Compression Methodologies for Columnar Database Optimization

Praveen Kumar Sadineni

Mahantech Corporation, Charleston, WV, USA.

*Corresponding Author: Praveen Kumar Sadineni. Email: praveenkumarsadineni1998@gmail.com

Received: 10 January 2022; Accepted: 4 March 2022

Abstract: Today’s life is completely dependent on data. Conventional relational databases take longer to respond to queries because they are built for row-wise data storage and retrieval. Due to their efficient read and write operations to and from hard discs, which reduce the time it takes for queries to produce results, columnar databases have recently overtaken traditional databases. To execute Business Intelligence and create decision-making systems, vast amounts of data gathered from various sources are required in data warehouses, where columnar databases are primarily created. Since the data are stacked closely together, and the seek time is reduced, columnar databases perform queries more quickly. With aggregation queries to remove unnecessary data, they allow several compression techniques for faster data access. To optimise the efficiency of columnar databases, various compression approaches, including NULL Suppression, Dictionary Encoding, Run Length Encoding, Bit Vector Encoding, and Lempel Ziv Encoding, are discussed in this work. Database operations are conducted on the compressed data to demonstrate the decrease in memory needs and speed improvements.

Keywords: Columnar Databases; Compression; Query Execution; Aggregates; Performance Optimisation.

1 Introduction

Relational Database Management System (RDBMS) is a traditional database which has grabbed our attention in the last few decades and stores data arranged in a tabular fashion in rows and columns, rows indicating entities and columns indicating entity attributes. For example, an organisation database shall have an employee table to stock data of its employees. Each row holds data of various employees, and every column holds a specific attribute such as an employee’s name, address, email etc. Traditional relational databases are row-based databases which store data in rows [1-8]. The value of each record is successively stored in a lengthy row. For example, the First row contains Employee 1 data such as empID, empName, Dept, Salary, DoB etc. Then, the next row contains the information about Employee 2, shown in Fig 1. Relational databases are well-suited for transactional applications. Database tables can hold the data in two ways: row-wise [8] or column-wise [8]. Row by row method holds the entire information about an entity, whereas column by column method holds the data about an attribute. For example, the employee table stores records of employee1, 2, … n row-wise and names, addresses, dept etc., column-wise. A suitable method can be chosen based on the requirements of the performance. For entity-based access, the row-by-row method is preferred; for attribute access, the column-by-column method is preferred. Consider an organisation having n number of employees and the table employee with different attributes such as empID, empName, Dept, Salary, DoB, etc. (refer to Table 1).
OLTP (Online Transaction Processing) applications store the particulars of employees row-wise on the disk, with each field neighbouring to the subsequent in the same block on the hard disk, as shown in Table 1a.

Table 1a: Employee Table with attributes and records

<table>
<thead>
<tr>
<th>empID</th>
<th>empName</th>
<th>Dept</th>
<th>Salary</th>
<th>DoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>Anush</td>
<td>CSE</td>
<td>40000</td>
<td>01-05-1984</td>
</tr>
<tr>
<td>112</td>
<td>Karan</td>
<td>CSE</td>
<td>45000</td>
<td>03-07-1976</td>
</tr>
<tr>
<td>113</td>
<td>Anuj</td>
<td>ISE</td>
<td>45000</td>
<td>05-06-1981</td>
</tr>
</tbody>
</table>

Sometimes entire information is not required to make useful decisions, and rather few is enough called OLAP (Online Analytical Processing) [3] which do not require row-wise access. The needed data can be extracted by querying a specific column [1]. For analytic analysis [3], storing the information in a columnar database is better, mainly designed for data warehousing and big data processing.

The relational database is optimised for row data storage and meant for Online Transaction Processing applications (OLTP). In contrast, a columnar database is improved for quick data extraction by columns, normally in Online Analytical Processing (OLAP) applications [3, 9] shown in Fig 2.

Figure 2: OLTP and OLAP Databases

Columnar databases enhance the performance of analytic queries because it radically lessens the overall disk I/O requests and decreases the quantity of information needed to load from the disk. Table 2 highlights
the differences between columnar and row-based databases [10-13].

**Table 2: Differences between the Columnar and Row based Databases**

<table>
<thead>
<tr>
<th>Operations</th>
<th>Columnar Database</th>
<th>Row-based database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate functions</td>
<td>Quick</td>
<td>Slow</td>
</tr>
<tr>
<td>Data Retrieval</td>
<td>Quick with columns retrieval</td>
<td>Has to remove irrelevant data</td>
</tr>
<tr>
<td>Compression Techniques</td>
<td>Effective as similar data are bounded together</td>
<td>-</td>
</tr>
<tr>
<td>Insertion/Updation</td>
<td>Slow</td>
<td>Quick</td>
</tr>
</tbody>
</table>

**Properties of Analytic Applications**

- Less predictable: Analytic queries are experimental in behaviour and are originated by experts who generate ad-hoc queries repetitively.
- Extended life: Analytical queries perform more read operations on data to extract information in aggregate rather than specific records.
- Focus more on reading operations: Analytic applications are read-oriented than written.
- Focus on Features (attributes): Data warehouse queries tend to read several objects and review or aggregate them. The emphasis is on a specific set of attributes rather than all attributes at a time.

