Mejora de la Eficiencia en la Recuperación de Imágenes en Bases de Datos por Mediante Indexación Semántica

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

Resumen. En este trabajo se presenta una forma de indexación que permite recuperar de manera eficiente metadatos introducidos en tripletas (RDF) en tablas OR. Estas tripletas son utilizadas para relacionar a imágenes con conceptos semánticos. Las imágenes se relacionan con los conceptos semánticos mediante referencias a registros de imágenes. La propuesta de indexación, tiene como objetivo mejorar el tiempo de recuperación de las referencias almacenados en las tripletas. Se realizaron pruebas con diferentes cantidades de tripletas. Los resultados en pruebas empíricas, muestran que la tasa media de recuperación para volúmenes importantes de tripletas disminuye aproximadamente un 50%.

Palabras Clave: Recuperación de imágenes, semántica, bases de datos. Mapas de bits.

1 Introducción

En la última década, se han realizado varias propuestas para la recuperación de imágenes basada en contenido (CBIR). La mayoría de ellas están limitadas por la diferencia que separa la información de bajo nivel (color, textura, forma, etc.) de sus anotaciones semánticas. Esta diferencia se refiere más precisamente a la percepción distinta que existe entre los datos de bajo nivel extraídos por los programas y la interpretación que el usuario tiene para la misma imagen [1]. Para corregir estas limitaciones, la tendencia actual, en recuperación de imágenes, es la integración bajo un mismo sistema, de los datos de bajo nivel y los semánticos. En general, los sistemas relacionados con la recuperación de imágenes pueden verse como un conjunto de módulos complejos de procesamiento de imágenes almacenadas en una base de datos, teniendo éstas últimas el protagonismo de ser soporte de información. La mayoría de los trabajos de la literatura tratan por separado los sistemas de gestión de base de datos y visión por computadora (*computer vision*) [2]. Sin embargo, los DBMS comerciales actuales, ofrecen un conjunto de programas y herramientas que permiten un manejo sofisticado de la información semántica y su procesamiento,

otorgando la capacidad de formular consultas asistidas por ontologías ó realizar inferencias semánticas por medio de ellas.

En este sentido, en Alvez y Vecchietti [3] presentaron una arquitectura de software para hacer frente a la recuperación de la imagen en un Sistema de Gestión de Bases de Datos Objeto-Relacional (ORDBMS) [4], en la propuesta combinaron características de bajo nivel y rasgos semánticos de las imágenes, con el objetivo de maximizar la utilización de componentes y herramientas que el DBMS proporciona. Esta arquitectura se compone de varios tipos definidos por el usuario (UDT), que consisten de atributos y métodos necesarios para gestionar datos físicos y semánticos. Los datos semánticos se introducen en el lenguaje RDF (Resource Description Framework) y RDFS (RDF Schema). Por medio de este trabajo se demostró que, si bien RDF es un lenguaje para la representación de recursos en la World Wide Web, estos se adaptan perfectamente para la recuperación de imágenes por contenido semántico en Bases de Datos. La principal ventaja que brinda el empleo de RFD/RDFS es su flexibilidad dado que, mediante la inserción de tripletas, se puede representar una ontología de referencia, instancias de clases (o conceptos) afirmadas (*asserted*) en dicha ontología, e instancias inferidas. Por ejemplo, en la Tabla 1, se presenta un conjunto de tripletas en las que se definen: una clase con dos subclases, instancias afirmadas de clases (*asserted instances*) e instancias inferidas.

