International Journal of Computer Applications (0975 – 8887)
Volume 53– No.9, September 2012

Efficient Approach for Compression in Data Warehouse

Meenakshi Sharma
Assistant Professor
Haryana College of technology and Management,
Kaithal, India

Sonia Dora
Student
Haryana College of technology and Management,
Kaithal, India

ABSTRACT
Data compression has become most requisite and necessary part of data warehousing as it helps in saving disk space and improves query performance as well. Different compression techniques exist at different levels and each type of compression is either best from query processing point of view or compression ratio. This paper focuses on lossless compression for relational databases at attribute level. Efficient compression techniques allow transferring more data on a given bandwidth. The proposed technique in this paper is used at attribute level by compressing three types of attribute (string, integer and date type) and the most interesting feature is that it automatically identifies the type of attribute.

General Terms
Attribute Level Compression, Data type of attribute

Keywords
Attribute compression, primary key, and compression ratio

1. INTRODUCTION
As the size of databases is growing at a higher rate and due to this, the transaction overhead is also increased. Therefore, it becomes almost mandatory to manage this data in order to provide a framework for the purpose of use. This leads to development of data warehouses in which data is organized in a proper and well managed way. The data warehouse can be considered as a collection of data as well as a decision support system. The advantage of storing the data in a data warehouse is that once written data is committed it can neither be overwritten nor deleted. Thus, it relieves us from the daily transaction overhead[9]. It is very advantageous to store the data in a compressed form in data warehouse to save the disk storage space. The main reasons behind storing the data in compressed form are: - 1) Scarce of space on disk as it helps in saving disk storage. 2) A significant fall in query execution time as the data stored in data warehouse is used only for read-only purposes. 3) Improvement in flexibility and help in removing the redundancy in data.
There are four type of compression levels at which compression can be performed[7]- File level compression, Page level compression, Record level compression, Attribute level compression. All these types of compression level have both advantages and disadvantages. Both File level compression and Page level compression have a high compression ratio but from query processing perspective they are not so good to use. The main problem in these two types is that at the time of decompression the whole relation/page has to be decompressed. Thus, a lot of information is decompressed un-necessarily. On the other hand, Record level compression and Attribute level compression are much better from query processing point of view but does not have a good compression ratio in comparison to the first two types[7].

We propose the technology for compression at attribute level. All the existing techniques emphasize on compression of either numerical attributes or text attributes and all these techniques demand the details of type of attribute. We propose to define an intelligent compression scheme to be applied on Relational Database Management System (RDBMS) that automatically identifies the type of attribute and performs the compression technique accordingly. Thus, there is no need to provide with the details of type of attribute. Compression for three data types (string, integer and date time) are done in a relation. Moreover, string type attribute is compressed in two ways: with index (if required) and without indexing.

2. CLASSIFICATION OF COMPRESSION TECHNIQUES
There are many compression techniques available. Their classification can be depicted as below in figure no. 1:-

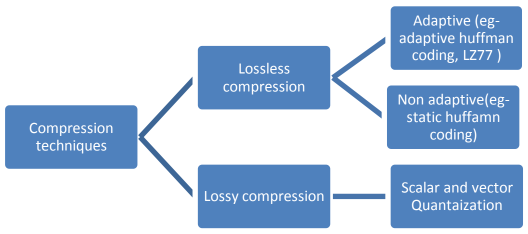
Compression techniques
Lossless compression
Lossy compression
Adaptive (eg-adaptive huffman coding, LZ77)
Non adaptive(eg-static huffamn coding)
Scalar and vector Quantaization

Figure no. 1:- Classification of compression techniques

Non-adaptive coding requires two passes: one pass to compute probabilities (or frequencies) and determine the mapping and a second pass to encode. All the adaptive methods are one-pass methods; only one scan of the message is required.

3. RELATED WORK
3.1 Page Level Compression
3.1.1 Compressing numeric attributes [3]
A page may have both numeric and non-numeric attributes. To compress numeric attributes a technique was developed in which a column containing integers were scanned to find out the lowest and highest integer value. Thus, all the values lying in between these two values are assigned the bits starting from 0. For example: - if a column on a page has 20 as minimum value and 24 as maximum then a range is defined. This range will be consisting of only 5 values and we can specify a given value in this range by using only 3 bits i.e. 20 as 000, 21 as 001, 22 as 010, 23 as 011 and 24 as 100. This minima and maxima provide a frame of reference in which all the values lie.

