Efficiency Analysis in Content Based Image Retrieval Using RDF Annotations

Carlos Alvez[1], Aldo Vecchietti[2]

[1] Facultad de Ciencias de la Administración, Universidad Nacional de Entre Ríos
Concordia, 3200, Argentina
[2] INGAR – UTN, Facultad Regional Santa Fe
Santa Fe, S3002GJC, Argentina

Abstract. Nowadays it is common to combine low-level and semantic data for image retrieval. The images are stored in databases and computer graphics algorithms are employed to get the pictures. Most of the works consider both aspects separately. In this work, using the capabilities of a commercial ORDBMS a reference architecture was implemented for recovering images, and then a performance analysis is realized using several index types to search some specific semantic data stored in the database via RDF triples. The experiments analyzed the mean recovery time of triples in tables having a hundred of thousands to millions of triples. The performance obtained using Bitmap, B-Tree and Hash Partitioned indexes are analyzed. The results obtained with the experiences performed are implemented in the reference architecture in order to speed up the pattern search.

Keywords: Image retrieval, Semantic data, RDF triples, Object-Relational Database.

1 Introducción

Recovering images by content in a database requires the use of metadata which can be of several types: low-level describing physical properties like color, texture, shape, etc.; or high level metadata describing the image: the people on it, the geographic place or the action pictured, e.g. a car race.

Most of the works dealing with image recovering are limited by the difference between the low-level information and the high level semantic annotations. This difference is due to the diverse perception between the low-level data extracted by the programs, and the interpretation the user has for the image [1]. To cover this limitations the actual tendency is to combine in the same approach the low-level and semantic data. On other hand most of the articles in the open literature treat separately the database management aspects of the image retrieval from the computer vision issues [2]; however, in the commercial nowadays Data Base Management Systems (DBMS) it is possible to get sophisticated tools to handle and process high level data, having the capacity to formulate ontology assisted queries and/or semantic inferences.

In this sense, Alvez y Vecchietti [3] presented a software architecture to recover images from an Object Relational Database Management System (ORDBMS) [4]

using physical and semantic information. This architecture behaves as an extension of the SQL language in order to facilitate the usability of the approach. The low-level and high level information are combined maximizing the use of the tools provided by the DBMS. The architecture is based on several User Defined Types (UDT) containing attributes and methods needed to recover images based on both data types. The semantic information is added by means of the RDF (Resource Description Framework) language and RDF Schema. In this work it was shown that, although the RDF language was created for data representation in the Word Wide Web, it can be perfectly used to recover images in the database. The main advantages of using RDF/RDFS are its simplicity and flexibility, since by means of a triple of the form (*subject property object*) it is possible to represent a complete reference ontology or classes and concepts of that ontology and to make inferences among the instances. In this work an extension of that architecture is presented for the case where millions of triplets are stored to represent the images semantic data. The idea behind this work is to speed up the search of the triples involved in pattern search. In order to fulfill this objective, several experiments are driven in Oracle 11g ORDBMS analyzing the behavior of several indices: Bitmap, B-Tree and Hash Partitioned Indexes. The conclusions obtained in this analysis are then implemented in the reference architecture.

The article is outlined as follows: in section 2 the related work is introduced, in section 3 the ORDBMS architecture is described, in section 4 the performance analysis made is presented: the indexes used, the experiments performed and the results obtained; and finally in section 5 the conclusions are included.

2 Related Work

In the last years it is possible to find in the open literature articles dealing with the integration of low-level and semantic data and also improving the efficiency recovering images by means of RDF triplets. RETIN is a search engine developed by Gony et al. [5] with the objective of diminishing the semantic gap. The approach is based on the communication with the user which is continuously asked to refine the query. The interaction with the user is composed of several binary levels used to indicate if a document belongs to a category or not.

SemRetriev by Popescu et al. [6] is a system prototype which uses an ontology in combination with CBIR (*Content Based Image Retrieval*) techniques to structure a repository of images from the Internet. Two methods are employed for recovering pictures: a) based on keywords and b) in visual similarities; in both cases the algorithm is used together with the proposed ontology.

Döller y Kosch [7] proposed and extension of an Object-Relational database to retrieve multimedia data by physical and semantic content based on the MPEG-7 standard. The main contributions of this system are: a metadata model based on MPEG-7 standard for multimedia content, a new indexation method, a query system for MPEG-7, a query optimizer and a set of libraries for internal and external applications.