Tabla 1. Ejemplo de tripletas RDF/RDFS con tripletas afirmadas e inferidas (filas *i*, *k*).

fila	**Sujeto**	**Propiedad**	**Objeto**
1	Rodado	Rdf:type[1]	Rdfs:Class[2]
2	Automóvil	Rdfs:subClassOf[3]	Rodado
3	Autobús	Rdfs:subClassOf	Rodado
4	Corsa Classic	Rdf:type	Automóvil
5	VW Gol	Rdf:type	Automóvil
	…	…	…
i	*Corsa Classic*	*Rdf:type*	*Rodado*
k	*VW Gol*	*Rdf:type*	*Rodado*
	…	…	…

Si bien la arquitectura presentada cumple con los objetivos de integrar los datos de bajo nivel con los semánticos y permite una correcta recuperación de imágenes, no se tuvieron en cuenta en su diseño original cuestiones de eficiencia, que es un aspecto importante a tener en cuenta cuando se trabaja con un conjunto grande de tripletas. En este trabajo se presenta un diseño para mejorar la eficiencia mediante la utilización de índices de mapas de bits. Los índices de mapas de bits, consisten básicamente en una grilla, en la cual para cada valor de la columna indexada, se crea un registro para

[1] Rdf:type, es una abreviatura de: http://www.w3.org/1999/02/22-rdf-syntax-ns#type.
[2] Rdfs:Class, es una abreviatura de: http://www.w3.org/2000/01/rdf-schema#Class.
[3] Rdfs:subClassOf, es una abreviatura de: http://www.w3.org/2000/01/rdf-schema#subClassOf.

donde se indica, por medio de un bit con valor 1, si ese valor existe para el dato almacenado en la base de datos o un bit en 0 si no existe. Por ejemplo, en la Tabla 2, se muestra un índice de mapa de bits, para indexar el campo *Objeto* de la Tabla 1.

Tabla 2. Ejemplo de índice de mapa de bit, indexando el campo Objeto de la Tabla 1.

Valor/fila	**1**	**2**	**3**	**4**	**5**	**…**	**i**	**k**	**…**
Rdfs:Class	1	0	0	0	0	…	0	0	…
Rodado	0	1	1	0	0	…	1	1	…
Automóvil	0	0	0	1	1	…	0	0	…
…	…	…	…	…	…	…	…	…	…

El empleo de un índice de mapa de bits para la búsqueda de imágenes por contenido semántico se basa en que, cuando se crea una ontología de referencia para indexar un conjunto grande de imágenes, muchas de estas últimas pueden tener un mismo valor una clase ó propiedad haciendo que esta estructura en principio se adapte más adecuadamente a esta característica.

En lo que sigue, este artículo se presenta de la siguiente manera: en la sección 2, se introducen los trabajos relacionados con el objetivo de este artículo; en la sección 3, se describe la arquitectura de referencia que se emplea para la implementación; en la sección 4, la propuesta de indexación para mejorar la eficiencia y en la sección 5 las conclusiones.

2 Trabajos Relacionados

En la literatura, en los últimos años, es posible encontrar trabajos que proponen la integración de los metadatos físicos y semánticos así como también, la mejora de eficiencia en la recuperación de contenido por medio del almacenamiento de tripletas RDF. RETIN es un motor de búsqueda desarrollado por Gony et al. [5] con el objetivo de disminuir la diferencia semántica. Para hacerlo propone un proceso interactivo de consulta al usuario que permite refinar la búsqueda tanto como sea necesario. La interacción con el usuario se compone de varios niveles binarios que sirve para indicar si el documento pertenece a una categoría o no.

SemRetriev propuesto por Popescu et al. [6] es un prototipo de sistema que utiliza una ontología para estructurar un repositorio de imágenes presentes en Internet en combinación con técnicas CBIR. Se emplean dos métodos para la recuperación de imágenes: basadas en palabras clave y en similitudes visuales en combinación con la ontología propuesta para ambos casos.

En el trabajo de Döller y Kosch [7] se realiza una propuesta que consiste en la extensión de una Base de datos Objeto-Relacional para la recuperación de datos multimedia por contenido físico y semántico basado en el estándar MPEG-7. Los principales aportes de este sistema son: un modelo de metadatos para el contenido multimedia en base al estándar MPEG-7, una nueva indexación, un sistema de

consulta para MPEG-7, el optimizador de consultas y el apoyo a las bibliotecas de aplicaciones internas y externas.

