ERAV: Almacenamiento de Relaciones con Atributos Diversos en Bases de Datos Relacionales

Adolfo Di Mare, Universidad de Costa Rica

adolfo.dimare@ecci.ucr.ac.cr

RESUMEN

La representación "ERAV" (Entidad Relación Atributo Valor) es un método simple para almacenar en una base de datos relacional los valores para entidades descritas por atributos diversos o distintos, como los atributos que se necesitan para describir entidades que no tienen una estructura rígida de tabla relacional. Esta solución permite manipular los datos almacenados usando SQL tradicional y también mantiene gran coherencia con soluciones XML.

Palabras clave: Relaciones con atributos diversos, diseño de bases de datos, representación de objetos, esquema de bases datos.

ABSTRACT

The "ERAV" (Entity Relation Atribute Value) representation is a simple method for storing in a relational database values for entities described by diverse or different attributes, as those attributes needed to describe entities that do not have the rigid structure of a relational table. This solution allows the manipulation stored data using traditional SQL and also maintains great coherence with XML solutions.

Keywords: Relations with divese attributes, data base design, object representation de objetos, data base schema.

1. INTRODUCCIÓN

Ya los sistemas relacionales de bases de datos son de uso común en muchas aplicaciones; por ejemplo, el SQLite desarrollado por D. Richard Hipp [Hipp-2000] permite utilizar un motor relacional como si fuera una subrutina en un programa C o C++. Desafortunadamente, las bases de datos relacionales son excesivamente estructuradas y por eso en muchos casos se utiliza XML para incorporar en los programas de aplicación datos etiquetados y organizados jerárquicamente. Aunque en muchos contextos esta solución es adecuada, con este trabajo mostramos que si se usa la representación "ERAV" (Entidad Relación Atributo Valor) se puede lograr representar la misma información en una base de datos relacional, para luego procesarla usando SQL directamente.

En este trabajo se muestra cómo utilizar un Sistema Relacional de Administración de Bases de Datos (RDBMS: Relational DataBase Management System) para almacenar valores y relaciones entre entidades que representan productos y servicios, con el fin de usarlos es contextos en que XML también es otra alternativa de solución. Esto no quiere decir que se busque suprimir XML, que es una herramienta muy útil en muchos contextos, sino más bien complementar su utilización al lograr almacenar el contenido XML para ser consultado con SQL.

```
automotor(toyota)           automotor(hyundai)
  |                           |
  +--- mecánico               +--- dueño
  |       |                   |
  |       +---revisión        +--- aceite
  |
  +--- dueño
```

```
automotor(    K=1110111 , ID=0000 , SUB=0000 ,"toyota", cantidad puertas , color )
  mecánico(   K=1110111 , ID=0101 , SUB=0000 , nombre , teléfono , dirección )
    revisión( K=1110111 , ID=0201 , SUB=0101 , fecha , monto )
    revisión( K=1110111 , ID=0202 , SUB=0101 , fecha , monto , ACEITE )
    revisión( K=1110111 , ID=0203 , SUB=0101 , fecha , monto )
  mecánico(   K=1110111 , ID=0102,  SUB=0000 , nombre , teléfono , dirección )
    revisión( K=1110111 , ID=0205 , SUB=0102 , fecha , monto )
  dueño(      K=1110111 , ID=0301 , SUB=0000 , fecha )
  dueño(      K=1110111 , ID=0302 , SUB=0000 , fecha )

automotor(    K=2220222 , ID=0000 , SUB=0000 , "hyundai" )
  dueño(      K=2220222 , ID=0101 , SUB=0000 , "Pedro Mecánico" , teléfono , dirección )
  aceite(     K=2220222 , ID=0201 , SUB=0000 , fecha="20051302" )
```