The main drawbacks of the works cited before are that they are difficult to implement, they are not flexible to introduce modifications, requires certain expertise in computer graphics and the learning curve is steep.

In the work of Fletcher y Beck [8] the authors present a new indexation method to increase the joins efficiency using RDF triplets. The novelty consists on generating the index using the triple atom as an index key instead of the whole triplet. In order to access the triple they use a bucket containing pointers to them having the corresponding atom value. For example, if K is the atom value of a triple, then three buckets can be created, the first one has pointers to the triplets having the form *(K P O)*, the second with those with the form *(S K O)* y the third *(S P K)*, where *S, P* and *O* are *Subject*, *Property* and *Object* respectively. The problem with this approach is that does not take into account issues like the key or the join selectivity, which can increase the cost of recovering images in the occurrence of a high key or join selectivity.

Atre et. al. [9] introduced BitMat, which consists of a compressed bit matrix structure to store big RDF graphs. They also proposed a new method to process the joins in the query language RDF SPARQL [10]. This method employs an initial prune technique followed by a linked variable algorithm to produce the results. This allows performing bit to bit operations in queries having joins.

In the approach presented in this paper in Section 4, similar structures to the one proposed in [8] and [9] are analyzed where its implementation is performed in a simple manner using the components provided by the ORDBMS adopted.

3 Reference Architecture

The reference architecture was implemented in Oracle 11g ORDBMS, it allows the image retrieval using CBIR techniques, semantic data and the combination of both. It has a three level structure: physical (low-level), semantic (high-level) and an interface linking them.

The semantic annotations of the images are stored in triples together a reference ontology. In Fig. 1 it is shown a graph with three classes related with the property *subClassOf*. The graph and the references to the image are stored in a table. Besides, the inferred instances can be also stored as it is shown in Table 1.

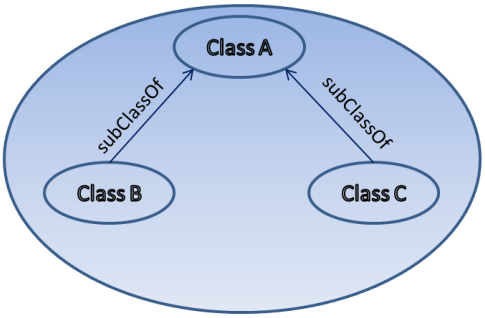

Fig. 1. RDF graph example.

Table 1. RDF/RDFS triples with inferred triples (rows *i*, *k*).

row	**Subject**	**Property**	**Object**
1	Class A	Rdf:type[1]	Rdfs:Class[2]
2	Class B	Rdfs:subClassOf[3]	Class A
3	Class C	Rdfs:subClassOf	Class A
4	Image 1	Rdf:type	Class C
5	Image 2	Rdf:type	Class B
	…	…	…
i	*Image 1*	*Rdf:type*	*Class A*
k	*Image 2*	*Rdf:type*	*Class A*

The references to the images stored in the database are implemented by image OIDs (Object Identifiers), in this way Oracle assigns to each Object-Table row a unique 16 bites long OID generated by the system that permits an unambiguous object identification in distributed systems. The architecture details and its implementation can be seen in [3].

The architecture was implemented in the database by means of several UDTs (*User Defined Types*) composed of attributes and operations. These methods plays a fundamental role in recovering images, they consist of set operations allowing the combination of semantic and low-level data. The physical content and the high-level information are managed separately and then they are related using the OID obtained in the queries and the set operations: union, intersection and difference, as it is shown in Fig. 2.

In Fig. 2, *similar* is an operation defined to recovery image *OIDs* with some physical properties. Basically, the method is defined as follows: *similar(d, t)*: *SetRef*, where *d* is the physical property to employ in the search and *t* is the threshold or distance allowed respect to a reference image. This function returns the *OIDs* of the images having a lower threshold respect to the image used as a reference. The function *semResultSet(p, o): SetRef* is defined for the semantic level, where *p* is a property and *o* an object. The function returns references to images having a matching with the property an object specified. Both functions returns a set of *OIDs* (*SetRef* type) referencing images stored in an Typed-Table. Having the sets of *OIDs* is now very simple to combine and operate with them by means of the set operations:

union(SetRef, SetRef): SetRef
intersection(SetRef, SetRef): SetRef
difference(SetRef, SetRef): SetRef

Since both *similar* and *semResultSet* methods return a *SetRef* type, then any combinations of the results set is valid and can be combined in the following form:

[1] Rdf:type, is a short name of: http://www.w3.org/1999/02/22-rdf-syntax-ns#type.
[2] Rdfs:Class, is a short name of: http://www.w3.org/2000/01/rdf-schema#Class.
[3] Rdfs:subClassOf, is a short name of: http://www.w3.org/2000/01/rdf-schema#subClassOf.

