Bitmap Indices: What are they and what do they achieve?

****

Abstract—**Bitmap indices are an integral part of several modern DBMSs, and are especially important in data warehousing applications. This paper reviews the current literature on bitmap indices, and presents an overview on what bitmap indices are, how they work, their benefits over other indexing methods, and evaluation of their use in various databases. We conclude with a brief outline of the limitations of current research on bitmap indices, and discussion of future issues surrounding them.**

I. Introduction

Bitmap indices are a type of database index that uses bitmaps as a data structure to improve the performance of queries on a database. Traditional indices associate with each index key value a list of row identifiers (RIDs for the set of rows containing that value. Bitmap indices, on the other hand, represent this set of rows by a bitmap (also known as a bit vector or bit array). Each bit in a bitmap is set to “1” if the row with the corresponding RID contains the index key value, otherwise it is set to “0”. These bitmaps are then used to answer queries by performing bitwise logical operations on them.

Traditional tree-based indices, such as the B-tree, are commonly used in most databases because they are very effective for On-Line Transaction Processing (OLTP). The operational complexity for searching and updating are nearly similar for tree-based indices, which is important for OLTP because search and updates are performed with similar frequency.

Data warehousing applications, however, require different indexing methods and designs from OLTP applications. Data warehouses are huge, special-purposes databases that contain integrated data from several different sources. They are typically used for analysis of large amounts of data via queries that aggregate, filter, and group the data. They form the foundation for Decision Support Systems that can be accessed via an On-Line Analytical Processing (OLAP) application.

Data warehouses are usually updated infrequently and in batch fashion, while search operations are performed very frequently and often involve complex, ad hoc queries that potentially access millions of records. Thus, query throughput and response times are more important than transaction throughput in data warehousing applications [1]. Additionally, search often involves multiple attributes in various combinations. Typical multi-dimensional indexing methods require a separate index for nearly every combination of attributes. Thus, the number of indices grows exponentially with the number of attributes in a data set – this is often referred to as the ‘curse of dimensionality’[2].

The data warehousing market has been growing substantially over the past few decades. Data warehousing applications are used in a variety of industries, including telecommunications (call analysis and fraud detection), manufacturing (order shipment and customer support), utilities (power usage analysis), retail (user profiling and inventory management), financial services (claims analysis, risk analysis, credit card analysis, and fraud detection), transportation (fleet management), and healthcare (outcomes analysis) [3]. This signifies a need for the research and development of indexing methods that favour OLAP applications, notably the bitmap index.

In this paper, we present a comprehensive review of the current literature surrounding these topics: what bitmap indices are and how they work; what they achieve over traditional database indices; and evaluation of their usage and performance in various databases. We then evaluate the current state of knowledge regarding these topics, and identify promising areas of future research.

II. Background

The following example briefly demonstrates how a basic bitmap index is constructed from a database table. Consider a table containing the customer data of a company (Table 1). Each record (row) pertains to a customer with two attributes (columns): region and gender.

TABLE I
Customer Data of a Company

Customer no.	Region	Gender
1	East	Male
2	North	Female
3	West	Female
4	West	Male
5	South	Female
6	North	Male

The basic bitmap index for the REGION attribute is shown in Table 2. It consists of four bitmaps, one for each region (East, West, North, and South). Each bit in the bitmap corresponds to one row of the CUSTOMER table. The value of each bit depends upon the values of the corresponding row in the table.

TABLE II
BITMAP REPRESENTATIONS OF CUSTOMER REGIONS

Customer No.	Region= 'East'	Region= 'North'	Region= 'West'	Region= 'South'
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	1	0
5	0	0	0	1
6	0	1	0	0

For instance, the bitmap REGION='east' in Table 2 contains "1" as its first bit, because REGION is 'east' in the first row of the CUSTOMER table. This bitmap has "0" for its other bits because none of the other rows of the CUSTOMER table contain 'east' as their value for REGION. On the other hand, the bitmap REGION='north' contains "1" as its 2nd and 6th bits, because the 2nd and 6th row of the CUSTOMER table contain 'North' as their value for the REGION column.

