P- ISSN 1991-8941 E-ISSN 2706-6703 Journal of University of Anbar for Pure Science (JUAPS) Open Access
2008,(2), (1) :169-178

BITMAP INDEX: A DATA STRUCTURE FOR FAST FILE RETRIEVAL

Murtadha M. Hamad

College of Computer-Al-Anbar University

ARTICLE INFO

Received: 1 / 4 /2008
Accepted: 24 / 4 /2008
Available online: 30/4/2008
DOI: 10.37652/juaps.2008.15456

Keywords:
Bitmap, Indexes,
B-tree,
compressed keys,
Retrieval Time.

ABSTRACT

Bitmap structure indexes are usually used in database environments which have large amount of data. Bitmap reduces response time and storage requirements for large database compared to other data structures like B- tree. In this research, the Bitmap structure is studied and analyzed using Visual Foxpro-8 Software. The empirical study proved the efficiency of this structure for compressing keys.A Comparison between this structure and B-tree was done as an example to explain more advantages for Bitmap structure.

Introduction

As computers become more pervasive, many scientific and commercial endeavors are collecting or generating tremendous amount of data. Typically a relative smaller number of records contain the keys to new insight or new trends[1, 2].

Bitmap indices are a specialized type of index designed for easy querying on multiple keys, although each bitmap index is built on single key. For bitmap indices to be used, records in a relation must be numbered sequentially, starting from, say , 0 , Given a number n ,it must be easy to retrieve the record number n.

This is particularly easy to achieve if records are fixed in size, and allocated on consecutive blocks of a file. The record number can then be translated easily into a block number and a number that identifies the record within the block.

Bitmap Index Structure

A bitmap is simply an *array of bits*. In its simplest form, a bitmap index on the attribute *A* of relation *r* consists of one bitmap for each value that *A* can take. Each bitmap has as many bits as the number of records in the relation. The *i*th bit of the bitmap for value *vi* is set to 1 if the record numbered *i* has the value *vi* for attribute *A*. All other bits of the bitmap are set to 0.

In this example, there is one bitmap for the value *m* and one for *f*. The *i*th bit of the bitmap for *m* is set to 1 if the *gender* value of the record numbered *i* is m. All other bits of the bitmap for *m* are set to 0. Similarly, the bitmap for f has the value 1 for bits corresponding to records with the value *f* for the *gender* attribute; all other bits have the value 0. Figure 1 shows an example of bitmap indices on a relation *customers*. [3]

We now consider when bitmaps are useful. The simplest way of retrieving all records with value *m* (or value *f*) would be to simply read all records of the relation and select those records with value *m* (or *f*, respectively). The bitmap index doesn't really help to speed up such a selection, as shown in Appendix A.

* Corresponding author at: College of Computer-Al-Anbar University, Iraq.E-mail address: mortadha61@yahoo.com

cust-id	*name*	*gender*	*Mar_status*	*region*	*In_level*
0	ali	m	married	central	L1
1	lena	f	single	west	L2
2	noor	f	married	central	L3
3	omar	m	married	east	L4
4	salv	f	single	west	L5

Bitmaps for gender	
m	10010
f	01101

Bitmaps for in_level	
L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Figure 1. Bitmap indices on relation *customers*..

In fact, bitmap indices are useful for selections mainly when there are selections on multiple keys. Suppose we create a bitmap index on attribute in-level(income-level), which we described earlier, in addition to the bitmap index on gender.

Consider now a query that selects women with income in the range 70.000-89.999. This query can be expressed as $\sigma_{gender=f\Delta income-level=L2}(r)$. To evaluate this selection, we fetch the bitmaps for in_level value L2, and perform an intersection (logical-and) of the two bitmaps. In other words, we compute a new bitmap where bit i has value 1 if the ith bit of the two bitmaps are both 1, and has a value 0 otherwise. In the example in Figure(1), the intersection of the bitmap for gender=f (01101) and the bitmap for in_level=L2 (01000) gives the bitmap 01000.

Since the first attribute can take 2 values, and the second can take 5 values, we would expect only about 1 in 10 records, on an average, to satisfy a combined condition on the two attributes. If there are further conditions, the fraction of records satisfying all the conditions is likely to guitar small. The system can then compute the query result by finding all bits with value 1 in the intersection bitmap and retrieving the corresponding recodes. If the fraction is large, scanning the entire relation would remain the cheaper alternative.

