Query Acceleration in Multimedia Database Systems

Ramzi A. Haraty and Rawa Karaki
Department of Computer Science and Mathematics
Lebanese American University
Beirut, Lebanon
e-mail: rharaty@lau.edu.lb

***Abstract*--With the increasing popularity of the World Wide Web comes the enormous increase in stored digital contents, which could challenge users to search and use the multimedia data efficiently. This work focuses on hastening techniques for efficient retrieval of multimedia data. In this paper, we exploit the use of bit-vectors to accelerate queries in multimedia databases. We use a compressed bit-vector to minimize the amount of data cashed on disk; thus, reducing the amount of memory and time needed to execute queries. We also compare our scheme with other related strategies.**

***Keywords-multimedia; bit vectors; query accelerations*.**

I. INTRODUCTION

Multimedia databases have become one of the puffs in computer science technology. It is a recent evolution of the Internet and data warehousing. Many authors wrote about the evolution of multimedia databases and ways to implement it [1][2]. Multimedia is a mix of multiple mediums - images, sounds, music, audios and videos etc. As long as the development of the Internet and computer technology continues, multimedia files will appear more and more in many applications. For that reason, it is important and significant that the data files of multimedia objects be arranged, ordered and categorized so we can simply access them at any time. Therefore, multimedia databases are the necessary tool to handle and support these enormous multi-media object files.

A multimedia database is a type of database that is similar to all other database types except that it contains multimedia files in its collection. To organize and manage multimedia data files, a multimedia database management system is needed. It is a program that runs and directs the collection of media files and allows entry for end users to retrieve multimedia files or objects. In general, multimedia databases hold images, audio, video, animations and many other file forms. All files or data are saved as binary forms in the multimedia database.

Multimedia database implementation differs from regular database implementation in the design of the media objects and files where the files are kept and stored. Different characteristics of multimedia data represent the diversity of the data since they are complex--composed of audio-visual data. Research shows that objects in multimedia data are complex and involve a chained structure that can hold a connection between them [3][4]. Static media, such as text, graphics, and images, are time-independent like. For instance, image files do not have time-related action because there is no connected time factor. Video files, on the other hand, are dynamic, and have both time and dimensional dependency. This is due to the fact that a video is composed of multiple ordered image frames which associate to form the video file.

In this paper, we use a compressed bit vector for multimedia data retrieval to select files from a database more efficiently. The method facilitates rapid searching of multimedia data objects in a multimedia database. A single bit vector is used to determine matches for the main query, returning a reduced set of multimedia objects instead of the entire multimedia data object; thereby greatly reducing the query search time, increasing the efficiency of the process by allowing the bit-level operations and minimizing the cost and amount of data transferred. The execution time is exactly proportional to the size of input. The algorithm complexity is of order O(n).

The rest of this paper is organized as follows: Section 2 provides a brief explanation of multimedia database management systems. Section 3 presents related work. Section 4 presents the compressed bit vector algorithm and its execution results. Section 5 gives the conclusion and future work.

II. MULTIMEDIA DATABASE MANAGEMENT SYSTEMS

With the evolution of Internet and computer users, multimedia data text, graphics, and images a greater effect on our daily lives. That is why finding a new technique to easily retrieve enormous multimedia information and a file, at any point of time, is in high demand. Any multimedia object can be generally described as a group of extended, shapeless

Copyright (c) IARIA, 2015. ISBN: 978-1-61208-415-2

series of bytes. These objects are called BLOBs (Binary Large Objects). BLOB files are usually very large in size; for this reason, database management systems provide particular maintenance to insert, delete, modify or retrieve BLOB objects from database.

Modern databases are frequently capable of storing BLOBs and CLOBs (Character Large Objects), as columns in their tables. Data stored in a BLOB column can be accessed using connectors and manipulated using client-side code. Reading a BLOB from the database is a slow task considering the size of a multimedia object. A BLOB can contain as much as four gigabytes of data for each field. Multimedia database systems are thus required to provide an efficient cache of the BLOB files, but this is not sufficient for multimedia implementation maintenance. Therefore, a query of a prolonged continual series of bytes is restricted to a matching pattern and reorganization of a BLOB multimedia object may return zero results due to missing constructional information. Even if it can be realized, to draw out information of the object in realistic time, for example working with pattern identification techniques, would be unrealistic. Thus, a multimedia database system should keep an analytical structure of the BLOB files. Multimedia objects can be saved in smaller parts to allow easier retrieval of BLOB objects based on content. Multimedia data is sizeable and have an impact on the retrieval, insertion and manipulation of multimedia data files. The large amounts of data to be processed can be checked against those that need to be processed. Table I illustrates the enormous sizes of data for media files of different types.