Op(similar (d_i, t_i), similar(d_j, t_j)): SetRef
Op(semResultSet(p_n, o_n), similar(d_k, t_k)): SetRef
Op(semResultSet(p_m, o_m), semResultSet(p_q, o_q)): SetRef

where *(d_i, t_i)* represent descriptors and threshold respectively and (p_n, o_n) are property and object. With these operators it is possible also to pose low-level queries with different descriptors and also semantic queries having diverse patterns. Note that the functions can be used recursively and their return can be used as an input parameter to other method. In the following example, the function intersection receives as an input the results obtained in the union between *semResultSet* and *similar*, and also the result obtained in the difference of two calls to the function *semResultSet*.

intersection(union(semResultSet(p_n, o_n), similar(d_i, t_i)),
diference(semResultSet(p_m, o_m), semResultSet(p_q, o_q)))

In the next section, it is presented the study about the alternatives to improve the efficiency in the queries invoking the function *semResultSet*.

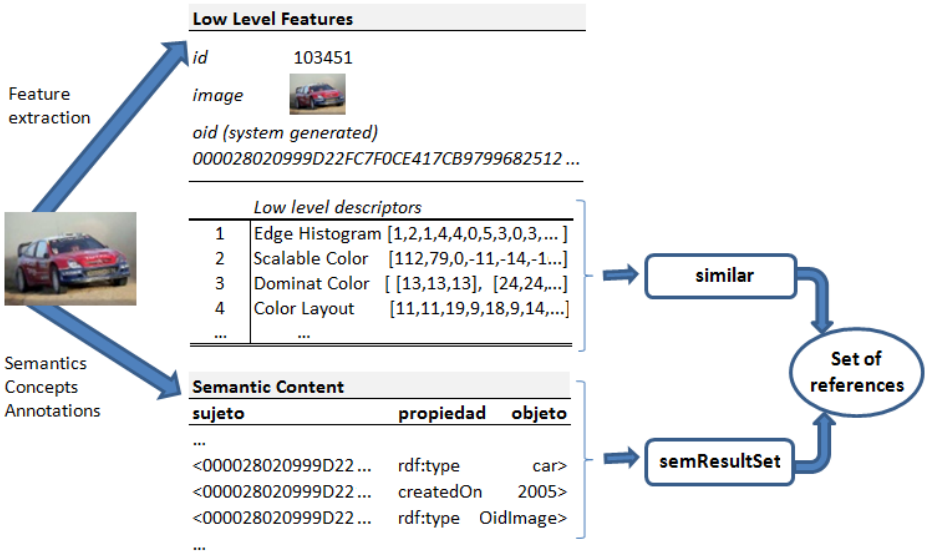

Fig. 2. Physical and Semantic data representation and its relation using the OID.

4 Performance analysis using different indexation methods

4.1 Issues about efficiency

The purpose of this work is to improve the efficiency of the reference architecture when the number of triples stored in the database is large. First it must be considered

that the subject (S) is the value to find, it means that every query has the following form *(? P O)* where *P* and *O* are property and object respectively. For queries where the subject (image to recover) is the value to find are three possible search pattern options:

a. (*?s* P *?o*)
b. (*?s ?p* O)
c. (*?s* P O)

and for composed patterns the set operations are used.

The *property* attribute is employed in patterns *(?s* P *?o*) and an index is created to improve the speed of the search, for patterns (?*s* ?*p* O) the *object* attribute is employed and for (*?s* P O) the index can be generated using the attributes object and property together, or a combination of the previous individual indexes.

4.2 Tests performed

For the efficiency analysis several index types are generated: Bitmap, B-Tree and Hash partitioned indexes; all of them provided in Oracle 11g DBMS. The Bitmap index was selected because it is appropriated for cases similar to the one analyzed in this article: the key has a low cardinality (high selectivity). In this structure, a bit map is constructed for each key value pointing to the block where a database register contains the data associated to the key. Other advantages of this index type are that needs lower space than traditional B-Tree indexes and some comparison operations using bits are executed faster in computer memory.

