + denotes appending b to S. The encoder then executes the following steps:

(1) It transmits the N-bit code for the sequence S.

(2) It adds a new entry to the string table for S + b. If the encoder finds the table full when it goes to add an entry, it reinitializes the table before the addition is made.

(3) It resets S to contain only the byte b.

This process repeats until the entire message bytes are consumed, at which point the encoder makes a final transmission of the N-bit code for the current sequence S. Note that for every transmission done by the encoder, the encoder makes a new entry in the string table. With a little cleverness, the decoder, shown in pseudo-code form in Figure can figure out what the new entry must have been as it receives each N-bit code. With a duplicate string table at the decoder constructed as the algorithm progresses at the decoder, it is possible to recover the original message: just use the received N-bit code as index into the decoder's string table to retrieve the original sequence of message bytes.

IV. TREBLE COMPRESSION TECHNIQUE

In this technique we employed combination three different algorithms known as Run Length Encoding, Huffman and LZW coding in same sequence during compression. The technique worked in three phases. The phases will compress the file again and again to achieve better compression ratio.

A. Proposed Compression Method

The proposed compression method can be explained by the block diagram.

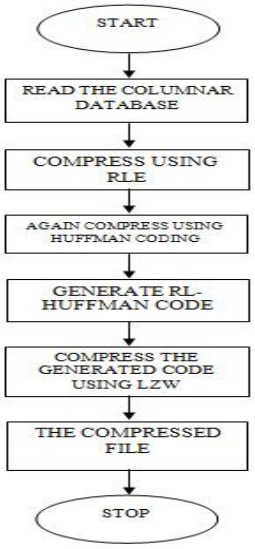

Fig 4.Compression using Treble Compression

Encoding Steps.

1. Create a Columnar Database.
2. Read the required column needed to be compressed.
3. Compress the data using Run Length and Huffman encoding to generate RL-Huffman code.
4. The Code will again be compressed by using LZW encoding to obtain compressed image.
5. Calculate Compression Ratio, and Elapsed time from compressed file.

B. Proposed Decompression Method

The original image can be obtained by decompression process.

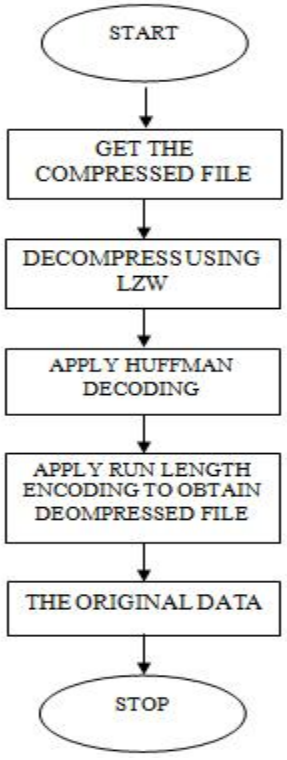

Fig 5.Decompression using Treble compression

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 11, May 2014

Decoding Steps.

1. Get the compressed file.
2. Apply LZW decompression to obtain RL-Huffman code.
3. Apply Huffman Decoding to get RLE Compressed Code.
4. Now apply Run Length Encoding resulting in Original Column oriented Database.

V. EXPERIMENTATIONS AND RESULTS

During Experimentation and result of algorithms we used MATLAB software on a redundant Column Oriented database and applied RLE, Huffman coding, Lempel-Ziv Welch and Treble Phase Compression algorithm on that database. Calculations on all algorithms done are

A. Run Length Encoding

Run-length encoding (RLE) is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run.

(1)The filling process starts from most significant character or bit

(2)Count the "1" until next "0".

(3)In the next step all the places are counted as zeros until we get next "1".

(4)The Process Continues until all the bits are counted and the number is placed with the corresponding bit.

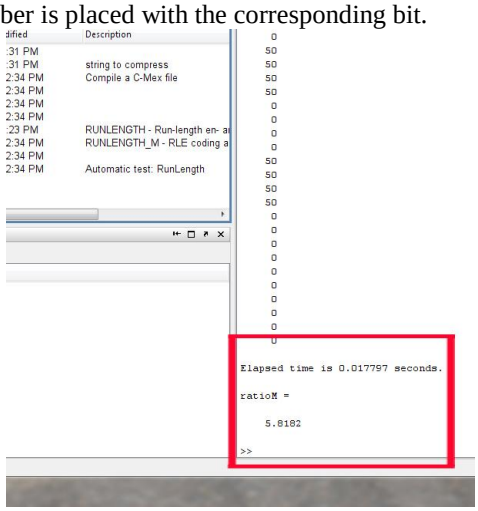

Fig 6.MATLAB result of RLE compression

The result from RLE compression brought us Compression Ratio of 5.8182 and Elapsed time of 0.177797 as shown in Figure 6.