III. LITERATURE REVIEW

A. 1980s

Spiegler *et al.* (1985): Storage and retrieval considerations of binary data bases [4]

This paper demonstrates how binary representations of data based on their attribute values can be used to enable more efficient querying of data through an attribute. Although the term 'bitmap index' was not yet coined then, the method used in this paper is in actuality a basic bitmap index.

Spiegler *et al.* propose a novel approach to data storage and retrieval, in which the data appears in a binary form. They view the database as a 2-dimensional matrix that relates entities (rows) to their attribute values (columns). The entries of the matrix are then given '1' or '0' values, indicating that it has or lacks a given attribute value, respectively.

The authors report that the main advantage of this approach is to create an efficient method to access data via managerial transactions. Operational transactions usually involve updates or simple retrieval of data based on the primary key, while managerial transactions usually require a full vertical scan of the database to obtain all entities with a specific set of attribute values.

First, the feasibility of storing data in a binary form is investigated and justified. The authors provide an example in Fig. 1, where tables S and P are the relational form of data and tables S_b and P_b are their respective binary representations.

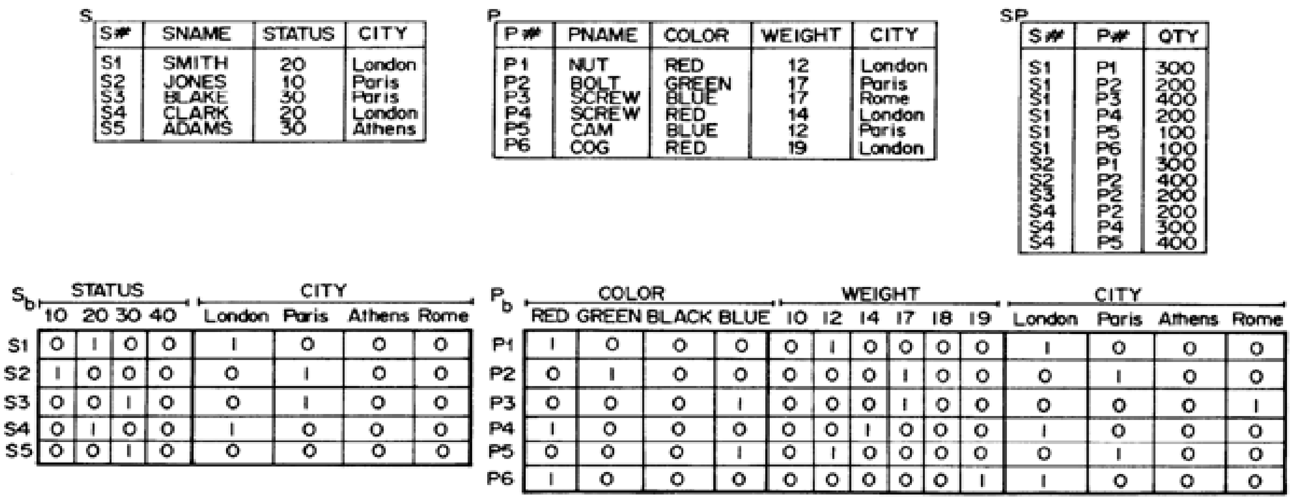
Operational transactions operate on rows based on the primary key; in the bitmap representation, each row is a vector of binary digits that represents a separate record. Unlike a standard query, in which the attributes are explicitly stated, the binary data base accesses rows, identifies the '1' values, and gets from a dictionary their respective meanings. Thus, the query "Get the data about P5" involves the operations:

1. Enter row 5 of P_b and locate the '1' entries
2. P_b(P5,4) = 'l', and (from the dictionary) color = blue
3. P_b(P5,6) = '1', and weight = 12
4. P_b(P5,12) = 'l', and city = Paris.