The traditional B-Tree index structure is in the opposite site of the Bitmap, so it is not appropriated for low cardinality attribute, it is used in this paper just for comparison reasons. In section 5, the results of the test show that the behavior of this structure was not so bad as was expected.

The Hash Partitioned Index is an intermediate structure where a database Table is partitioned according to an attribute selected, and a regular B-Tree index is created for each partition. The number of partition to be generated must be selected; in our case 4 partitions were created.

For the test performed, the database was loaded with different amount of triples extracted from UniProt [11]: 500,000, 2,000,000 and 10,000,000; and the average recovery time was determined using the indexes constructed. The experiments were executed on a PC CPU-INTEL CORE 2 Duo Processor 3.0 GHz, with 8 GB RAM and a 7200 rpm disk, running in Windows 2003. One hundred (100) queries were executed over the three set of triples using different selectivity values for the properties. As was explained before, selectivity counts for the number of times that the property value is repeated over the triples. The average execution times (in seconds) obtained for search pattern *a*, *b* and *c* are shown in Fig. 3.

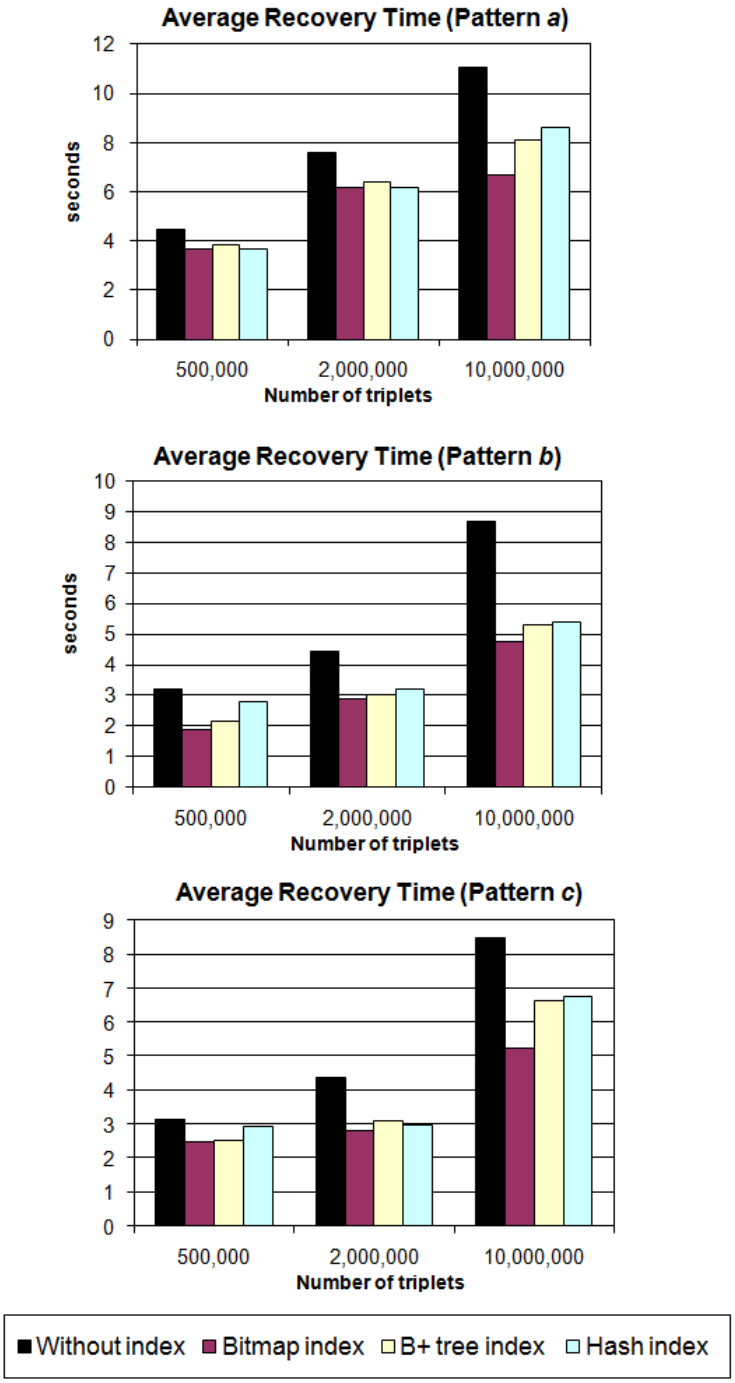

Fig. 3. Average recovery time for pattern *a*, *b* and *c*.