Los trabajos citados anteriormente son muy difíciles de implementar, tienen problemas de flexibilidad y adaptación a los cambios, se requiere un elevado grado de conocimiento y experiencia en computación gráfica, y la curva de aprendizaje es muy empinada. El objetivo de nuestra propuesta es que pueda ser empleada por usuarios y/o desarrolladores no especialista en la temática, pero con un grado de conocimiento medio en Bases de Datos.

Existen trabajos que se enfocan en mejorar la eficiencia en la recuperación de contenido almacenados en tripletas, Fletcher y Beck [8] presentan un novedoso método de indexación de para aumentar la eficiencia de los *joins* en RDF. La novedad consiste en que el índice se construye utilizando como claves los átomos que ocurren en el conjunto de tripletas, en lugar de toda la tripleta. Los átomos están en el índice, independientemente de las funciones que desempeñan (es decir, como sujeto, propiedad, u objeto). Para acceder a las tripletas se utilizan cajones (buckets) con punteros hacia aquellas tripletas que contienen dicho átomo. Por ejemplo, si la clave *K* es el valor del átomo, habrá tres cajones. El primero con punteros hacia las tripletas de la forma (*K P O*), el segundo para las aquellas de la forma (*S K O*) y el tercero para la forma (*S P K*) donde *S*, *P y O* son *sujeto*, *propiedad* y *objeto* respectivamente. El problema de esta propuesta es que no tiene en cuenta cuestiones como la selectividad de las claves y de los *joins*, lo que puede aumentar el costo en algunos casos como por ejemplo, una baja selectividad de las claves (*átomos* utilizados en los patrones de búsqueda) y alta selectividad en los *joins*.

En Atre et. al. [9] se presentó BitMat, que consiste en una estructura de matrices de bits comprimida para almacenar enormes gráficos RDF, y un nuevo método de procesamiento de *joins* en el lenguaje de consultas para RDF SPARQL [10]. Este método emplea una técnica de poda inicial seguido de un algoritmo variable vinculante, que permiten a BitMats producir los resultados finales. Este método de procesamiento de consultas no construye resultados intermedios al reunir tablas y trabaja directamente en los datos comprimidos. Para esto se representa un grafo RDF como un cubo de bits en 3D, donde las dimensiones son: *sujeto*, *propiedad* y *objeto*, similar a *RDFCube* [11]. Este cubo en 3D es particionado para obtener matrices en 2D (BitMats). Se crean BitMaps S-O y O-S para cada P; P-O para cada S y P-S para cada O; de esta manera se tiene un total de $2Vp + Vs + Vo$ BitMats, donde *Vp*, *Vs* y *Vo* son la cantidad de *sujetos, propiedades* y *objetos* diferentes respectivamente. Esta estructura permite realizar operaciones bit a bit en consultas con *joins*.

En nuestra propuesta, que se presenta en la sección 4, se utilizará estructuras similares, pero más simples de implementar con los elementos que provee una BDOR, adaptadas a la arquitectura que se describirá en la sección 3.

3 Arquitectura de referencia

La arquitectura de referencia, que se implementó en un Sistema de Gestión de Base de Datos Objeto-Relacional (Oracle 11g), permite la recuperación de imágenes por contenido físico y semántico. Esta arquitectura está pensada como una extensión del

lenguaje SQL para que sea simple de utilizar por usuarios de bases de datos. En la misma se pueden recuperar imágenes realizando consultas semánticas, basadas en contenido físico y combinaciones de ambas. La misma se estructura en tres niveles: nivel físico (o de bajo nivel), nivel semántico (alto nivel) y una interfaz que los integra.

La arquitectura es implementada en la base de datos mediante un conjunto de UDTs (*User Defined Types*) los cuales están compuestos por atributos y operaciones (funciones y procedimientos). Los atributos de contenido físico y semántico, se gestionan y consultan por separado, relacionándose mediante el *OID* (*Object Identifier*) del registro de la imagen como se muestra en la Fig. 1. Oracle asigna a cada fila de una Tabla-Objeto (*Object Table*) un *OID* único generado por el sistema de 16 bytes de longitud, que en ambientes distribuidos permite que el objeto sea identificado sin ambigüedades. Los detalles de la arquitectura y su implementación se pueden ver en [3].