Analytical queries support aggregates, joins, scans and other operations used to perform data access computations on the data present in the database. Table 3 shows a few sample queries differentiating transactional and analytical operations.

**Table 3: Examples of Analytical and Transactional Queries**

<table>
<thead>
<tr>
<th>Analytical Queries (OLAP)</th>
<th>Transactional Queries (OLTP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT SUM(SALARY) FROM EMPLOYEE</td>
<td>SELECT * FROM EMPLOYEE WHERE SSN=4321</td>
</tr>
<tr>
<td>SELECT ORDERVALUE FROM ORDERS WHERE STATE='KARNATAKA'</td>
<td>INSERT INTO ORDERS VALUES (...) WHERE ORDENUM=121</td>
</tr>
<tr>
<td>SELECT COUNT (*) FROM EMPLOYEE WHERE SALARY &gt; 75000 AND SALARY &lt;150000</td>
<td>UPDATE EMPLOYEE SET</td>
</tr>
<tr>
<td>SELECT MIN(ORDERVALUE) FROM ORDERS WHERE STATE='KARNATAKA'</td>
<td>SALARY=SALARY+10000 WHERE SSN=4321</td>
</tr>
</tbody>
</table>

RDBMS such as Oracle, IBM DB2, and Microsoft SQL servers use a row-wise storing strategy. They are designed to automate sensitive business tasks. Earlier, for banking applications, these databases were used to fetch the details of the customers and transactions and later, for detailed analytical queries such as a query to analyse the entire data to discover an association between the attributes of a customer and risks of loan, it started to use database systems for decision making and planning. This led to two problems. First, analytical queries are time-consuming, and the tinier transactional write queries must wait until the analytical queries are over. Second, the analytical queries did not usually process the same facts as the transactional queries since both working and past facts are appropriate for making decisions. Therefore, businesses are inclined to build two databases where transactional queries are taken care of by transactional databases, and analytical queries [3] are taken care of by data warehouses.

### 1.1. Columnar Database

Columnar databases [2] are the future of data warehouse intelligence which builds data repositories for businesses to support decision-making systems. Data warehouses profit from the advanced performance they can obtain from a database that stores data by column rather than by row. A columnar database [5] is meant for analytical query processing and stores data column-wise. It is well-suited for data warehouse
applications. MonetDB, C-Store, MariaDB, ClickHouse, Apache Kudu, and Apache HBase are some of the columnar database systems. Columnar databases are better for Queries that involve fewer columns, aggregation queries against a huge volume of data and Column-wise compression. Columnar databases [2] perform operations faster. Because storage devices have to fetch data from hard disk drives, which retain data magnetically on spinning platters using read/write heads that move around to locate the data requested by consumers. The fewer the heads have to move, the faster the drive works. If data is retained tightly, lessening seek time, machines can quickly deliver data.

2 R Compression Techniques

This section discusses various compression techniques [4] which can be used to improve and optimise the performance [1] [6] in columnar databases.

- NULL Suppression: In this technique, successive zeros or spaces in the data are removed and exchanged with their count and where they were located. It is well suited for datasets having more zeros or spaces. Oracle’s Byte Aligned Bitmap Codes are one of the best examples. It is well suited for tables with scarce data.

- Dictionary Encoding: This technique replaces recurrent patterns with minor codes. It builds a table which maps unique values for an attribute to a dictionary ID. It can be organised by frequency or lexicographical ordering. In a dictionary, it holds a 32-bit id of its value in place of the real value. Fig 3 shows an example of this technique.

- Run Length Encoding (RLE): RLE [9] packs run of the same value in a single column into triplets where each part shown in Fig 4 is given a static number of bits, which are:
  - Attribute Value
It requires expertise to sort the columns to maximise the compression.

- **Bit vector Encoding**: This technique suits columns with fewer unique data values. For every unique value \( v_i \) in column \( C_i \), it creates a bit vector \( B \) such that \( B[i]=1 \) if \( C[i]=v_i \) as shown in Fig 5. Bit vector can be compressed further if it is sparse. As shown in Fig 5, value \( v_1 \) with item ID 1 is set to 1 and the rest all set to 0, followed by value 2 with item ID 2 is set to 1, making the rest all values 0 and so on.