Updating a record via standard updating procedures entails replacing an old value with the new one of the same attribute. With bitmap representations, updating is performed by identifying the column of the old value, entering a '0' value in the record's row, and placing a '1' in the column that represents the new value. For example, in the database depicted in Fig. 1, the query "Change the city of supplier S3 from Paris to Rome" entails these operations:

1. Find, in S_b, the column "city = Paris" (column 6)
2. Find the column "city = Rome" (column 8)
3. S_b(S3,6) = '0'
4. S_b(S3,8) = '1'.

Managerial transactions, on the other hand, enter the database through its columns, usually accessing multiple rows at the same time. With standard representations, mapping techniques are usually required to perform a global query.



S

S#	SNAME	STATUS	CITY
S1	SMITH	20	London
S2	JONES	10	Paris
S3	BLAKE	30	Paris
S4	CLARK	20	London
S5	ADAMS	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	NUT	RED	12	London
P2	BOLT	GREEN	17	Paris
P3	SCREW	BLUE	17	Rome
P4	SCREW	RED	14	London
P5	CAM	BLUE	12	Paris
P6	COG	RED	19	London

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

S_b

	STATUS				CITY			
	10	20	30	40	London	Paris	Athens	Rome
S1	0	1	0	0	1	0	0	0
S2	1	0	0	0	0	1	0	0
S3	0	0	1	0	0	1	0	0
S4	0	1	0	0	1	0	0	0
S5	0	0	1	0	0	0	0	0

P_b

	COLOR				WEIGHT						CITY			
	RED	GREEN	BLACK	BLUE	10	12	14	17	18	19	London	Paris	Athens	Rome
P1	1	0	0	0	0	1	0	0	0	0	1	0	0	0
P2	0	1	0	0	0	0	0	1	0	0	0	1	0	0
P3	0	0	0	1	0	0	0	1	0	0	0	0	0	1
P4	1	0	0	0	0	0	1	0	0	0	1	0	0	0
P5	0	0	0	1	0	1	0	0	0	0	0	1	0	0
P6	1	0	0	0	0	0	0	0	0	1	1	0	0	0

Fig. 1: Binary representation of supplier-part database

However, the binary database does not require this, but rather executes the query by entering the columns and performing bitwise logical (AND or OR) operations on the “1” entries to obtain the rows that satisfy the query condition.

For example, in the query “Get the suppliers who live in Paris and their status is 10”, the search is performed by intersecting columns 1 and 6 of the S_b matrix via the AND operator as such:

$$\begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix} \cap \begin{bmatrix}0\\1\\1\\0\\0\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix}$$

Thus qualifying row 2 (*i.e.* supplier S2). A performance advantage is gained over standard methods because only the records that meet all conditions of the query are accessed.

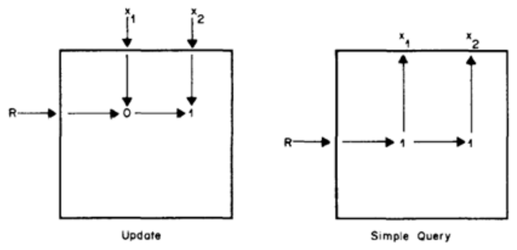


Fig. 2: Update and simple query operations of operational transactions on binary database

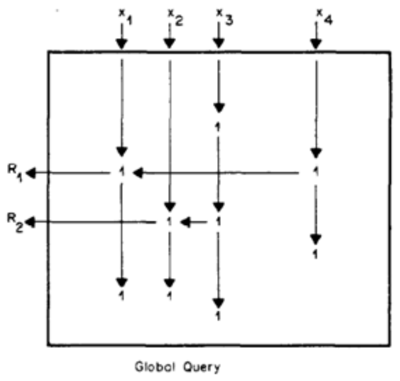


Fig. 3: Global query of managerial transactions on binary database

The authors then describe the benefits of using bitmap representations over the inverted list traditionally used for mappings. Given a file with *N* records, where *L* is the number of bits required to store any given value, and *k* is the number of distinct attribute values, the binary method requires *k*N* bits of storage, whereas the inverted list requires *N*L* bits. Given a typical length L of 32 bits, this indicates that binary storage is more efficient for $k < 32$.