From Fig. 3 can be seen that the *bitmap index* has the better performance when the number of triplets increases. However, note that the results obtained without using an index are in the order of those employing it. In order to have an insight about this issue, a performance comparison was made between the Bitmap index and without it using triples attribute values of diverse selectivity. The results obtained can be seen in Fig. 4. From Fig. 4 it is clear that the advantage of the index diminishes when the number of triples and/or the attribute selectivity increases. This situation is very common when using a RDF graph particularly considering a property attribute. In the same direction another test was performed using *Oracle hints*, by means of this capability (hints) the query optimizer is instructed to execute the query using a specific pathway. In this case for pattern a) the average execution time was improved using the following hint:

/*+ INDEX (tripet_t ix_p) CACHE(t) */.

The first part of the hint indicates to the optimizer what index type to use and the second instructs the optimizer to place the blocks retrieved for the table at the most recently used end of the LRU (Last Recently Used) list in the buffer cache. This is particularly important for this search pattern since it is likely to make several search for the same property, for example Rdf:type. In Fig. 4 the results with the hint are shown with a green line that compared with the red one (Bitmap index without hints) can be observed the improvement in the average execution time.

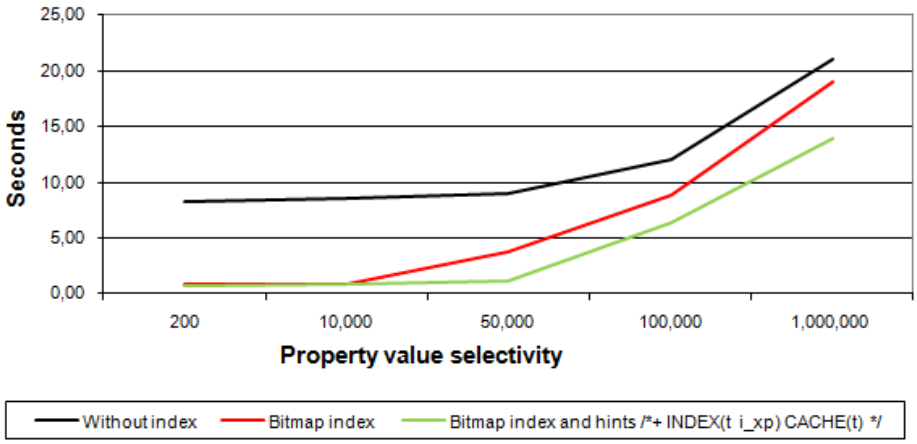

Fig. 4. Average recovery time for pattern *a.*

Similar results can be obtained for pattern *b*.

In the case of pattern *c*, no improvement was obtained using the previous hints using the index generated for the composed attributes; so a test was made using the combination of the individual indexes (for property and object attributes)using the following hint:

/*+ INDEX_COMBINE(t ixp ixo) CACHE(t) */

The hints INDEX_COMBINE explicitly chooses a bitmap access path for the table. If no indexes are given as arguments for the INDEX_COMBINE hint, the optimizer uses whatever Boolean combination of bitmap indexes has the best cost estimate for the Table. If certain indexes are given as arguments, the optimizer tries to use some Boolean combination of those particular bitmap indexes by using a conjunctive (AND) bitwise operation. The results obtained are shown in Fig. 5, where again the use of the hint improves the performance.