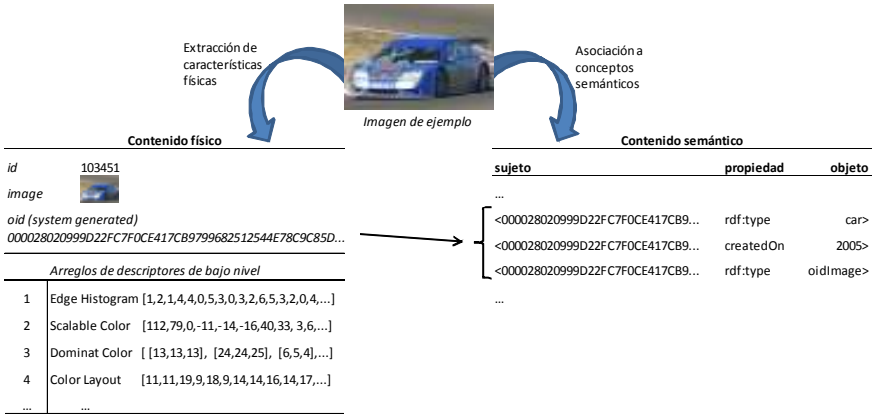

Fig. 1. Representación de metadatos físicos y semánticos de una imagen y su relación mediante el *OID* de la tabla que almacena las imágenes.

Si bien las consultas por contenido físico y semántico se realizan por separado, estas se integran mediante operaciones de conjuntos: unión, intersección y diferencia para que combinar los resultados obtenidos.

Básicamente la arquitectura funciona de la siguiente manera: se definieron dos funciones como miembros de las clases a nivel físico y semántico, a nivel físico se definió *similar(d, t)*: *SetRef*, donde *d* es el descriptor a utilizar y *t* es el valor umbral o distancia tolerada. La función retorna las referencias a imágenes que tengan una umbral menor a la imagen empleada como dato; a nivel semántico se define *semResultSet(p, o): SetRef*, donde *p* es una *propiedad* y *o* un *objeto*. La función retorna las referencias a imágenes que tengan un "*matching*" con la propiedad y el objeto especificado.

En ambas funciones el tipo retornado es una colección de *OIDs* (tipo *SetRef* [4]) a registros de la tabla de imágenes.

La combinación de consultas que integren ambos aspectos es ahora muy simple de realizar empleando el conjunto de operadores mencionados anteriormente:

union(SetRef, SetRef): SetRef
intersection(SetRef, SetRef): SetRef
diference(SetRef, SetRef): SetRef

Debido a que tanto el método *similar* como el método *semResultSet* retornan un tipo *SetRef*, entonces en el conjunto de operadores cualquiera de las siguientes combinaciones es válida:

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

donde los (d_i, t_i) descriptores y umbrales respectivamente y los (p_n, o_n) son pares de propiedad y objetos.

Esto significa que, no solo es posible combinar consultas a datos semánticas con datos de bajo nivel, sino que además, consultas de bajo nivel con diferentes descriptores o consultas semánticas con diferentes patrones. Cabe destacar que estas funciones se pueden utilizar de manera recursiva y la salida de cada operador puede ser utilizada como parámetro de entrada de otro operador. En el siguiente ejemplo, la function *intersection*, recibe como parámetros los resultado de la union entre funciones *semResultSet* y *similar*; y el resultado de la diferencia entre dos invocaciones a la función *semResultSet*.

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

En la siguiente sección se presenta una alternativa para mejorar la eficiencia en las consultas que utilicen las funciones *semResultSet*.

4 Índices de mapas de bits para mejorar la eficiencia en la búsqueda de imágenes.