Figura 1

Como se muestra en la Figura 1, en algunas ocasiones los valores que es necesario almacenar para una entidad no son simples. En este ejemplo se usan varios registros para almacenar los datos de varios vehículos: para el "toyota" hay que almacenar las revisiones mecánicas del automotor, agrupadas de acuerdo a la persona que las realizó; también hay que registrar la lista de dueños, pero en este caso solo hace falta conocer la fecha en que cada nuevo dueño adquirió el vehículo. Para el "hyundai" hay que registrar la lista que contiene todos los cambios aceite junto con la lista de los dueños anteriores. Para cada una de estas instancias, la jerarquía de valores a almacenar es distinta: para el "hyundai" hay un registro padre con 2 registros hijos, mientras que para el "toyota" la jerarquía es raíz - hijo - nieto. Lo más natural es registrar todos estos detalles usando un identificador único para cada automotor: en este caso se usa el número "1110111" para el "toyota" y "2220222" para el "hyundai". En algunos casos se ha usado el nombre del atributo en lugar de su valor específico para hacer el ejemplo más legible.

La llave de cada registro se ha completado usando 2 identificadores adicionales, "ID" y "SUB" que indican, dentro de cada automotor, la relación jerárquica: la raíz tiene su marcador "ID" en cero "0000" y el campo "SUB" indica cuál es el padre inmediato de cada registro; por supuesto, dentro de una misma entidad ("1110111" y "2220222" en este caso), todos los valores de "ID" deben ser diferentes para cada registro.

También la Figura 1 es muestra de que los valores almacenados pueden ser arbitrariamente diversos. Por ejemplo, para la "revisión" del vehículo "toyota" marcada con el identificador "ID=0202" existe un valor para el atributo "aceite", mientras que para las otras 2 revisiones, de identificador "0201" y "0203", ese atributo simplemente no existe (cuarto renglón). Si se modificaran estos esquemas para que cumplan con la primera forma normal de las bases de datos relacionales, también cumplirían con la forma normal de Boyce-Codd debido a que la llave de todas las relaciones sería "[K+ID]" y la dependencia funcional "[K+ID]→SUB" no sería una dependencia funcional parcial.

2. LA ALTERNATIVA XML PARA REPRESENTAR INFORMACIÓN

Es innegable que la información jerárquica puede representarse con gran sencillez usando XML para el que ya existe un lenguaje de consulta bien estructurado [XQuery-2007]. Se puede almacenar un documento XML como un gran BLOB (Binary Large OBject) en la base de datos, para luego hurgar dentro de él con Xquery.

```
<entity ID_ENT="2220222">
    <record REL_TYPE="e" ATTRIB="automotor" VAL="">
        <attrib REL_TYPE="s" ATTRIB="marca" VAL="hyundai" />

        <record REL_TYPE="e" ATTRIB="dueño" VAL="">
            <attrib REL_TYPE="s" ATTRIB="nombre" VAL="Pedro Mecánico" />
            <attrib REL_TYPE="i" ATTRIB="phone" VAL="77538888" />
            <attrib REL_TYPE="s" ATTRIB="dirección" VAL="3 kms al este del Guayabero" />
        </record>

        <record REL_TYPE="e" ATTRIB="aceite" VAL="">
            <attrib REL_TYPE="d" ATTRIB="fecha" VAL="20051302" />
        </record>
    </record>
</entity>
```

Figura 2

Hay muchas formas de representar en XML los datos de la Figura 1 pero el formato usado en la Figura 2 tiene la ventaja de que sigue la estructura jerárquica que tienen los registros. Para el "hyundai" el valor del campo "REL_TYPE" indica el tipo del dato: "e" para entidad (o registro), "s" para "string", "i" para "int", etc. La etiqueta "entity" contiene la identificación única de la entidad (producto o servicio). Cada renglón marcado "ATTRIB" es un atributo cuyo valor aparece en el parámetro "VAL": como la marca del auto identificado con el número "2220222" es "hyundai", hay un renglón de atributo que incluye ese valor específico (se usa el inglés para nombrar los identificadores para facilitar la reutilización de los módulos en otras aplicaciones).

El anidamiento de los registros de la Figura 1 se logra poniendo renglones dentro de un bloque etiquetado "record", que puede contener tantos atributos como sea necesario para describir a la entidad: esa es la flexibilidad inherente de XML.