Another important use of bitmaps is to count the number of topples satisfying a given selection. Such queries are important for data analysis. For instance, if we wish to find out how many women have an income level L2, we compute the intersection of the two bitmaps and then count the number of bits that are 1 in the intersection bitmap. We can thus get the desired result from the bitmap index, without even accessing the relation.

3. Bitmap and Degree of Cardinality

The advantages of using bitmap indexes are greatest for columns in which the ratio of the number of distinct values to the number of rows in the table is small. We refer to this ratio as the degree of cardinality. A gender column, which has only two distinct values (male and female), is optimal for a bitmap index. However, data warehouse administrators also For example, on a table with one million rows, a column with 10,000 distinct values is a candidate for a bitmap index. A bitmap index on this column can outperform a B-tree index, particularly when this column is often queried in conjunction with other indexed columns. In fact, in typical large database environments, a bitmap index can be considered for create a bitmap index on cust_id because this is a unique column. Instead, a unique B-tree index on this column provides the most efficient representation and retrieval.

Table 1 illustrates the bitmap index for the gender column in this example. It consists of two separate bitmaps, one for gender.

	gender='m'	gender='f
cust_id 70	0	1
cust_id 80	0	1
cust_id 90	1	0
cust_id 100	0	1
cust_id 110	0	1
cust_id 120	1	0
cust_id 130	1	0
cust_id 140	1	0

Each entry (or bit) in the bitmap corresponds to a single row of the customers table. The value of each bit depends upon the values of the corresponding row in the table. For example, the bitmap cust_gender='f' contains a one as its first bit because the gender is f in the first row of the customers table. The bitmap cust_gender='f' has a zero for its third bit because the gender of the third row is not f.

Query No.2 with Bitmap Index

An analyst investigating demographic trends of the company's customers might ask, "How many of our married customers have an income level of G or H?" This corresponds to the following SQL query:

SELECT COUNT(*) FROM customers

WHERE mar_status = 'married' AND in_level IN ('H: 150,000 - 169,999', 'G: 130,000 - 149,999');

Bitmap indexes can efficiently process this query by merely counting the number of ones in the bitmap illustrated in Figure ٢. The result set will be found by using bitmap or merge operations without the necessity of a conversion to rowids. To identify additional specific customer attributes that satisfy the any non-unique column. B-tree indexes are most effective for high-cardinality data: that is, for data with many possible values, such as customer name or phone number. In a data warehouse, B-tree indexes should be used only for unique columns or other columns with very high cardinalities (that is, columns that are almost unique). The majority of indexes in a data warehouse should be bitmap indexes.

In ad hoc queries and similar situations, bitmap indexes can dramatically improve query performance. AND and OR conditions in the WHERE clause of a query can be resolved quickly by performing the corresponding Boolean operations directly on the bitmaps before converting the resulting bitmap to rowids. If the resulting number of rows is small, the query can be answered quickly without resorting to a full table scan.

Query No.1 with Bitmap Index

The following table shows a portion of a company's customers table(appendix A).

SELECT cust-id, gender, mar_status, in_level FROM customers;

cust_id geder mar_status in_level

build bitmap indexes on columns with higher cardinalities [4].

70	f		L2: 70,000 - 89,999
80	f	married	L6: 150,000 - 169,999
90	m	single	L6: 150,000 - 169,999
100	f		L7: 170,000 - 189,999
110	f	married	L1: 50,000 - 69,999
120	m	single	L4: 110,000 - 129,999
130	m		L8: 190,000 - 249,999
140	m	married	L5: 130,000 - 149,999

Because gender, mar_status, and cust_in_level are all **low-cardinality** columns (there are only three possible values for marital status and region, two possible values for gender, and 8 for income level), bitmap indexes are ideal for these columns. Do not of C bits on[7]. The arithmetic we choose for the value representation, i.e., the decomposition of the value in digits according to some base, and the encoding scheme of each digit in bits are the two dimensions of the space and are analyzed below with the following algorithm:.