Antes de analizar cuestiones de eficiencia, es importante destacar que con la arquitectura propuesta, se pueden evitar utilizar la mayoría de los *joins* en RDF. Por ejemplo, una consulta de la forma:

(s? P_1 O_1) ∧ (s? P_2 O_2)

[4] Oracle tiene dos formas de representar colecciones: *arreglos de longitud variables y tablas*. En este caso *SetRef* se define como tablas de *OIDs* a imágenes.

en la arquitectura de referencia se escribe como:

intersection(semResultSet(p_1, o_1), semResultSet(p_2, o_2))

De manera similar se puede utilizar el método *union* para la operación "$\vee$" (OR) y distintas combinaciones con *diference* para casos donde aparece se utilice "$\neg$" (NOT). De de esta manera, en lugar de realizar un *join* entre tripletas resultantes, se realiza una intersección/unión/diferencia de conjunto entre colecciones de *OIDs*, que hacen referencia directamente a la tupla donde está almacenada la imagen.

Esto no sólo es más eficiente que los *joins* entre tripletas, sino que también es más eficiente para recuperar la/s imagen/es, motivo de la consulta, dado que los *OIDs* hacen referencia no ambigua a una tupla aún en ambientes distribuidos.

4.1 Cuestiones de eficiencia.

El propósito de este punto, es mejorar la eficiencia en la arquitectura de referencia, para lo cual, se debe tener en cuenta que el sujeto (S) es siempre una incógnita. Esto significa que toda consulta va a tener un patrón simple (? P O) donde P y O son propiedad y objeto respectivamente.

Para las consultas en donde el sujeto (imagen a recuperar) es siempre la incógnita, se tienen tres opciones posibles:

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

y para patrones compuestos se utilizan operaciones de conjunto como ya se mencionó (no se tienen en cuenta patrones con tres incógnitas).

Nuestra propuesta por un lado, se basa en el trabajo de Fletcher [8], que consiste en indexar sólo átomos de las tripletas. Por otro lado, utilizar partes del cubo de bits (BitMats [9]) que sean convenientes para el tipo de consultas que se realizan en la arquitectura, con la idea de evitar varias combinaciones de BitMats innecesarias y además que sean unidimensionales.

Por lo tanto, para mejorar la eficiencia en las consultas con los patrones mencionados anteriormente, se tienen las siguientes alternativas:

a. Construir un BitMat de la forma P-O, similar al presentado por Atre et. al. [9].
b. Construir dos mapas de bits, uno para P y otro para O
c. Construir un mapa de bits para los campos PO
d. Construir tres mapas de bits: para los campos P, O y PO

Se eligió la segunda alternativa **(b.)** porque es la más apropiada para nuestro caso, dado que mejora la eficiencia en la consultas de cualquiera de los patrones *a*, *b* y *c* mencionados anteriormente. Se debe notar que los mapas de bits sobre P y O, a diferencia de los BitMats, sólo toman una dimensión.

Para responder a los patrones propuestos, se opera de la siguiente manera: para un patrón del tipo *a* (*?s ?p* O), se utiliza el mapa de bits sobre la columna O (Tabla 2); para un patrón de tipo *b* (*?s* P *?o*), un mapa de bits sobre la columna P (Tabla 3). Para un patón del tipo *c* (*?s* P O), necesitaría un mapa de bits sobre las columnas PO (Tabla 4). Sin embargo, se puede tener el mismo resultado utilizando la intersección entre los mapas de bits sobre P y O.

Por ejemplo, si se para responder a la consulta:

(s? Rdf:type Automóvil)

que se puede responder con la segunda fila de la Tabla 4 (**Rdf:type&Automóvil**), también se pude obtener el mismo resultado con la intersección (operación **And**) de los mapas de bit sobre O y P (tablas 2 y 3 respectivamente) con las claves **Rdf:type** en P y **Automóvil** en O (ver Tabla 5). La operación **And** es una operación bit a bit previstas por los SGBD.

De esta manera, para responder consultas para los patrones que permite la arquitectura propuesta, basta con indexar los átomos en O y en P.