III. RELATED WORK

Querying and retrieving information in multimedia databases differs from traditional databases [5][6]. A fairly straightforward search can be done in alphanumeric databases. Multimedia databases contain pictures and different complex multimedia data objects; thus, the database is not easily indexed, classified and retrieved [7]. How is it possible to retrieve a picture with a cup of water or a horoscope sign? Those shapes are difficult to recognize. Some retrieval classes for multimedia databases include:

- Retrieval by Browsing (RBR): Browsing multimedia objects to retrieve the best matching file. For example, using a simple interface to let users browse small images known as “thumbnails” to pick the image that matches the query.
- Retrieval by Metadata Attributes (RMA): Designing a query that addresses the Meta and logical characteristics. For this purpose, any media file is stored with information describing the file. For example, we will not query an image with a bird but we will address our search to find which media handles the keyword ‘bird’ as its meta information.
- Retrieval by Shape Similarity (RSS): It is a type of retrieval based on media content. Searching in a multimedia database based on shape similarity of the file. For example, retrieve all the images that contain a circle.
- Retrieval by Content Attributes (RCA): Query is sent with enough detail describing the file to be retrieved. For example, retrieval of all images that contain a specific celebrity.

In this paper, we focus on the RBR and RMA since they are the most widely used retrieval classes in multimedia databases.

A. The Retrieval by Browsing

A user who requests the search for a specific file uses terms and details to illustrate the retrieval system. Then, the software matches the query with existing matching objects and returns a list of files to the end user for examination. The end user then considers the retrieved files and picks items that exactly match his needs. This type of retrieval works best in finding the exact requested file, but multiple problems appear with its implementation:

1. End users find it hard to formulate queries.
2. Queries may return only unwanted files and result in too many suggested unwanted matches.
3. Query terms are not properly valued.
4. Multiple forms of image and audio files that need conversion.

TABLE I. SAMPLE MEDIA TYPES, FORMATS, AND RELATED DATA VOLUMES AND TRASFER RATES [5].

Media Type	**Sample Format**	**Data Volume**	**Transfer Rate**
Text	ASCII	1MB/ 500 pages	2KB/page
B/W Image	G3/4-Fax	32MB/500 images	64KB/page
Color Image	GIF, TIFF, JPEG	1.6GB/500 images 0.2GB/500 images	3.2MB/image 0.4MB/image
CD-music	CD-DA	52.8MB/5 minutes	176KB/sec.
Consumer Video	PAL	6.6GB/5 minutes	22MB/sec.
High quality video	HDTV	33GB/5 minutes	110MB/sec.
Speech	m-law, linear; ADPCM, PEG audio	2.4 MB/5 minutes 0.6MB, 0.2MB/5 min.	8KB/sec.

Copyright (c) IARIA, 2015. ISBN: 978-1-61208-415-2

Different authors have proposed that browsing, which uses the human recognition capabilities, can control and solve the above difficulties [8][9]. Though, the retrieval by browsing is suggested to be a direction solving many problems in multimedia retrieval and handling multimedia systems, but it is logically seen as a difficult and time inefficient task for humans to solve [10].

B. The Retrieval by Metadata Attributes

Generally, human beings have the power to retrieve and correlate information efficiently. It is unfeasible to search millions of data by simply "staring" in order to assemble diverse documents, which may involve texts, videos, audio and images files, either alone or as multimedia items. Thus, we seek a simple technological multimedia search based on known information of the file.

Metadata are data about data. Metadata can describe any data using different categories: quantity, quality, materials, shape and different properties of the data as tools to find, understand and access the data files. Metadata details can aid users to have an explanation about the data being searched in multimedia databases. The picture itself describes nothing more than an ordinary image with colours. Without having the metadata description associated with the picture, it will be out of question for machines to know the properties of this picture. For example, if we would like to know when and where this picture was taken, or its resolution etc., we turn to Metadata. All this information does, is provide a key that aids in specifying the properties of the image to be used in many applications [11].