  ![Figure 5: Bit Vector Encoding](image)

- **Lempel Ziv Encoding** is a broadly used method for lossless file compression. It makes use of a UNIX command, `gzip`. It replaces variable-sized patterns with static length codes. This is different from Huffman encoding, which generates flexible-sized codes. Lempel Ziv encoding does not need the awareness about pattern occurrences in prior. It dynamically constructs the pattern table as it encrypts the data. The simple idea is to parse the input data arrangement into non-overlapping chunks of varying lengths while building a dictionary of chunks. Pointer exchanges succeeding presences of these chunks to a former occurrence of the same chunk.

2.1. Aggregate Query

An aggregate query [7][10] is a method of the deriving group and subgroup data by analysing a data set in the database. Database developers and administrators use this query to tell the database how to group so that data can be analysed and used.

Following are some of the common aggregate queries.

- Compute the average value from a numeric arrangement
- Count the number of elements in a sequence
- Count the count of entries in a column/table
- Search for the max value in a numeric arrangement
- Search for the min value in a numeric arrangement
- Compute the standard deviation of a numeric arrangement

A few examples of the equations used to compute aggregate operations are as follows.

The sum is the summation of all the data values and is computed using the formula and equation 1.

\[
\text{Sum} = \sum_{i=1}^{n} x_i
\]  

(1)

Mean is the summation of all the data values/total count of elements and is computed in equation 2.

\[
\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}
\]  

(2)

Variance is the average squared deviations from the mean calculated in equation 3.

\[
S^2 = \frac{\sum_{i=0}^{n}(x_i-\bar{x})^2}{n-1}
\]  

(3)
Standard deviation is the square root of the variance and is computed as follows in equation 4.

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}}$$  \hspace{1cm} (4)

Fig.6a and b show a query which tells you how the aggregate operation can be directly performed on compressed data in columnar databases.

**Figure 6a:** Operations on the compressed data

**Figure 6b:** Operations on the compressed data

### 3 Experiments and Results

This paper discusses six compression techniques to enhance columnar databases’ performance. The compression ratio to be achieved depends on the properties of the arrangement of the values to be encoded. Table 4 shows the efficiency of various compression techniques on various data types, which helps us select...
suitable compression techniques. The selection of compression technique depends on the queries which use this data.

### Table 4: Selection of Compression Techniques

<table>
<thead>
<tr>
<th>Properties on Values</th>
<th>Less Values</th>
<th>More Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runs of repeated values</td>
<td>RLE encoding</td>
<td>RLE encoding</td>
</tr>
<tr>
<td>Sorted</td>
<td>RLE encoding</td>
<td>Lempel Ziv encoding</td>
</tr>
<tr>
<td>Unsorted</td>
<td>Dictionary encoding</td>
<td>Lempel Ziv encoding</td>
</tr>
<tr>
<td>Big values</td>
<td>Dictionary encoding</td>
<td>Lempel Ziv encoding</td>
</tr>
<tr>
<td>Small values</td>
<td>Null Suppression, Dictionary encoding</td>
<td>Null Suppression</td>
</tr>
</tbody>
</table>

The experiment is carried out with an aggregation query on one data column encoded using six compression techniques. The query used is as follows.

**SELECT SUM(C1) FROM TABLE GROUP BY C1**

The column to be aggregating has 30000 32-bit integer values, and an assumption is made that sorted runs are of size X. Suppose column C1 is sorted. The column1 in the projection has 500 unique values, and column2 has 1000 unique values, then C1 will have an average sorted run of size \( \frac{100000000}{500 \times 1000} = 200 \).

The experimentation is carried out with a column with sorted runs of size 40, shown in Fig 7. The data compression is done using all the techniques described, such as NULL Suppression, Run Length Encoding, Bit Vector, Dictionary, and Lempel Ziv Encoding. Compressed column sizes are mapped to several distinct values. The compressed column sizes are shown in Fig 7a and b for a diverse number of distinct values known as Cardinality. Dictionary and Lempel Ziv achieve the highest compression ratio, Run Length Encoding and Dictionary Encoding with better compression ratio with increasing lengths of runs of repeated values and Bit vector resulting linear compression ratio.

**Figure 7a & 7b: Sizes of compressed columns for different compression techniques on columns with organised runs of size 50 and 1000**

### 4 Conclusion

Data is compressed in the databases during query processing to optimise disk space storage and I/O operations. This paper discusses various data compression techniques such as NULL Suppression, Dictionary Encoding, Run Length Encoding, Bit Vector Encoding and Lempel Ziv Encoding to achieve performance enhancements in columnar databases. Experiments are carried out using these techniques to
reduce disk space and memory needs and improve I/O performance by letting database operators directly operate on the compressed data resulting in varied compression ratios.

Acknowledgments: The author thanks Mahantech Corporation, Charleston, USA, for their continued support.

Funding Statement: The author(s) received no specific funding for this study.

Availability of Data and Materials: The data used to support the findings of this study can be obtained from the corresponding author upon request.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References


This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium provided the original work is properly cited.