Tabla 3. Ejemplo de índice de mapa de bit, indexando la columna Propiedad de la Tabla 1.

Valor/fila	**1**	**2**	**3**	**4**	**5**	**…**	**i**	**k**	**…**
Rdf:type	1	0	0	1	1	…	1	1	…
Rdfs:subClassOf	0	1	1	0	0	…	1	1	…
…	…	…	…	…	…	…	…	…	…

Tabla 4. Ejemplo de índice de mapa de bit, indexando las columnas Propiedad-Objeto de la Tabla 1.

Valor/fila	**1**	**2**	**3**	**4**	**5**	**…**	**i**	**k**	**…**
Rdf:type&Rdfs:Class	1	0	0	0	0	…	0	0	…
Rdf:type&Automóvil	0	0	0	1	1	…	0	0	…
Rdf:type&Rodado	0	0	0	0	0	…	1	1	…
Rdfs:subClassOf&Rodado	0	1	1	0	0	…	0	0	…
…	…	…	…	…	…	…	…	…	…

Tabla 5. Intersección entre tablas sobre P = **Rdf:type** y O = **Automóvil**.

Valor/fila	**1**	**2**	**3**	**4**	**5**	**…**	**i**	**k**	**…**
Rdf:type	1	0	0	1	1	…	1	1	…
Automóvil	0	0	0	1	1	…	0	0	…
∩	0	0	0	1	1	…	0	0	…

4.2 UDF search_subject

En este trabajo se propone el uso un UDF (*User Defined Function*) para la búsqueda de patrones en tripletas, llamada *search_subject*(*p*, *o*). Esta UDF es utilizada en el modelo por el método *semResultSet* descripto en la Sección 3. Para ello se generó un UDF similar a SEM_MATCH [12], con la diferencia que esta función toma implícitamente al sujeto como incógnita.

Los parámetros *p* y *o*, son cadenas de caracteres que representan la *propiedad* y el *objeto* de una tripleta respectivamente. Si la función recibe como parámetro una cadena que comienza con el signo de interrogación, la función toma este parámetro como una incógnita. Por ejemplo en *search_subject (´?p´, ´automovil´)*, significa que la propiedad *p* es la incógnita. El hecho de que *p* sea una incógnita, se utiliza sólo para la coincidencia (*matching*) de tripletas, dado que para el resultado de la función no interesa qué elemento se instancian en *p*, sino, como se instancian los *sujetos* (implícitos en ese patrón) que hacen referencias a imágenes. Esto último, significa que siempre debe cumplir implícitamente el patrón *(´?s´, ´rdf:type´, ´oidImage´)*, dado que nos interesan sólo aquellos sujetos que son *OIDs* a registros de imágenes.

En resumen, como se trato anteriormente, sólo tenemos tres tipos de patrones simples (‘*?p’*, o), (p, ‘?o’) y (p, o). En la Fig. 2 se muestra el ejemplo con el cual se realizaron las experiencias con los mapas de bits propuestos para encontrar las tripletas que coinciden con el patrón pasado como parámetro.

fila	Sujeto	Propiedad	Objeto
1	Rodado	rdf:type	rdfs:Class
2	oidImage	rdf:type	rdfs:Class
3	Camioneta	rdfs:subClassOf	Rodado
4	Automovil	rdfs:subClassOf	Rodado
5	Camion	rdfs:subClassOf	Rodado
6	Omnibus	rdfs:subClassOf	Rodado
7	Utilitario	rdfs:subClassOf	Camioneta
8	PickUp	rdfs:subClassOf	Camioneta
9	2Puertas	rdfs:subClassOf	Automovil
10	3Puertas	rdfs:subClassOf	Automovil
11	4Puertas	rdfs:subClassOf	Automovil
12	5Puertas	rdfs:subClassOf	Automovil
13	Familiar	rdfs:subClassOf	Automovil
14	Van	rdfs:subClassOf	Familiar
15	Rural	rdfs:subClassOf	Familiar
16	*oid1*	rdf:type	Automovil
17	*oid1*	rdf:type	oidImage
18	*oid2*	rdf:type	Automovil
19	*oid2*	rdf:type	oidImage
20	*oid3*	rdf:type	Familiar
21	*oid3*	rdf:type	oidImage
22	*oid4*	rdf:type	Van
23	*oid4*	rdf:type	oidImage