The Metadata model requires descriptive information of the content, combined with contextual information, saved in the multimedia database in reference to the multimedia object, and used as an information tool for browsing with a point of association of a specific media. Descriptive information is valuable for searching a multimedia object, and is of major importance when contacting explored results where the attribute, such as a photographer name, a singer name or date, are applied to choose and retrieve the file. The metadata representation of the file is flexible and adopts a multilevel approach for describing the file to permit multiple particles to describe the facts and figures of the file. The metadata model may be unusable to work on a single level in describing a media file with multiple classes of representation [12]. For an image, multiple descriptive data are associated with saved image snaps that can provide accommodation in the model. For a video file in a broadcasting station, information could be automatically produced for each shot or segment that describes the scene.

Lord and Pratt reported a technique of retrieving data from a BLOB data warehouse using SAS as the data analysis tool [13]. The data warehouse architecture requires storing summary data in traditional database relational databases and storing raw chip data in a multimedia database BLOB data type. With this BLOB data type, many opportunities have opened up for experimenting with various methods of retrieving data. Since the databases are fragmented among multiple machines (due to the large data volumes), and to make it easy to register a structure that is required to access the inner parts of the BLOBs, a machine is set aside specifically to direct the client applications and SQL users to the machine where the required data resides. This machine also provides the information necessary to extract parts of the BLOBs. We refer to this machine as the application director. At the database end, the objects would be too large to be practical. With data volumes in the hundreds of gigabytes, adding descriptive information into the records would explode the data storage requirements beyond reasonable limits. Objects also allow us to store large numbers of data values.

After the storing of the object, we have to specify how to access this object. This is where the registry comes in. The registry is a set of tables that define the type of object; in this case the type is defined by the application, (not necessarily a database data type) and the contents of the object. Each object is comprised of elements that have a name, type, and length. All of this information is stored in the registry. The query looks into the objects and extracts that element, returning it as a column in the user view. An example of a query is as follows:

```
SELECT LOT, WAFER, CHIP, SETELEMENT
    (OBJECT1, D_VAL1)
FROM DB.TABLE1
WHERE LOT = '123456789' AND WAFER = 'ABCDEF'
```

This query gets the BLOB object1 in the database from the TABLE1 table and finds the D_VAL1 element in each object, returning it as a column in the table.

Srivastava and Velegrakis [14] described that several metadata management tools consider the metadata as an integral part of the data, which means that metadata cannot be retrieved without also retrieving the data with which it is associated. The authors showed that storing the metadata in independent tables, associated to the data through the q-values, allows them to be queried and retrieved independently. For instance, if a user would like to know the sources that have been used to collect info of a file, he can simply query the metadata table alone.

IV. THE COMPRESSED BIT-VECTOR FOR MULTMEDIA DATA RETRIEVAL

The existence of an enormous volume of media data files questions the aspects of the management of multimedia objects and the problem of implementation. Typically, queries in multimedia database are multidimensional and have complex selections. Users that request specific queries in multimedia databases usually find it hard to find answers to all requirements. Due to these characteristics, bit-vector indexing techniques have shown promising results for processing multimedia databases [15]. A significant advantage of the bit-vector technique is that complex logical selection can be performed very quickly via bit-wise AND, OR and NOT operators. In this paper, we further explore the issues of query acceleration using bit-vectors, and we concentrate on optimizing one of the query operations "Selection,"

Copyright (c) IARIA, 2015. ISBN: 978-1-61208-415-2

which is further discussed with simple queries, and later with more complex queries using the four different types of joins: hash join, inner join, merge join and nested loop join. The space for the compressed bit-vectors works best compared to other techniques.

A bit-vector is a vector or array of data that stocks bits briefly. A bit vector is time composed from the bit values of the collection {0, 1}. Bit-vector is a term applied here to denote a large classification and indexing plan that stocks index as bit sequence. A bit-vector is a bit string in which each bit is mapped to a record ID. A bit in a bit-vector is set to 1 if the corresponding ID has a property "P" and is reset to 0, otherwise. The property "P" is true for a record if it has the value "x" as attribute "X." The query selection can also involve many attributes. Bit-vectors permit vectors of bits to be stocked and handled in the memory set for extended time phases. Bit-vectors can potentially explore bit-level similarity, utilize the data cache to the max, and minimize access to memory. Bit-vectors usually work best in different data forms on reasonable data sets, and on those that are efficient asymptotically [16]. To further improve their effectiveness, we study their compression scheme, which will potentially minimize the area used without expanding the managing time of the query.