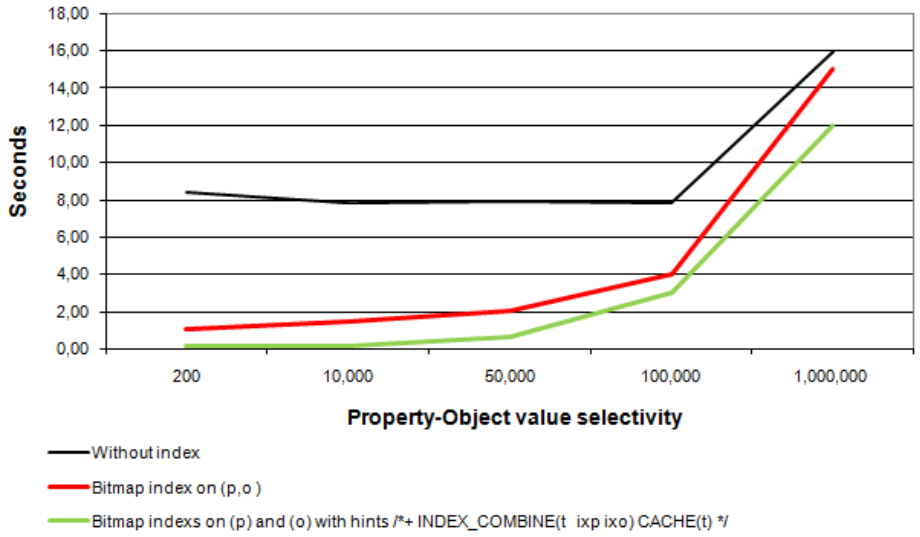

Fig. 5. Average recovery time for pattern *c*.

4.3 Index Implementation in the Reference Architecture

Based on the results obtained, the implementation of a *User Defined Function* (UDF) is proposed to execute the pattern search of triples using Bitmap indexes. The function is called *search_subject*(*p*, *o*). This UDF is employed by the method *semResultSet* described in Section 3. For this purpose, an UDF similar to SEM_MATCH [12] is created but in this case this function takes the subject as a default value to search. The parameters *p* and *o* represents the property and object of the triples respectively; when the function receives a parameter with a question mark this one becomes the value to search, for example a call like *search_subject (´?p´, ´car´)* means that triples having the property ´car´ must be get.

The function parameter *p* it is just used to get the triples matching with that criteria, once having the triple, the next step is to find the subjects *(OIDs)* related to that search pattern pointing at the images stored in the database. In this sense, the pattern *(´?s´, ´rdf:type´, ´oidImage´)* must be implicitly satisfied to get the OIDs of the images.

In Fig. 6 it is shown an example about the use of the Bitmap indexes to find the triples matching with the search pattern.

row	**Subject**	**Property**	**Object**
1	Vehicle	rdf:type	rdfs:Class
2	oidImage	rdf:type	rdfs:Class
3	Light_Truck	rdfs:subClassOf	Vehicle
4	Car	rdfs:subClassOf	Vehicle
5	Truck	rdfs:subClassOf	Vehicle
6	Omnibus	rdfs:subClassOf	Vehicle
7	Furgon	rdfs:subClassOf	Light_Truck
8	PickUp	rdfs:subClassOf	Light_Truck
9	2_doors	rdfs:subClassOf	Car
10	3_doors	rdfs:subClassOf	Car
11	4_doors	rdfs:subClassOf	Car
12	Wagon	rdfs:subClassOf	Car
13	Wagon	rdfs:subClassOf	Car
14	Van	rdfs:subClassOf	Wagon
15	Station_wagon	rdfs:subClassOf	Wagon
16	*oid1*	rdf:type	Car
17	*oid1*	rdf:type	oidImage
18	*oid2*	rdf:type	Car
19	*oid2*	rdf:type	oidImage
20	*oid3*	rdf:type	Wagon
21	*oid3*	rdf:type	oidImage
22	*oid4*	rdf:type	Van
23	*oid4*	rdf:type	oidImage

Bitmap on *Property*

Valor/fila	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
rdf:type	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
rdfs:subClassOf	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Bitmap on *Object*

Valor/fila	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
rdfs:Class	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Vehicle	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Camioneta	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Light_Truck	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	0	1	0	0	0	0	0
Wagon	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0
Van	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
oidImage	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1

Fig. 6. The triples specification using car taxonomy and its instances are shown in the top of the figure; below, the Bitmap indexes generated with those triples.

Using a query *search_subject (´?p´, O)* the Bitmap index created for *object* is employed. For example the query *search_subject (´?p´, 'Car')* retrieves the rows 10-13, 16 and 18, then only the subjects of those rows must be taken into account; but not all of them are included in the final results only those satisfying the pattern *(´?s´, ´rdf:type´, ´oidImage´)* because they have OIDs values referencing images in the database.

For a query of type *search_subject(P, ´?o´)* it is used the index created for the *property* column. For example the query *search_subject(´Rdf:type´, ´?o´)* get the rows 1, 2, 16-23, then these rows must be intersected with subjects having the pattern *(´?s´, ´rdf:type´, ´oidImage´)*.