Bit map sobre *Propiedad*

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

Bit map sobre *Objeto*

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
Rodado	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
Automovil	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	0	1	0	0	0	0
Familiar	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. 2. A la izquierda se muestra un ejemplo de tripletas especificando una taxonomía de rodados e instancias y los mapas de bits generados para objetos y propiedades.

Ante una consulta del tipo *search_subject (´?p´, O)*, se utiliza en mapa de bits sobre la columna *objeto*. Por ejemplo, con la consulta *search_subject (´?p´, Automovil)*, se obtienen las filas 10-13, 16,18; de aquí se debe tomar los sujetos de esas filas. Sin embargo, no todos esos sujetos forman parte del resultado, sino sólo aquellos que satisfacen además el patrón *(´?s´, ´rdf:type´, ´oidImage´)* de manera de

garantizar que sean *OIDs* de registros imágenes. Para esto se obtiene la intersección de los sujetos que cumplen el patrón de la consulta y los sujetos que satisfacen el patrón *(´?s´, ´rdf:type´, ´oidImage´)*.

Ante una consulta del tipo *search_subject(P, ´?o´)*, se utiliza en mapa de bits sobre la columna *propiedad*. Por ejemplo, la consulta *search_subject(´Rdf:type´, ´?o´)*, se obtienen las filas 1, 2, 16-23. De aquí se debe tomar los sujetos de esas filas y se realiza la intersección con los sujetos que satisfacen el patrón *(´?s´, ´rdf:type´, ´oidImage´)*.

Ante una consulta del tipo *search_subject (P, O)*, se utiliza la intersección sobre los mapas de bits sobre las la columnas *propiedad* y *objeto*. Por ejemplo*: search_subject (´Rdf:type´, Automovil)*, con el mapa de bits sobre *propiedad* se obtiene las filas 1, 2, 16-23 y con el mapa de bits sobre *objeto* las filas 10-13, 16,18. La intersección entre estos nos dará las filas 16 y 18; se toman los sujetos de esas filas y se realiza la intersección con aquellos que satisfacen el patrón *(´?s´, ´rdf:type´, ´oidImage´)*.

4.3 Análisis de eficiencia

Una de las principales ventajas de la utilización de mapas de bits, es que son muy pequeños comparados con el tamaño real de la tabla que indexa. Para nuestro caso el tamaño de registro de una tripleta, suponiendo que se reserva 4000 bytes para *sujeto* y *predicado* y 10000 bytes para el *objeto*, entonces el espacio ocupado por un millón de tripletas es aproximadamente 16.8 GB. Sin embargo, un mapa de bits, (por ejemplo para Propiedad) con 80 valores de clave diferentes para un millón de tripletas ocuparía aproximadamente unos 9.5 MB. Esto significa, que aún para una gran cantidad de registros, este índice es muy pequeño y podría cargarse completamente en memoria, acelerando considerablemente el acceso a los registros.

Por otro lado, se puede acelerar considerablemente el cálculo de la intersección usando las instrucciones de bits ***And*** soportadas por la mayoría de los conjuntos de instrucciones de las computadoras. Una única instrucción bit a bit *and* puede calcular la intersección de 32 o 64 bits a la vez (palabra de 32 o 64 bits).

Uno de los factores más importantes que puede degradar el rendimiento en el uso de mapas de bits, es la cardinalidad de las columnas a indexar. Por esto se realizan las siguientes consideraciones.

Si las imágenes pertenecen a un dominio en particular (por ejemplo imágenes de Rodados), cada una de ellas puede ser asociada a una clase o concepto particular de ese dominio como sigue:

<oid rdf:type clase>
<oid rdf:type oidImage>

Esto significa que los objetos, para nuestro ejemplo, son en su mayoría las clases (vehículo, camión, automóvil, etc.), y se asume que son relativamente pocos los valores de las claves a indexar.