3.1.2 *Compressing non-numeric attributes [3]*

The same technique could be used for compression of non-numeric attributes which have low cardinality or one can say which have high redundancy in values of an attribute like gender, state or country. A high compression ratio can be achieved in such type of attributes. For example: - Gender attribute contains only two values for all the records. Therefore, only two bits are used to define its values 0 for male and 1 for female. Similarly, state and country are other attributes which have a very limited range of values

3.2 Attribute Level Compression

The main objective of this technique is to allow the reduction of the space occupied by dimension tables with high number of rows, reducing the total space occupied by the data warehouse and leading to a consequent gains on performance[8]. It is aimed to compress two different types of database attributes:-

- Text attributes with low cardinality (known as *categories*).
- Attributes of free text: comments or notes (known as *descriptions)*.

Categories[8] are those textual attributes that have a low cardinality. For example: city, country, gender, etc. Compression can be done on these types of attributes using 1 or 2 bytes. Only 1 byte is enough if value of such kind of attributes are less than 256 , and 2 bytes will be required if values are upto 65536.

A **description** is a text attribute that is mainly used for visualization. For example: description attribute is the comment attribute, which is frequently found in dimension tables. This type of attributes has the particularity of having a low access frequency and it is only necessary to decode it when the final result is being constructed.

3.2.1 *Categories Coding[8]*

Categories coding is done through the following steps:

1. The values of the attribute is analyzed and its frequency is calculated.
2. The table of codes is build based on the frequency: the most frequent values are encoded with a one byte code; the least frequent values are coded using a two bytes code. In principle, two bytes are enough, but a third byte could be used if needed.
3. The codes table and necessary metadata is written to the database.
4. The attribute is updated, replacing the original values by the corresponding codes (the compressed values).

3.2.2 *Descriptions Coding[8]*

It is very similar to the categories coding with the major difference that in this case the value in the attribute is not regarded as a single value, but as a set of values (an ASCII string). Any text compression algorithm can be used to perform this type of compression.

4. PROPOSED WORK

In this paper, a technique combining the compression techniques for three different data types (string, integer and date time) of an attribute is proposed in which there is no need to provide with the details of attribute type. The proposed algorithm automatically identifies the suitable code matching with the given attribute and performs the compression accordingly. Moreover, string type attribute is compressed in two ways: with index (if required) and without indexing. Some previous techniques focus on indexing in which a dictionary or an index is constructed for storing the codes. The constructed index itself consumes some space for storage. In proposed technique the index is constructed only if it is necessary or if it will help in reduction of space required for storing data which is calculated by finding out the frequency of values in a particular field.

4.1 How to compress string type attribute

The first step is to calculate the frequency ratio by acquiring the information from Meta data. This will help in finding whether the index is required to be created or not. If the frequency of a value for a String type attribute approaches to 50 percent of values then the index will be created otherwise it will be compressed by other way. The most repeating values will be replaced with an integer and thus these assignment codes are stored in the index. For example, the fields like state and country have most repeating values and thus for such kind of attributes, creating an index is really beneficial for space savings. For other string type attributes like name, another technique is applied. It is well known that a character takes 1 byte (8 bits) to store in memory. In the proposed technique, we have saved 1 bit for each character by applying the bitwise operators. These bitwise operators (left shift and right shift) are most efficient to use and takes less time for compression. Thus, the idea of using bitwise operators for compressing string type attributes is also helpful from the performance point of view. Each character is seen as its ASCII code and then converted into its binary code. After that, binary code of each character is left shifted by one bit. Now each character is of 7 bits but it occupies 8 bits in memory. So, the MSB of next character's binary code will be shifted to previous character binary code which is of 7 bits and will become its LSB. Thus in the same manner, the characters in the given string can be compressed. At last, it can be concluded that, a string of 26 characters can be stored in 23 bytes which would have taken 26 bytes otherwise.

4.2 How to compress Integer type attribute

In a relation, there are also attributes of integer type like ID, contact number. The compression for the integers can be performed by defining a range out of the values i.e find the maximum number (max) and minimum number(min). All the values lying between max and min are assigned the integer value starting from 1. For example: - if the ID attribute has value 2012 as minimum and 3970 as maximum then the range will contain 1888 (3900-2012) values. These 1888 values are numbered starting from 1 to 1888. In this way, the integer type field of a table can be compressed. A lot of space in memory can be saved using this method.

4.3 How to compress Date type attribute

The date type attribute has three parts: day, month and year. Number of days in a month lies from 1 to 31. So, the number of bytes required for storing day is 5. Similarly, the number of months in a year lies from 1 to 12. So, the number of bytes required for storing months is 4. For compressing the year part, we can assume a base year, say 1950. Then subtracting the given year in data of birth (say 1985) from base year and then storing the resulting value will occupy less space than the actual value. Thus, this method of date compression can save up to 8 bytes.