The authors concluded that their proposed binary database is able to offer a unified solution for both operational and managerial transactions. The main advantage of the binary database is apparent in managerial transactions where complex queries can be processed in a simple and straightforward manner. On the other hand, there is not an evident advantage in the use of the binary database in handling basic operational transactions, though it is still feasible. Their major concern is related to the size of the binary matrix, which is likely to be quite large while containing only a small ratio of meaningful values (“1” entries), thus they propose the use of compression to handle this issue.

O’ Neil *et al.* (1989): Model 204 architecture and performance [5]

The first commercial implementation of the bitmap index, Model 204, is described in this paper. Model 204 is a DBMS that uses an indexing method that is a hybrid between the basic uncompressed bitmap index and a RID list. The index is organized as a B+tree, with each leaf node containing the RIDs. When the attribute cardinality is low, the RID lists are represented as bitmaps, with each bitmap representing one distinct value. When attribute cardinality increases, the bitmaps become sparser and require more disk storage space, so it automatically switches to use RID-lists instead.

The authors then measured Model 204’s performance against the performance of another DBMS, DB2. They found that Model 204 vastly outperformed DB2 in terms of CPU time, disk I/O and elapsed time for their trials with queries involving indexed access to data. However, they noted that DB2 performed better than Model 204 in retrieving records based on non-indexed field values.

Thus, they concluded that Model 204 is designed to work best with larger amounts of data, where the increased cost of access by non-indexed fields disappears in the face of the vastly increased performance in access of indexed fields. In small files of <10,000 records, Model 204’s performance advantage disappears.

It is also noted that there has been no mention of Model 204’s performance in terms of updating records.

B. 1990s

O’ Neil *et al.* (1995): Multi-table joins through bitmapped join indices [6]

This paper proposes the use of bitmap indices to create a query execution method that efficiently executes multi-table joins. The authors have focused on the star-join, a type of join that involves joining a central table with large cardinality to a descriptive smaller table, but claim that their technique is applicable to other types of multi-table joins as well.

The method proposed by the authors uses join indices with compressed bitmap representations. This allows predicates restricting columns of the descriptive (smaller) tables to determine an answer set in the central (high cardinality) detail table, and easily finishes up the join of the rows of the restricted central table back to the descriptive tables. The authors explain how these techniques would outperform alternative star-join methods that do not use bitmap representations. However, performance results of their method obtained via empirical analysis are not shown.

Chan *et al.* (1998): Bitmap index design and evaluation [7]

This paper contains the first set of guidelines for database design using bitmap indices. The authors present a general framework to study the design of bitmap indices for selection queries, and examine the space-time tradeoff of various types of bitmap indices.

They conclude that range-encoded bitmap indices generally offer a better space-time tradeoff than equality-encoded bitmap indices, and propose an optimal bitmap buffering policy.

Wu *et al.* (1998): Encoded bitmap indexing for data warehouses [8]

The authors propose and evaluate encoded bitmap indexing for data warehousing applications. Encoded bitmap indexing inherits the beneficial properties of simple bitmap indexing, including cooperativity of different bitmap vectors, low cost of construction and maintenance, and low processing cost. It also solves the problems inherent in simple bitmap indexing, namely sparsity and the dramatic increase in space required to index high cardinality attributes. They conclude that encoded bitmap indices are suitable for indexing OLAP data.

C. 2000s

Jürgens *et al.* (2001): Tree based indexes versus bitmap indexes: A performance study [9]

In this paper, the authors compare the performance of bitmap indices against tree-based indices, which are the basis of most traditional indexing methods. They use a set of nine parameters (configurations) to compare two tree-based index structures with two bitmap structures, to evaluate which index structure works the best for the given range queries.

Classification trees are first applied to analyze which parameters influence the performance of the index structure, and to determine rules to select the most efficient index structure. Then an aggregation technique is applied to investigate which index structure is most efficient for various types of configurations.