Generally, a bit-vector is stocked as a group of bits and the majority of operations on regular bit-vectors are logical bitwise operations. Considering our concerns in using the bit index on huge databases, the main aid is to reduce the sizes of the index. In addition, we aim to efficiently execute logical operations on the compressed bit-vectors. A problem with using uncompressed bit-vectors is their large size and possibly of high expression assessment costs when the indexed attribute has a high cardinality [17]. A single technique to deal with using bit-vectors on high-cardinality attributes problem is to store them in a compressed bit-vector form. Using compressed bit-vectors has multiple advantages that potentially adjust performance: minimized disk space needed to stock the indices, faster reading of the indices from the disk into the memory, and more cached indices in the memory with this compressed form. Several Boolean operation evaluation algorithms, which operate on compressed bitmaps without having to decompress them, might be faster than same operations on the regular bit-vectors. The scheme for compressing data, in addition to transforming data, guides the reducing of enormous volume required. The technique here is to alter the issued multimedia data bit-vector to another modified area to eliminate the redundancies in the real data.

A bit vector "B" of "u" bits can be represented as B[0::u). It can be stored in uH1(B) bits so that the operations can be answered in constant time. We will only save the 1-bits in if the response to the query is true. With this representation of "B," we can access any block of size "b" in constant time, which is sufficient for implementing rank and selecting. In addition, access queries can be answered in constant time, as well.

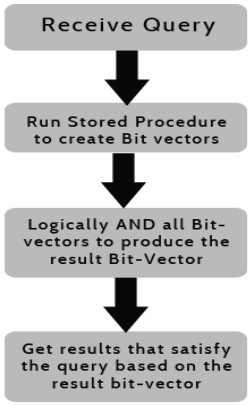

Figure 1. Algorithm workflow.

Decompression is made from the backwards process to retransform and decode the data to its native origin form. This operation generally encounters some data loss, which is a major problem of multimedia applications. Our algorithm ensures negligible loss of data when retrieving information.

Our algorithm compresses bit-vector for multimedia data retrieval and uses these bit vectors to return exact answers to any query in multimedia databases, with any retrieval process used. For example, a specific shape may be compared to a number of pictures in a multimedia database to find a picture or many pictures with the same characteristics. The search may result in either one or more matches found, or no matches at all in a set of objects in the multimedia database.

Figure 1. is an example operation on how a query can be handled in searching for a specific attribute in a multimedia database. First, a receive query operation receives a query item. When a user requests a query in multimedia database with some attribute, a bit vector index is created for each attribute. Each bit vector index indicates whether each of the attributes in the selected database does or does not exist in any of the retrieval strategies used. When a query is received, the bit vector indices associated with each of the selected attribute values are then logically ANDed together to form a single result bit vector index. The result bit vector index identifies a reduced set of accepted IDs of the data table containing the multimedia objects. This reduced set of IDs in the multimedia data objects returned by the bit operations may then be quickly searched using a linear scan to determine a match or matches for the query point. To retrieve resulting matches, we simply select the IDs of the query table that contain a "1" bit in the bit-vector. The stored procedure used in building the bit vector of the specified attributes for any query in multimedia database is depicted in figure 2. For simplicity and straightforwardness, we used the "retrieval by meta and logical attributes" strategy in a real university database.

Copyright (c) IARIA, 2015. ISBN: 978-1-61208-415-2

```
DROP PROCEDURE IF EXISTS  mysql_BitVectorTable //
CREATE PROCEDURE mysql_BitVectorTable ( IN attributeValue
VARCHAR(255))
BEGIN
        DECLARE idSelected    VARCHAR(255);
        DECLARE exit_loop BOOLEAN;
        -- Cursor for select statement
        DECLARE query_cursor CURSOR FOR SELECT id FROM
students where city = attributeValue;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET
exit_loop = TRUE;
        DROP TABLE IF EXISTS bitvector;
        -- create a new table in database with id and Boolean
   CREATE TABLE bitvector (id VARCHAR(7),bitValue BOOLEAN);
        OPEN query_cursor;
        query_loop: LOOP
                FETCH  query_cursor INTO   idSelected;
                -- save in bitvector
                INSERT INTO bitvector (id,bitValue) VALUES
(idSelected,1);
                IF exit_loop THEN
                        CLOSE query_cursor;
                        LEAVE query_loop;
                END IF;
        END LOOP query_loop;
END //
```

Figure 2. The Create Bit Vector stored procedure.

After the construction of the bit vector, it will be stored in the database as a regular table. Each bit vector contains two fields: The first corresponds to the original table index, and the second contains the bit 0 or 1 referring to the absence or presence of the main query attribute. Each bit vector should contain the same number of indexes as the original table. But to compress our bit vector, we will only save the 1 bits associated with the presence of the query attribute and remove the 0 bits from the bit vector. Thus, the bit vector will contain a smaller number of bits and minimize the response time of the process.