5. Compression/Encoding Algorithm

Table 1: Results of database compression

```
1.  Setup the connection with DBMS.
2.  Get table list from database.
3.  Repeat for each table in table list
        a.  CompressionEngine.CompressTable(T)
        b.  Update the compression statistics
4.  [end of for loop]
5.  Show the compression statistics
6.  Exit
                CompressTable(T)
1.  Create a file to store compressed table
2.  Get table , metadata from database
3.  Setup temporary indexes of those field having
    maximum repeated values.
4.  Get the recordset of table into datareader.
5.  Repeat for each record R from datareader
6.  Repeat for each field value V from record R
                        if (datatype of V is "string")
                            if ( field is indexed)
                                R=getindex(V)
                            else
                                R=CompressString(V)
                          else
                          if (datatype  of V is "int")
                            R=CompressNumber(V)
                        else
                          if (datatype of V is "Date")
                         R=CompressDate(V)
7.  Save R into compressed file
8.  [end of if]
9.  [end of for]
10. exit
```

6. RESULTS AND DISCUSSION

The sample database in MS SQL was created for evaluating the result of compression technique. Several tables can be created and can be compressed using the proposed technique. We have created only four tables named products, vendor, parts and employee. These tables contain string, date, integers type attributes. Other fields are also included like emailid, address which are compressed by considering them as string type. After applying the compression on these tables the results are shown in table 1.

The above results can be shown with the help of graph which shows the comparison of tables before compression and after compression.

Figure 1: Depiction of results

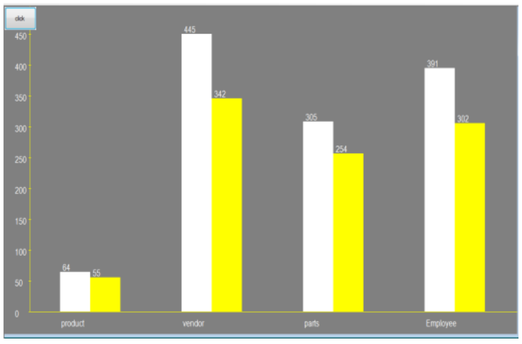

Table Name	**Initial size (in bytes)**	**Size after compression (bytes)**	**Gain (%)**	**Compression ratio(%)**
Product	64	55	14.06	85.94
vendor	445	342	23.14	76.85
Parts	305	254	16.44	83.28
Employe	391	302	22.76	77.24
Total	1205	953	20.9%	79.09%

7. CONCLUSION AND FUTURE WORK

At last, it is concluded that an intelligent compression technique has been proposed that automatically identifies the type of attribute in a relation and performs the appropriate compression on it. Compression at attribute-level is done for three types of data types. This compression technique can be applied to multi databases. One can see the status of the table being compressed i.e. which table is currently being compressed out of many tables. The obtained results show that an overall compression ratio of 79.09% has been achieved and gain in performance (saving in space) has increased to 20.91%. In the proposed work, we have considered only three data types but there are so many other data types like Boolean, short int, long int etc which can be compressed. Moreover, the compression of primary key and foreign key is also a main area for research as it contains all the unique values.

8. REFERENCES

[1] J.Ziv and A. Lempel "A universal algorithm for sequential data compression" in IEEE transactions in information theory, vol. 3, no. 3, pp. 337-343(1977)

[2] Amy Turske McNee "The Evolutionary Data Warehouse--An Object-Oriented Approach" (2008)

[3] Jonathan Goldstein, Raghu Ramakrishnan ,Uri Shaft "Compressing relations and indexes "in IEEE computer society Washington,USA (1998)

[4] P. O'Neil and D. Quass. "Improved query performance with variant indexes," Proceedings of the ACM SIG\IOD Conference, Tucson, Arizona, May, 1997.

[5] S. Chaudhuri and U. Dyal, "An overview of Data warehousing and OLAP Technology", ACM SIGMOD record, vol 21, no.1.

[6] W. Kim, "Modern Database Systems", ACM Press. New York, New York, 1995

[7] G. Ray, J. R. Haritsa, and S. Seshadri. Database compression: A performance enhancement tool. In International Conference on Management of Data, pages 0, 1995.

[8] Jorge Vieira1, Jorge Bernardino2, Henrique Madeira3 "Efficient compression of text attributes of data warehouse dimensions" (2005)

[9] Akanksha Baid and Swetha Krishnan" Binary Encoded Attribute-Pairing Technique for Database Compression"(2008)

[10] Goetz Graefe and Leonard D. Shapiro" Data Compression and Database Performance"(1991)

[11] Debra A. Lelewer and Daniel S. Hirschberg"Data Compression"(1987)