From the use of their aggregation technique, the authors found that for very large query boxes, the range encoded bitmap index performed best. They then extrapolate their findings to future disk technology, based on the parameters for disks expected to be available in 2005. They hypothesize that in the future, as disk latency and bandwidth increases, bitmaps are expected to gain performance advantages over tree-based indexing methods, especially if the number of dimensions of the query is increased. Bitmaps are proposed to perform better than trees with future disk technology for queries with at least three dimensions.

Wu *et al.* (2003): Using bitmap index for interactive exploration of large datasets. [10]

The authors seek to demonstrate how bitmap indices work in solving a common issue faced in exploration of large spatio-temporal datasets. A common method of exploring datasets is to identify and track regions of interest, which is usually sped up by using a tree-based indexing scheme. However, the tree-based indices are usually efficient only if search is limited to a small number of select attributes, whereas scientific datasets usually contain hundreds of attributes and scientists usually need to study them in complex combinations. The authors propose that bitmap indices are an efficient method for simultaneous search on multiple criteria in region growing and region tracking

The thresholded region algorithm and isocontouring algorithm are algorithms that have been previously used to identify and track regions of interest. However, they suffer from the limitation of only being designed for regions of interest defined on one attribute. Several multidimensional indexing schemes can be applied to remove this limitation, but most of them either suffer from the 'curse of dimensionality' or are only efficient if the queries involve all attributes. The 'curse of dimensionality' means that their performance decreases rapidly as the number of attributes increases. Thus most existing index schemes are only useful when the number of attributes is small, *e.g.* 10 or less.

Also, most of the multidimensional indexing schemes reorder the records according to attribute values, which is crucial for them to conduct efficient search operations, but unfortunately also causes the region growing procedures to be very slow. This is because the reordering of records by value destroys spatial relation among neighbours, which needs to be recovered during region growing.

The authors hypothesize that region tracking problems are similar to data warehouse applications, in that the data is read-only and the queries are complex and large. As bitmap indices are widely used in data warehouse applications, they propose that its use can be extended to region tracking problems. Bitmap indices do not suffer from the 'curse of dimensionality', and are especially efficient when queries do not involve all of the attributes. Also, they do not require reordering of records, thus the records can be kept in original order to maintain neighbourhood relation for the region growing step.

The authors formulated their region-tracking program using a bitmap index with WAH compression and binning, and tested its performance on analyzing the datasets produced from a simulation of the hydrogen-oxygen autoignition process. They found that these tests confirmed the efficiency of their program. On a 33GB dataset, it took <1 minute in worst-case to process complex conditions involving 4 different attributes, taking about 10 seconds on average.

However, the authors have not compared their program against the optimal isocontouring algorithm for region tracking, as the isocontouring algorithm cannot be directly used in region tracking. They have shown that their algorithm has the same theoretical complexity as the optimal isocontouring algorithm; however, they propose the future work of actually implementing an isocontouring algorithm to compare performance.

Wu *et al.* (2004): On the performance of bitmap indices for high cardinality attributes [11]

The authors note that while the efficiency of bitmap indices for read-only attributes with low cardinality has been well-established, the size of the bitmap index for attributes with high cardinality can be prohibitively large, unless specialized compression schemes are used. They also noted that while empirical evidence exists that some of these schemes work well, there has not been any systematic analysis of the effectiveness of bitmap indices employing these schemes with high cardinality attributes.

Thus, in their work, they present a systematic analysis on the performance of compressed bitmap indices on answering range queries. They first analyze the space and time complexities of bitmap indices using WAH and BBC compression for answering one-dimensional range queries, and show that the total sizes of the compressed bitmap indices are relatively small even for high cardinality attributes. For most high cardinality attributes, they find that the WAH compressed indices use about 2N words, half the size of a traditional B-tree index, while BBC compressed indices are even smaller.