Además, una cantidad de relativamente reducida de predicados es lo típico en los datos RDF [9].

En resumen, si la cardinalidad de los objetos y predicados a indexar en relativamente bajo (en el orden de 100 valores), el tamaño del índice se mantendrá en el orden de los 10MB por cada millón de tripletas. Esto puede mejorarse por un método de compresión como "*gap compression Scheme*" [9], que consiste en indicar con qué bit comienza y luego sólo las cantidades de *0s* o *1s* contiguos. Por ejemplo, el mapa de bits para rdf:type en la Fig. 2, se reduce a: [1] 2 13 8. Este esquema tiene como ventaja que no es necesario la descompresión para realizar operaciones "And" bits a bits.

Para verificar la mejora en la eficiencia, se realizaron pruebas con diferentes cantidades de tripletas (en el orden de 10.000, 100.000 y 1.000.000 de tripletas). Se comparó el tiempo medio recuperación (TMR) para consultas por cada tipo de patrón. Los resultados que se presentan en la Fig. 3, muestra que el TMR mapas de bits no disminuye significativamente cuando el número de tripletas es bajo. Sin embargo, cuando la cantidad de tripletas es del orden de 100.000 a 1.000.000, el TMR es aproximadamente del 50%.

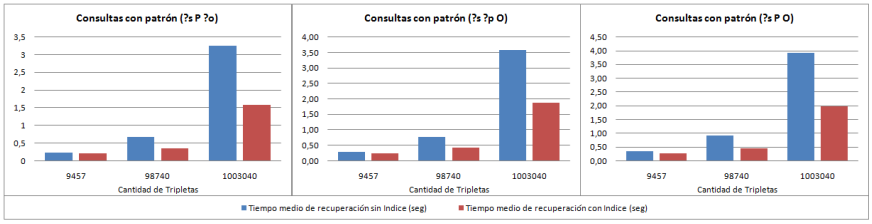

Fig. 3. Comparación del tiempo medio de recuperación sin el índice de mapa de bits y con el índice de mapa de bits, para los patrones: (?s P ?o) (?s ?p O) (?s P O).

5 Conclusiones

En este trabajo se presenta una forma de indexación que permite recuperar de manera eficiente metadatos introducidos en tripletas (RDF) en tablas OR. Estas tripletas son utilizadas para relacionar imágenes con conceptos semánticos, esto se hace por medio de los *OIDs* a registros de imágenes, que se utilizan como URI en los sujetos de las tripletas. Esta estructura, en sí, permite mejorar la eficiencia en la recuperación de imágenes, dado que el conjunto de *OIDs* que se obtienen como resultados de las consultas permite acceder directamente a las imágenes de manera no ambigua, aún en ambientes distribuidos, o sea, independientemente de la BD o sitio donde se encuentre y sin necesidad de realizar *joins*. También, la arquitectura presentada evita la mayoría de los *joins*, realizando operaciones de conjuntos sobre *OIDs*.

La propuesta de indexación, tiene como objetivo mejorar el tiempo de recuperación de los *OIDs* almacenados en las tripletas. Se realizaron pruebas con diferentes cantidades y se comparó TMR para consultas por cada tipo de patrón de la arquitectura. Los resultados en pruebas empíricas, muestran que la TMR para volúmenes importantes de tripletas disminuye un 50%. Es importante destacar, que el

uso de esta arquitectura y el método de indexación están soportados por la mayoría de los SGBD existentes.

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. Akiyoshi Matono, Said Mirza Pahlevi and Isao Kojima. RDFCube: A P2P-based Three-dimensional Index for Structural Joins on Distributed Triple Stores. Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P 2007). Lecture Notes in Computer Science, 2007, Volume 4125/2007, pp. 323-330. Springer-Verlag Berlin Heidelberg (2007).
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).