The first experimental query is to select all information and profile picture of students that belong to a specific campus in a specific major. We ran our algorithm on a database table containing multimedia files. We used a traditional database application that uses fixed sized data, but the multimedia size of data can vary dynamically. All unformatted data (mainly text and images) has been handled in this database system through BLOBs. They usually support only a few generic operations, such as reading or writing parts of BLOB. The first table used is the student application table with student images in each record. The table includes more than 510,000 records of student information. The tested query involves retrieving the student images that match certain required parameters. The outcome result will determine the time it took to handle this simple query.

In this simple query, the program indicates that it requires an execution time of 107.334 seconds. This means that there is a need for a method to run queries and return results in a more efficient time. The stored procedure, described above, is used to build the bit vector for the same simple query. A stored procedure is built for every attribute value in the query. After selecting the first attribute, a bit vector table is created and saved in the database. A second bit vector is created for the second attribute. Creating both bit vector took:

$$0.799+1.446 = 2.245 \text{ seconds}$$

Next, we will "AND" all bit vectors created to maintain the final bit vector. Using a time calculator, the retrieval of student images took 3.84 seconds to display on the website. We have also tested our algorithm on different kinds of queries. Other than the simple query noted above, we used two attributes for tables with an index.

We ran our algorithm on simple queries using two attributes for tables without index, then for complex query using hash join, inner join, and nested loop join. To test our algorithm on another more complex query, we will use the "inner join" type. For example, we ran our algorithm with the following query:

```
SELECT id FROM applications
INNER JOIN majors
        ON applications.mjrid = majors.mjrid
WHERE attribute1 = 'a' and attribute2 = 'b'
```

The time it took to build the results of this query in the regular case is: 112.182 seconds. Furthermore, the processing time to display the result is: 3.6691 seconds. The required total time for our algorithm is: 10.73 seconds. The previous results show the efficiency and rapidity of searching of multimedia data using the bit vector algorithm with the metadata retrieval system. Table II shows the time of different kinds of queries with and without applying our algorithm.

To further enhance our algorithm, we wrote it without a stored procedure function. Code that generates the bit vectors stored on the web server functioned as the bit vector. The query selected each attribute alone to retrieve the IDs that match the query results. Then the bit vector was saved in the memory using a key and a value. The key corresponds to the media file ID in the database, and the value corresponds to {0, 1} of the bit vector. To compress our bit vector, we only saved the 1 bits in memory.

TABLE II. EXECUTION TIME OF VARIOUS QUERY STRATEGIES.

Query Type	Running Time Without Bit-Vector Algorithm	Running time With Bit-Vector Algorithm Using Stored Procedure
Query with Attributes For Table With Index	107.33 seconds	6.87 seconds
Query with Attributes For Table Without Index	121.54 seconds	11.28 seconds
Query with Inner Join	112.18 seconds	10.73 seconds
Query with Hash Join	106.53 seconds	5.53 seconds
Query with Nested Loop Join	107.87 seconds	6.71 seconds
Query with Merge Join	107.12 seconds	6.54 seconds

Copyright (c) IARIA, 2015. ISBN: 978-1-61208-415-2

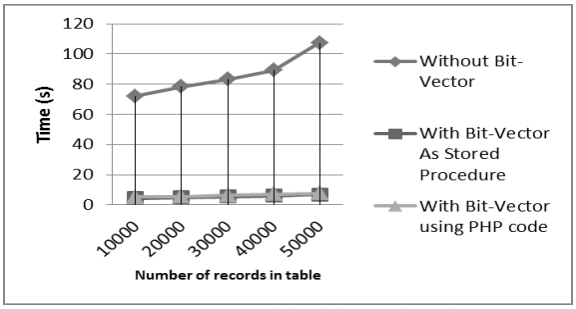

Figure 3. Comparison of the different algorithm.

After saving the bit vectors for each attribute, we added the "AND" or "OR" in the bit vectors according to the query requirements to get the final IDs that respond to the query result. The results are depicted in figure 3.

To calculate the complexity of our algorithm, we defined time taken by the algorithm without depending on the implementation details as our algorithm runs in linear time. The execution time is exactly proportional to the size of input. The algorithm complexity is of order $O(n)$.