They also found that that the time and space required to perform a bitwise logical operation on two compressed bitmaps is in worst-case proportional to the total size of the two. Also, bitwise OR operations on a large number of bitmaps can be performed in linear time to the total size, by using an in-place algorithm. For a search algorithm, an important definition of optimality is that its time complexity is linear to the number of hits, thus the compressed bitmap index can be considered optimal for high cardinality attributes because the total size of the bitmaps involved in the query is proportional to the number of hits.

The authors measured and compared the performance of their compressed bitmap indices to the projection index, which is often considered the best indexing method for searching high dimensional datasets. They noted that on average, the compressed bitmap indices performed significantly better than the projection index. In worst case, the WAH bitmap index takes no more time than the projection index, but the BBC bitmap index takes longer because more CPU time is required.

It is concluded that with the use of appropriate compression, the bitmap index can perform efficiently on high cardinality attributes as well as low cardinality attributes.

Wu *et al.* (2006): Optimizing bitmap indices with efficient compression [12]

This article reinforces previous observations regarding the feasibility of WAH compressed bitmap indices on both high and low cardinality attributes, based on comparison with other popular indexing methods such as projection indices and B-tree indices. The authors conclude that the WAH-compressed bitmap index is optimal, and it has an advantage over other similarly optimal indexing schemes such as B+ and B* tree indices, in that the results of one-dimensional queries can be efficiently combined to answer multidimensional queries. This makes WAH compressed bitmap indices very suitable for ad hoc analyses of high-dimensional and large datasets.

Stockinger *et al.* (2007): Bitmap indices for data warehouses [2]

In this article, the authors reviewed the recent developments in bitmap indices, under the categories of encoding, compression, and binning. They also presented an overview of commercial bitmap index implementations in DBMS systems. They conclude that although most of the bitmap indexing methods were designed to answer multi-dimensional range queries efficiently, they are also efficient for other types of queries, *e.g.* joins on foreign keys and computation of aggregates.

IV. Conclusion

Bitmap indices are a promising solution to many of the issues faced by data warehouses and OLAP applications. The main benefit of using bitmap indices is the significant space and performance advantage gained in performing queries (eg. range, aggregation, and join queries) on large amounts of data, especially data with low attribute cardinality. Recent improvements on the bitmap index, such as WAH compression and range encoding, have even made the bitmap index a viable method for indexing of data with high attribute cardinality [8],[11].

Representing a large table using a bitmap instead of a RID list significantly reduces I/O. Also, as storage of bitmaps is especially efficient (*e.g.* a bitmap with 10 million rows only requires a maximum of 1.25 MB of storage), bitmaps can be pipelined or cached in memory to facilitate complex queries [4]. The original ordering of rows can be maintained with bitmap indices, which is useful in region tracking applications [10], and when retrieving rows from disk and combining predicates. Also, the most common operations used in combining predicates, AND and OR, can be parallelized very efficiently by executing 32 or 64 bits in parallel [4].

The bitmap index is capable of breaking the 'curse of dimensionality' which afflicts other efficient indices such as B+ and B* tree indices [2], as it allows one-dimensional queries to be efficiently combined to answer multi-dimensional queries. This makes bitmap indices, especially with the use of WAH compression, very suitable for ad hoc analyses of high-dimensional and large datasets [12].

The bitmap index, due to its proven efficiency, has been implemented in several commercial DBMS. The first commercial product to implement it was Model 204, which used the basic bitmap index with no improvements. ORACLE implements a proprietary version of compressed bitmap indices in its flagship product since version 7.3, while Sybase IQ implements unbinned, binary encoded, and uncompressed bitmap indices. IBM Informix contains bitmap indices for queries involving multiple tables, designed to speed up join-operations. InterSystem Corp's Cache has had bitmap index support since version 5.0 [2].

V. Future Issues

Most of the studies have concentrated on the efficiency of bitmap indices with various improvements (*e.g.* compression, encoding) in answering multi-dimensional range queries on read-only attributes. Thus far, the issue of updating the indices has been neglected. Although this issue is of lesser importance in data warehousing than efficient queries, there is still a need to update indices when new records are added. Efficient solutions to this issue could be instrumental in allowing the bitmap index to gain more widespread use in commercial DBMS, and to extend its use to other applications.