B. Huffman Encoding

In computer science and information theory, Huffman coding is an entropy encoding algorithm used for lossless data compression. The term refers to the use of a variable length code table for encoding a source symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol. It was developed by David A. Huffman while he was a Ph.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes."

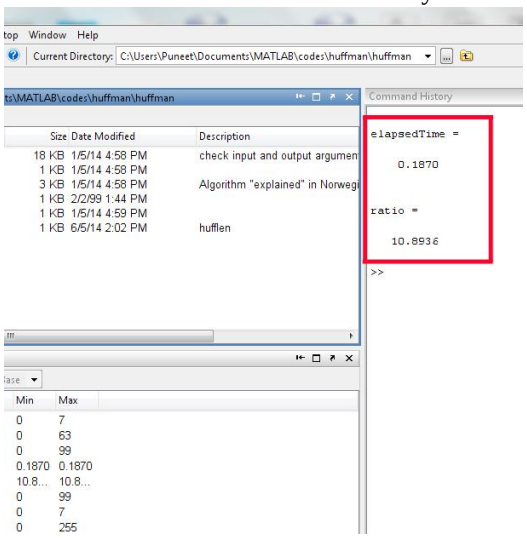

Fig 7.MATLAB result of Huffman coding

The result from Huffman coding brought us Compression Ratio percentage of 10.8936 and Elapsed time of 0.1870 as shown in Figure 7.

ALGORITHM 1. HUFFMAN ENCODING

Huffman (W, n) //Here, W means weight and n is the no. of inputs

1. **Input:** A list W of n (Positive) Weights.
2. **Output:** An Extended Binary Tree T with Weights Taken from W that gives the minimum weighted path length.
3. **Procedure:** Create list F from singleton trees formed from elements of W.
4. **While** (F has more than 1 element) **do**

   (a)Find T1, T2 in F that have minimum values associated with their roots // T1 and T2 are sub tree

   (b)Construct new tree T by creating a new node and setting T1 and T2 as its children

5.Let, the sum of the values associated with the roots of T1 and T2 be associated with the root of T Add T to F

C. PHASE 3: LZW Coding (Lempel-Ziv Welch)

LZW compression is named after its developers, A. Lempel and J. Ziv, with later modifications by Terry A. Welch. It is the foremost technique for general purpose data compression due to its simplicity and versatility. Typically, you can expect LZW to compress text, executable code, and similar data files to about one-half their original size. LZW also performs well when presented with extremely redundant data files, such as tabulated numbers, computer LZW is the basis of several personal computer utilities that claim to "double the capacity of your hard drive." If the codeword length is not sufficiently

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 11, May 2014

large, Lempel-Ziv codes may also rise slowly to reasonable efficiency, maintain good performance briefly, and fail to make any gains once source code, and acquired signals. Compression ratios of 5:1 are common for these cases.

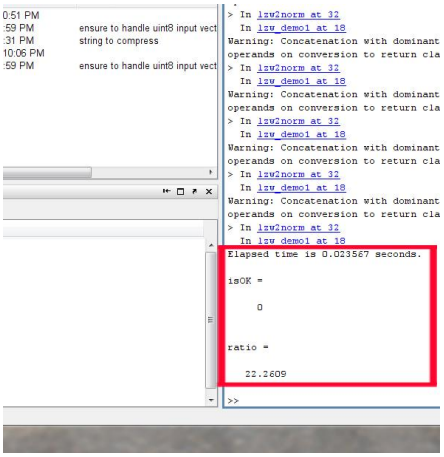

Fig 8.MATLAB result of LZW coding

The result from LZW coding brought us Compression Ratio percentage of **22.2609** and Elapsed time of **0.023567** as shown in Figure 7.

ALGORITHM 2. LZW ENCODING

1. At the start, the dictionary contains all possible roots, and P is empty;

2. C: = next character in the char stream;

3. is the string P+C present in the dictionary?

(a) if it is, P := P+C *(extend P with C)*;

(b) if not, output the code word which denotes P to the code stream;

Add the string P+C to the dictionary;

P = C (P now contains only the character C);

(c) Are there more characters in the char stream?

if yes, go back to step 2;

if not:

4. Output the code word which denotes P to the code stream;

5. END.

Treble Compression Encoding

Treble Compression compresses the database using all above algorithms. The sequence of compression used started with RLE to Huffman Encoding to LZW Encoding. The result after compression of columnar database with treble compression brought us a surprising elapsed time of **0.1944** and compression ratio percentage of **55.6000**.