Finally, when the query has the form *search_subject (P, O)* the intersection of the Bitmap indexes over the columns *property* and *object* must be used. For example, the query *search_subject (´Rdf:type´, 'Car')*having the map property recovers the rows 1, 2, 16-23 and with the bitmap index *object* rows 10-13, 16,18 are obtained. The intersection are the rows 16 and 18; the subject of those rows are intersected with the subject of the pattern *(´?s´, ´rdf:type´, ´oidImage´)* to get the images in the final result.

5 Conclusions

In this work it is presented a performance analysis for recovering semantic data stored in an Object-Relational database in the form of RDF triples. Different indexation methods are selected to perform the analysis. The triples are used to relate images with its semantic information via the OIDs created by the ORDBMS when the image is stored in a Typed-Table. The goal pursued with the use different indexation methods is to improve the efficiency in recovering the image via a faster retrieve of the OIDs. A Reference Architecture was employed to drive the test and also to implement the results obtained.

One conclusion arrived in this work indicates that the Bitmap index has a better performance compared to the B-tree and Hash Partitioned indexes when the RDF graph is composed of thousands and millions of triples. All the experiments were executed using Oracle 11g ORDBMS. Another conclusion verified was that the combination of two individual Bitmap indexes has a better performance than the composed one over property and object columns. The use of hints may improve the efficiency when used appropriately.

Based on the previous conclusions, the Bitmap index together with the *search_subject* UDF function were implemented to speed up the RDF triples search and as a consequence the image recovery. It is important to note that the architecture, the index and functions used, are all implemented using tools that are provided by most of the nowadays commercial ORDBMS, which facilitates its realization.

Referencias

1. Neumamm D. and Gegenfurtner K. "Image Retrieval and Perceptual Similarity" ACM Transactions on Applied Perception, Vol. 3, No. 1, January 2006, Pages 31–47.

2. Carlos Alvez, Aldo Vecchietti. A model for similarity image search based on object-relational database. IV Congresso da Academia Trinacional de Ciências, 7 a 9 de Outubro de 2009 - Foz do Iguaçu - Paraná / Brasil (2009).
3. Carlos E. Alvez, Aldo R. Vecchietti. Combining Semantic and Content Based Image Retrieval in ORDBMS. Knowledge-Based and Intelligent Information and Engineering Systems Lecture Notes in Computer Science, 2010, Volume 6277/2010, pp. 44-53. Springer-Verlag Berlin Heidelberg (2010).
4. Melton Jim, "(ISO-ANSI Working Draft) Foundation (SQL/Foundation)", ISO/IEC 9075-2:2003 (E), United States of America (ANSI), 2003.
5. Gony J., Cord M., Philipp-Foliguet S. and Philippe H. "RETIN: a Smart Interactive Digital Media Retrieval System". ACM Sixth International Conference on Image and Video Retrieval – CIVR 2007 - July 9-11, 2007, Amsterdam, The Netherlands, pp. 93-96, (2007).
6. Popescu A., Moellic P.A. and Millet C. "SemRetriev – an Ontology Driven Image Retrieval System". ACM Sixth International Conference on Image and Video Retrieval – CIVR 2007 - July 9-11, Amsterdam, The Netherlands, pp. 113-116, (2007).
7. Mario Döller, Harald Kosch. The MPEG-7 Multimedia Database System (MPEG-7 MMDB). The Journal of Systems and Software 81, pp. 1559–1580. Elsevier (2008).
8. George H. L. Fletcher, Peter W. Beck: Scalable indexing of RDF graphs for efficient join processing. ACM Conference on Information and Knowledge Management CIKM 2009: 1513-1516. (2009).
9. Medha Atre, Vineet Chaoji, Mohammed J. Zaki, and James A. Hendler. Matrix "Bit"loaded: A Scalable Lightweight Join Query Processor for RDF Data International World Wide Web Conference Committee (IW3C2). April 26–30, 2010, Raleigh, North Carolina, USA. ACM (2010).
10. Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 15 January 2008.
11. UniProt RDF. http://dev.isb-sib.ch/projects/uniprot-rdf/.
12. Eugene Inseok Chong, Souripriya Das, George Eadon and Srinivasan, Jagannathan. An efficient SQL-based RDF querying scheme. Proceedings of the 31st international conference on Very large data bases, VLDB '05, pp. 1216—1227. Trondheim, Norway. (2005).