A suggestion was made to mark the bits of the modified records in the indices as invalid, store the updated records separately, and update indices when the system is idle; however, this has not been followed up yet [12]. We propose that the design and evaluation of various methods of updating bitmap indices could be a promising area of future research.

Studies have shown that WAH compression and range encoding have allowed the bitmap index to be used efficiently with high-cardinality data as well as low-cardinality data [8],[11], but little research has been done on extreme cases of high-cardinality, *i.e.* when every value is distinct. In this scenario, even a compressed bitmap index may be three times as large as other indices [12]. Model 204 alleviates this issue by switching from the bitmap index to a compact RID list when attribute cardinality is high [5]. However, it would be interesting to see other approaches to this problem that do not involve switching to another type of index.

The bitmap indexing method is considered to be suitable and efficient for ad hoc range queries on multidimensional data, as it generates one bitmap index for every attribute. However, if some attributes are frequently used together, a composite index might be a more efficient approach. There might be room for future work involving the evaluation of the optimality of composite bitmap indices [12].

References

[1] O'Neil, P., & Quass, D. (1997, June). Improved query performance with variant indexes. *ACM SIGMOD Record*, *26*(2), 38-49. http://dx.doi.org/10.1145/253262.253268

[2] Stockinger, K., & Wu, K. (2007). Bitmap indices for data warehouses. *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, 5, 157-178. http://dx.doi.org/10.4018/987-1-59904-364-7.ch007

[3] Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and OLAP technology. *ACM Sigmod record*, *26*(1), 65-74. http://dx.doi.org/10.1145/248603.248616

[4] Spiegler, I., & Maayan, R. (1985). Storage and retrieval considerations of binary data bases. *Information processing & management*, *21*(3), 233-254. http://dx.doi.org/10.1016/0306-4573(85)90108-6

[5] O'Neil, P. (1989). Model 204 architecture and performance. In *High Performance Transaction Systems* (pp. 39-59). Springer Berlin Heidelberg. http://dx.doi.org/10.1007/3-540-51085-0_42

[6] O'Neil, P., & Graefe, G. (1995). Multi-table joins through bitmapped join indices. *ACM SIGMOD Record*, *24*(3), 8-11. http://dx.doi.org/10.1145/211990.212001

[7] Chan, C. Y., & Ioannidis, Y. E. (1998). Bitmap index design and evaluation. *ACM SIGMOD Record*, *27*(2), 355-366. http://dx.doi.org/10.1145/276305.276336

[8] Wu, M. C., & Buchmann, A. P. (1998, February). Encoded bitmap indexing for data warehouses. In *Data Engineering, 1998. Proceedings., 14th International Conference on* (pp. 220-230). IEEE. http://doi.ieeecomputersociety.org/10.1109/ICDE.1998.655780

[9] Jürgerns, M., & Lenz, H. J. (2001). Tree based indexes versus bitmap indexes: A performance study. *International Journal of Cooperative Information Systems*, *10*(03), 355-376. http://dx.doi.org/10.1142/S0218843001000382

[10] Wu, K., Koegler, W., Chen, J., & Shoshani, A. (2003, July). Using bitmap index for interactive exploration of large datasets. In *Scientific and Statistical Database Management, 2003. 15th International Conference on* (pp. 65-74). IEEE. http://escholarship.org/uc/item/5zn6n438

[11] Wu, K., Otoo, E., & Shoshani, A. (2004, August). On the performance of bitmap indices for high cardinality attributes. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30* (pp. 24-35). VLDB Endowment. http://dx.doi.org/10.1016/B978-012088469-8.50006-1

[12] Wu, K., Otoo, E. J., & Shoshani, A. (2006). Optimizing bitmap indices with efficient compression. *ACM Transactions on Database Systems (TODS)*, *31*(1), 1-38. http://dx.doi.org/10.1145/1132863.1132864