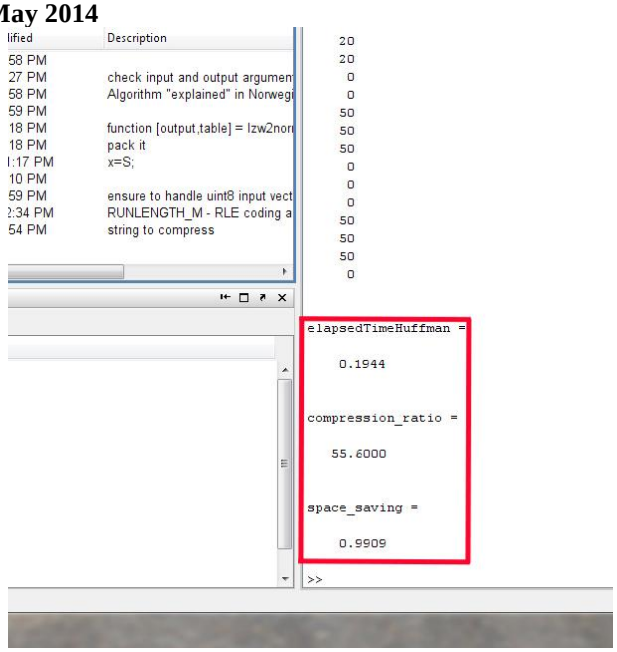

Fig 9.MATLAB result of Treble Phase coding

Table 1. Results of compression and elapsed time of all algorithmns

Algorithm	Compressed Percentage	Elapsed time (Seconds)
Original text	0	N.A.
RLE	5.8182	0.177797
Huffman Encoding	10.8936	0.1870
LZW Encoding	22.2609	0.023567
Treble Phase Compression	55.6000	0.1944

From above results we conclude that Treble compression brought us a high percentage of compression than other lossless algorithms with a minimal difference between their elapsed time. Hence Treble Phase Compression proved to be very efficient for columnar databases.

VI. ACKNOWLEDGMENT

Foremost of all, I express my sincere indebtedness to the “Almighty”, for bestowing me with all the favourable circumstances and kept me in high spirits, I am very great full to my advisor and co author, Mrs.Punam Bajaj for her patience, encouragement, and many fruitful discussions from the very early stage of this project. I would like to thank Dr Neeraj, Head of Department of Computer Science and Engineering, Chandigarh Engineering College, Landran. I would like to thank again Mrs Punam Bajaj (Assistant Professor), for her great help during research work. Finally, I wish to express my appreciation to my family for my continuous love and encouragement, for always believing in me, for never failing to provide all the support, and for coping with the pressure that naturally comes with such endeavour.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 11, May 2014

REFERENCES

[1] http:// en.wikipedia.org/

[2] Punam Bajaj, Simranjit Kaur Dhindsa, A Vision towards Column-Oriented Databases, International Journal of Engineering Research & Technology (IJERT), Vol. 1 Issue 4, June – 2012, ISSN: 2278-0181.

[3] Punam Bajaj, Simranjit Kaur Dhindsa, A Comparative Study of Database Systems, International Journal of Engineering and Innovative Technology Volume 1, Issue 6, June 2012.

[4] Mohd. Faisal Muqtida, Raju Singh Kushwaha, Improvement of compression efficiency of Huffman Coding, International Journal of Computer applications (0975-8887) Volume 45-No. 24 May, 2012.

[5] Compression Algorithms: Huffman and Lempel-Ziv-Welch (LZW) MIT 6.02 DRAFT Lecture Notes February 13, 2012

[6] MEHRDAD NOURANI and MOHAMMAD H. TEHRANIPOUR, RL-Huffman Encoding for Test Compression and Power Reduction in Scan Applications, ACM Transactions on Design Automation of Electronic Systems, Vol. 10, No. 1, January 2005.

[7] Ramez Elmasri, Shamkant B. Navathe "Fundamentals of Database Systems", 5th ed., Pearson Education inc., 2008.

[8] C-store: A column-oriented DBMS, Stone breaker et al., Proceedings of the 31st VLDB conference, Trondheim, Norway, 2005.

[9] D.J. Abadi, S.R. Madden, N. Hachem, Column Stores vs. Row Stores: How Different Are They Really?, in: SIGMOD'08,2008, pp.967-980.

[10] William McKnight, Best Practises on the use of columnar databases in: Calpont Corporation, Texas, 2011.

[11] Alan Halverson, Jennifer L. Beckmann, Jeffrey F. Naughton, A Comparison of C-Store and Row-Store in a Common Framework David J. DeWitt, Proceedings of the 32nd VLDB Conference, Seoul, Korea, 2006

[12] Daniel Abadi, Peter Boncz, Stavros Harizopoulos, The Design and Implementation of Modern Column Oriented Database, Foundations and Trends in Databases Vol. 5, No. 3 (2012) 197–280 2013 D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos and S. Madden DOI: 10.1561/1900000024

AUTHOR'S PROFILE

Punam Bajaj
Assistant Professor, Department of Computer Science Engineering, CEC Landran, Punjab, India

Puneet Bagga,
M.Tech, Department of Computer Science Engineering, CEC Landran, Punjab, India