Attribute Value Decomposition Algorithm with Modification:

Inputs:

An attribute value v and a sequence of (n-1) numbers $<b_{n-1},b_{n-2},\ldots,b_1>$.

Outputs:

$\{B_i^{ni-1}, B_i^{ni-2}, \ldots, B_i^{0}\}$, where n_i denote the number of bitmaps in the i^{th} component of an index .

Step 1: define

$b_n = \left\lceil \dfrac{C}{\prod_{i=1}^{n-1} bi} \right\rceil$ Then v can be decomposed into a sequence of n digits $<v_n$,

vn-1,… , v1> ,(Let C denote the attribute cardinality; i.e., the number of distinct actual values of the indexed attribute.) as follows :

$$\begin{aligned} v &= V_1 \\ &= V_2 b_1 + v_1 \\ &= V_3(b_2 b_1) + v_2 b_1 + v_1 \\ &= V_4(b_3 b_2 b_1) + v_3(b_2 b_1) + v_2 b_1 + v_1 \\ &= \ldots.. \\ &= v_n\left(\prod_{j=1}^{n-1} b_j\right) + \ldots + v_i\left(\prod_{j=1}^{i-1} b_j\right) \\ &\quad + \ldots + v_2 b_1 + v_1 , \end{aligned}$$

where $v_i = V_i \bmod b_i$, $V_i = \left\lfloor \dfrac{v_{i-1}}{b_{i-1}} \right\rfloor$, $1 < i <= n$, and each digit v_i is in the range $0 <= v_i < b_i$.

criteria, we use the resulting bitmap to access the table after a bitmap to rowid conversion.

$$\begin{array}{c} \text{Status=} \\ \text{'married'} \\ 0\\1\\1\\0\\0\\1 \end{array} \;\text{AND}\; \left(\begin{array}{ccc} \text{Region=} & & \text{Region=} \\ \text{'central'} & & \text{'west'} \\ 0 & & 0 \\ 1 & & 0 \\ 0 & \text{OR} & 1 \\ 0 & & 1 \\ 1 & & 0 \\ 1 & & 0 \end{array}\right) = \begin{array}{c} \\ \\ 0\\1\\1\\0\\0\\1 \end{array} \;\text{AND}\; \begin{array}{c} \\ \\ 0\\1\\1\\1\\1\\1 \end{array} = \begin{array}{c} \\ \\ 0\\1\\1\\0\\0\\1 \end{array}$$

Figure 2. Executing a Query Using Bitmap Indexes

Bitmap Index Performance

Introduction

While the query performance issues of on-line transaction processing (OLTP) systems have been extensively studied [5] and are pretty much well-understood, the state-of-the-art for Decision Support Systems (DSS) is still evolving as indicated by the growing active research in this area [6].

Bitmap Indexes for Selection Queries with Attribute Value Decomposition

In this section, we present a framework to examine the design space of indexes for selection queries.Let C denote the attribute cardinality; i.e., the number of distinct actual values of the indexed attribute. The attribute cardinality is generally smaller than the cardinality of the attribute domain; i.e, the number of all possible values of the indexed attribute. Without loss of generality and to keep the presentation simple, we assume in this paper that the actual attribute values are consecutive integer values from 0 to C-1.

In particular, considering the Value-List index again, we observe that each attribute value is represented as a single digit (in base-C arithmetic), this digit being encoded in bits by turning exactly one out

Join index:

In addition to a bitmap index on a single table, we can create a bitmap join index, which is a bitmap index for the join of two or more tables.

This method is used to speed up specific join queries. A join index maintains the relationships between a foreign key with its matching primary keys. The specialized nature of star schemas makes join indices especially attractive for decision support [8].

We use the following example to illustrate the join index. Let us consider the two relations **((Sale))** and **((Product))** shown in Tables 1 and 2.

If we perform join on sale. Prod -id = prod-id, and recompute the result, we can obtain the join index as shown in Table 3. Note that the result shown in Table 3 has the same effect of Table 4, which represents a materialized view.

In bitmap indexing, each attribute has its own bitmap index table. Bitmap indexing reduces join, aggregation, and comparison operations to bit arithmetic. Join indexing records the joinable tuples of two or more relations from a relational database, reducing the overall cost of OLAP join operations.