V. CONCLUSION

A new strategy is proposed for retrieving multimedia data objects stored in a database. We searched for specific queries selecting objects from a multimedia database such as searching for particular images stored in the database. As a result of the search, either one or more true results are found, or no result exists in the set of objects in the database. Our bit vector for retrieving media files algorithm was proposed and tested on real data. In fact, bit vector indexing techniques have shown promising results for processing multimedia databases. We have explored the issues of query acceleration using bit vectors, and we have concentrated on optimizing "Selection" using the four different types of joins: hash join, inner join, merge join and nested loop join. To optimize the results returned, our method uses a compressed bit vector to save the accepted rows of information. This method guarantees fast and efficient query results. This technique also minimizes the cost and amount of data transferred. Our test results show that the simplest approach towards solving queries in multimedia database is the linear scan. This approach outperformed more complicated approaches.

As for future work, we are currently working on using this compressed bit vector to construct abstractions to be used for more powerful concurrent query analyses in multimedia databases, such as saving repeated queries in existing libraries. This may lead to more efficient and faster query response time.

ACKNOWLEDGMENT

This work was sponsored by the Lebanese American University - Beirut, Lebanon.

REFERENCES

[1] G. Chechik, . Le, M. Rehn, S. Bengio, and D. Lyon, "Large-scale content-based audio retrieval from text queries". Proc. of the ACM MIR'08, Vancouver, Canada, October 2008.

[2] D. Grangier and A. Vinciarelli, "Effect of segmentation method on video retrieval performance". Proc. IEEE International Conference on Multimedia and Expo, pp. 5-8, Amsterdam, The Netherlands, 2005.

[3] O. Kalipzis, "Query processing in multimedia databases". Journal of Applied Science, Volume 2, pp. 109-113, 2002.

[4] H. Kosch and M. Döller, "Multimedia database systems: where are we now?" Institute of Information Technology, University Klagenfurt Universitätsstr. Austria, 2006.

[5] H. B. Kekre, "Image retrieval with shape features extracted using gradient operators and slope magnitude technique with BTC". International Journal of Computer Applications, Volume 6, Number 8, pp. 28-33, 2010.

[6] P. Sapra, S. Kumar, and R. K. Rathy, "Query processing in multilevel secure distributed databases". Proc. of the Fourth International Advance Computing Conference, 2014.

[7] P. Sapra, S. Kumar, R. K. Rathy, "Development of a concurrency control technique for multilevel secure databases". Proc. of the First International Conference on Reliability, Optimization and Information Technology, February 2014.

[8] X. Ma, D. Schonfeld and A. Khokhar, "Video event classification and image segmentation based on non-causal multi-dimensional hidden Markov models", IEEE Transactions on Image Processing, Vol. 18, No. 6, pp. 1304-1313, June 2009.

[9] B. V. Patel and B. B. Meshram, "Content based video retrieval systems". International Journal of UbiComp, Vol. 3, No. 2, pp. 13-30, 2012.

[10] T. C. Rakow, E. J., Neuhold and M. Lohr, "Multimedia database systems - the notions and the issues", Tagungsband GI-Fachtagung Datenbanksystems, Buro, Technik and Wissen-Schaft, Springer Informatik Aktuell, Berlin, 1995.

[11] C. Ribeiro and G. David, "A metadata model for multimedia databases". Proc. International Cultural Heritage Informatics Meeting, Archives and Museum Informatics, pp. 469-483, 2001.

[12] A. Burad, "Multimedia databases". Seminar Report, Roll No : 03005009, Computer Science and Engineering, Indian Institute of Technology, India, 2006.

[13] L. Lord and C. Pratt, "Retrievals from DB2 BLOB (Binary Large Objects) data warehouse using SAS". Proc. of NESUG Conference, 2000.

[14] D. Srivastava and Y. Velegrakis, "MMS: using queries as data values for metadata management". Proc. of the International Conference on Data Engineering, 2007.

[15] B. Panda, W. Perrizo, R. A. Haraty, "Secure transaction management and query processing in multilevel secure database systems". Proc. of the ACM Symposium on Applied Computing (ACM SAC 1994), pp. 363-368, 1994.

[16] R. A. Haraty and R. C. Fany, "Query acceleration in distributed database systems". Colombian Journal of Computation. Volume 2, Number 1, pp. 19-34, 2001.

[17] J. F. Abbass and R. A. Haraty, "Bit-level locking for concurrency control". Proc. of the Seventh ACS International Conference on Computer Systems and Applications (AICCSA 2009) – Sponsored by IEEE. Rabat, Morocco, pp. 168-173, May 2009.

Copyright (c) IARIA, 2015. ISBN: 978-1-61208-415-2