Bitmapped join indexing, which combines the bitmap and join methods, can be used to further speed up OLAP query processing. Suppose we have very few products to consider, then the bitmap can be used for products. (This is a very important condition to check. It is not appropriate if there are many products.) The join index table after the bitmap technique is incorporated is shown in Table 5.

Indexing is important to materialized views for two reasons. Indexes for a materialized view reduce the cost of computation to execute an operation (Analogous to the use of an index on the key of a relation to decrease the time needed to locate a specified tuple ; indexing also reduces the cost of maintenance of the materialized views. One important

Step2: Each choice of n and sequence $<b_n , b_{n-1}, \dots ,$ $b_1>$ gives a different representation of attribute values , and therefore a different index .

Step3: An index is *well-defined* if $b_i>=2 ,1<=i <=n$ .The sequence $<b_n , b_{n-1}, \dots,$

$b_1>$ is the *base* of the index , which is in turn called a base - $<b_n , b_{n-1}, \dots ,$

b_1q > index .

Step4: If $b_n = b_{n-1}= \dots = b_1 =$ b, then the base is called *uniform* and the index is called

base-*b* for short . The index consists of *n components*

Step5: Let n_i denote the number of bitmaps in the i^{th} component of an index and

$\{B_i^{ni-1}, B_i^{ni-2}, \dots, B_i^{0}\}$ denote the (Output) collection of n_i bitmaps that form

the i^{th} component .

Numerical Example for algorithm:

Figure 3 shows a base-<3,3> Value-List index (based on the 12-record relation R) . By decomposing a single-component index into a 2-component index, the number of bitmaps has been reduced from 9 to 6, *i.e the **compression ratio** approximated to 30% of original space.*

π_A **(R)**			B_2^2	B_2^1	B_2^0	B_1^2	B_1^1	B_1^0
1	3	$1\times3+0 \rightarrow$	0	1	0	0	0	1
2	2	$0\times3+2 \rightarrow$	0	0	1	1	0	0
3	1	$0\times3+1 \rightarrow$	0	0	1	0	1	0
4	2	$0\times3+2 \rightarrow$	0	0	1	1	0	0
5	8	$2\times3+2 \rightarrow$	1	0	0	1	0	0
6	2	$0\times3+2 \rightarrow$	0	0	1	1	0	0
7	2	$0\times3+2 \rightarrow$	0	0	1	1	0	0
8	0	$0\times3+0 \rightarrow$	0	0	1	0	0	1
9	7	$2\times3+1 \rightarrow$	1	0	0	0	1	0
10	5	$1\times3+2 \rightarrow$	0	1	0	1	0	0
11	6	$2\times3+0 \rightarrow$	1	0	0	0	0	1
12	4	$1\times3+1 \rightarrow$	0	1	0	0	1	0

Figure 3 shows a base-<3,3> Value-List index(Value Decomposition Algorithm based on the 12-record relation R)

Table 5 combined join / Bitmap Indexing

Sale.prod-id	P1	P2
R1, R3 , R5 , R6	1	0
R2 , R4	0	1

Comparison study between Bitmap and B-tree structures:

The potential of using bitmap index to significantly improve the query performance did not go unnoticed by the database vendors. For example, ORACLE and Sybase IQ have implemented different bitmap indexes.

However, bitmap indexes are not as popular as B-trees because most of the existing databases products are designed initially for transactional data, where any index on data must be updated quickly as data records are modified. This is required in transactional applications, but for majority of the data analysis applications, such large database or scientific data management, the data to be queried is not modified, at least not modified frequently. In these cases, a new physical data layout and a new set of indexing techniques are more appropriate.

Through the theoretical and empirical study of the two types of index, we got differences related to the two types as in the following table:

Table 6 Differences related to the two types

	Bitmap Index	B- tree index
1.	Bitmap indexes are typically only a fraction of the size of the indexed data in the	Fully indexing a large table with a traditional B-tree index can be prohibitively expensive

problem in data warehousing is the maintenance of materialized views due to changes made in the source data. Maintenance of materialized views can be a very time consuming process. There need to be some method developed to reduce this time (one method is use of supporting views and / or the materialized of indexes).

Table 1 The sale table

Rid	Prod – id	Store – id	Date	Amount
R1	P1	C1	1	12
R2	P2	C1	1	11
R3	P1	C3	1	50
R4	P2	C2	1	8
R5	P1	C1	2	44
R6	P1	C2	2	4

Table 2 The Product Table

ID	Name	Price
P1	Bolt	10
P2	Nut	5

Table 3 Example of a join index

Product-id	Rid
P1	R1 , R3 , R5 , R6
P2	R2 , R4

Table 4 A Materialized view

Rid	Prod-id	Name	Price	Store-id	Date	Amount
R1	P1	Bolt	10	C1	1	12
R2	P2	Nut	5	C1	1	11
R3	P1	Bolt	10	C3	1	50
R4	P2	Nut	5	C2	1	8
R5	P1	Bolt	10	C1	2	44
R6	P1	Bolt	10	C2	2	4

Performance Analysis for Time Retrieval:

Through this study of Bitmap Structure , and by applying a number of queries with different sizes of the number of optional records (reaches half million records) concerned with the relation (customers) referred to in appendix (A), and when applying this on queries varying in complexity and on the cases of index , (Bitmap) and (B-tree) , the following table was obtained(Table 7) .

Table 7 Performance time for two Index types

Query No.	Selected Attribute (Field)	Time(Sec.) using Bitmap	Time(Sec.) using B-Tree
1.	Cust-id	٨,٠١	٣,٠٤
2.	Name	7.02	٤,٠٣
3.	Gender	2.04	٦,٠٠
4.	In-level	2.01	٧,٥٠

We notice from the above table that the performance is good in dealing with the index type (Bitmap) when number of distinct values used in the meant attribute in the relation is low or what when is called Degree of Cardinality is high.

And the opposite is true concerning the performance of B-tree .It is better when DoC is low,

	table.	in terms of space because the indexes can be several times larger than the data in the table.
2.	Bitmap indexes store the bitmaps in a compressed way(effective for low-cardinality data). If the number of distinct key values is small, bitmap indexes compress better and the space saving benefit compared to a B-tree index becomes even better.	B-tree indexes are most effective for high-cardinality data: that is, for data with many possible values, such as *customer_name* or *phone_number*.
3.	In many cases, it may not be necessary to index these columns in a data warehouse, because unique constraints can be maintained without an index, and because typical data warehouse queries may not work better with such indexes.In general, bitmap indexes should be more common than B-tree indexes in most data warehouse environments.	.B-tree indexes are most commonly used in a data warehouse to index unique or near-unique keys B-tree indexes are more common in environments using third normal form schemas.
4.	Bitmap indexes on partitioned tables are always local.	B-tree indexes on partitioned tables can be global or local.
5.	Implements the OLAP (On-Line Analytical Processing).	Implements the OLTP (On-Line Analytical Processing).

Reference will be made in the next paragraph (7) analytically to the differences between the two types of index.

2. J.Gray D.T .Liu, M. Nieto-Santisteb an, A. Szalay , D. Dewitt, and G. Heber. Scientific Data management in the coming decade .CT watches Quarterly, February, 2005

.3.Avi Silberschatz, Hank Korth, S. Sudarshan, Database System Concepts, p520-523, 5th ed, 2006.

4. Paul Lane, Oracle Database Data Warehousing Guide, 10g Release 1 (10.1) Part No. B10736-01, Copyright © 2001, 2003 Oracle Corporation.

5. G. Graefe. Query Evaluation Techniques for Large Databases. Computing Surveys, 25(2):73–170, 1993.

6. S. Chaudhuri and U. Dayal. An Overview of Data Warehousingand OLAP Technology. ACM SIGMOD Record, 26(1):65–74, March 1997.

7. Chee-Young and E. yannis, Bitmap Index Design and Evaluation, Department of Informatics, University of Athens, 1997.

8. Chen. Zhengxin, Data Mining and Uncertain Reasoning, p73-75, by John Wiley and Sons, 2001.

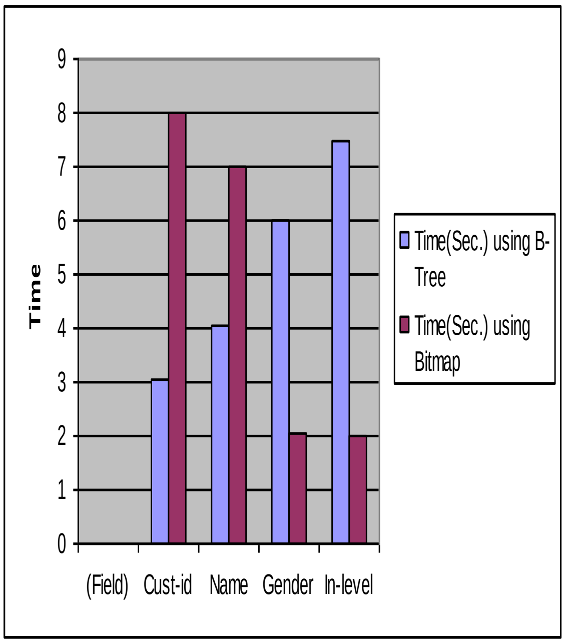

Figure 4 shows the different fields used for two Index types (Data of Table 7)

i.e. the Attribute approaches the state Unique.(Figure 4).

Conclusions and Future Works:

In this paper, we have presented a simple framework to study the implementation of bitmap indexes for selection queries, and have examined the performance of two types of indexing. The study has reached the following conclusions:

- Bitmap indexes store the bitmaps in a compressed way. If the number of distinct key values is small, bitmap indexes compress better and the space saving benefit compared to a B-tree index becomes even better. The applied algorithm (paragraph 4.2) was dealt with after it was readjusted through an arithmetic model. This led to compress the keys 30-5o% of the original space through the empirical experiment.
- The use of the logical devices NOT, XOR, AND and OR has an important role when it exists in the query in achieving better performance when dealing with the index Bitmap.Bitmap indexes are greatest for columns in which the ratio of the number of distinct values to the number of rows in the table is small.

Concerning the future works, we suggest the following:

Using association rules for data mining to improve the efficiency of execution time of Query. Implement the binary tree structure for the bitmap representation.

- Apply the concept of grouping to process the complex queries and using more statistical and mathematical tools to support the OLAP for Bitmap structure. References

1. J.Becla and D.L. Wang, Lessons Leamed from managing apetabyte, CIDR, 2005.

Appendix (A)Tables(Relations) Specifications for Optional Database

1) Table Name : Customer

Field Name	Type	Caption	Other
Cust-id	numeric (6)	Customer - id	Primary key
Name	character (50)	Customer – Name	
Gender	logical	Customer – gender	
Mar-status	character (7)	Customer – marital status	
Region	character (10)	Customer – address	
In-level	character (2)	Customer- income Level	

2) Table Name : Sale

Field Name	Type	Caption	Other
Rid	Numeric(6)	Read-id	Primary key
Prod – id	Numeric(6)	Product-id	
Stord-id	Numeric(6)	Store-id code	
Date	Date	Date of store	
Amount	Numeric(6)	Amount of product	

3) Table Name : product

Field Name	Type	Caption	Other
id	Numeric(6)	product - id	Primary key
Name	character (30)	product – Name	
Price	Float(10.3)	product – price	

P- ISSN 1991-8941 E-ISSN 2706-6703 Journal of University of Anbar for Pure Science (JUAPS) Open Access
2008,(2), (1) :169-178

هيكل بيانات الفهرسة (Bitmap) للاسترجاع السريع للملفات

مرتضى محمد حمد

Email :mortadha61@yahoo.com

الخلاصة

يستعمل هيكل البيانات (Bitmap) في البيئات ذاتالكميات الهائلة من البيانات والتي تتعامل مع قواعد البيانات. يقوم هذا الهيكل بتقليل وقت الاستجابة ومتطلبات االتخزين مقارنة مع هياكل بيانية اخرى مثل (B-tree). في هذا البحث تم دراسة وتحليل هذا الهيكل بأستخدام برامجيات فوكس برو –٨، وقد اثبتت الدراسة العملية اثبتت كفاءة هذا الهيكل في رص او ضغط المفاتيح. اجريت مقارنة بين هذا الهيكل وهيكل (B-tree) كنموذج لتبيان الفوائد الاكثر للهيكل (Bitmap).