En un ambiente en que el poder de cómputo es amplio, como ocurre con los programas que corren en potentes servidores de aplicación empresariales o institucionales, tiene mucho sentido usar XML para representar este tipo de información, pues la mayor parte de los sistemas administradores de bases de datos contemporáneos brindan apoyo XML adecuado, más si se toma en cuenta que es posible utilizar expresiones "FLWOR" (For Let Where Order by Return) que extienden la funcionalidad de las expresiones XPath para ponerlas a la par de la funcionalidad SQL de las bases de datos relacionales [Kay-2006].

Hay otros escenarios en los que XML no está disponible. Por ejemplo, si se construyen programas para procesadores embebidos o empotrados, puede que ocurrir que no esté disponible procesador XML adecuado. Por ejemplo, si se escoge para la implementación el motor de bases de datos SQLite, es posible que sea más sencillo utilizar una base de datos y manejarla directamente con SQL sin utilizar las avanzadas herramientas de XML.

El "ERAV" es uno de los módulos que serán usados en la implementación de un gestor de consultas sobre entidades, gestor que funciona en un ambiente distribuido de bases de datos en donde la interconexión entre los nodos del sistema se hace usando protocolos P2P (Pier to Pier) [AS-2004], entre computadores desperdigados en toda la internet. Este tipo de aplicación requiere de una gran inter-operatividad entre bases de datos, lo que se logra usando la versión común más simple de SQL sin las alteraciones o mejoras ofrecidas por cada implementación.

```
<entity ID_ENT="2220222">
    <record REL_ID="0000" REL_SUB="0000" REL_TYPE="e" ATTRIB="automotor" VAL="">
        <attrib REL_TYPE="s" ATTRIB="marca" VAL="hyundai" />

        <record REL_ID="0101" REL_SUB="0000" REL_TYPE="e" ATTRIB="dueño" VAL="">
            <attrib REL_TYPE="s" ATTRIB="nombre" VAL="Pedro Mecánico" />
            <attrib REL_TYPE="i" ATTRIB="phone" VAL="77538888" />
            <attrib REL_TYPE="s" ATTRIB="dirección" VAL="3 kms al este del Guayabero" />
        </record>

        <record REL_ID="0201" REL_SUB="0000" REL_TYPE="e" ATTRIB="aceite" VAL="">
            <attrib REL_TYPE="d" ATTRIB="fecha" VAL="20051302" />
        </record>
    </record>
</entity>
```

Figura 3

Se puede partir del documento XML de la Figura 2 para obtener una base de datos que contenga la misma información. Lo primero que hay que hacer es agregar los campos "ID" y "SUB" que sirven para indicar la relación jerárquica entre registros, como se muestra en la Figura 3 en las que se han agregado los atributos "REL_ID" y "REL_SUB" para este efecto.

El esquema de la base de datos en la que se pueden almacenar el documento ahora es claro: debe incluir una relación llamada "ERAV" que tenga los campos { Entidad=ID_ENT , Relación=REL(ID+SUB+TYPE] , Atributo=ATTRIB , Valor=VAL }. Estos datos se pueden obtener directamente de los valores que aparecen en la Figura 3.

3. JERARQUÍA ENTIDAD RELACIÓN ATRIBUTO VALOR

El problema que la representación "ERAV" (Entidad Relación Atributo Valor) resuelve es el de almacenar los valores y atributos para productos y servicios, que se representan como una entidad que tiene atributos cuya descripción requiere de datos simples o de varios datos ligados en una jerarquía cuya raíz es la entidad descrita. Por ejemplo, para la entidad "automotor" la raíz de la jerarquía contendrá su color, año de construcción y kilometraje. En registros hijos estarán varias fotos del automotor o sus cambios de aceite. En algunos casos una jerarquía de 2 niveles es suficiente para almacenar todos los datos de la entidad, pero en otros es necesario usar más niveles. La ventaja de la representación "ERAV" es que permite que instancias distintas de la misma entidad pueden tener almacenados atributos diferentes, con lo que se levanta el requerimiento relacional de uniformidad en las columnas de las relaciones. Por eso, puede ocurrir que para el auto "toyota" si esté almacenado su número de motor, mientras que el "hyundai" no lo tiene aunque sí esté registrado que sí tiene focos de tipo "alógeno". Para lograr este grado de libertad en el "ERAV" hay que mantener la indicación de cuáles atributos o cualidades están registradas para cada instancia.

WORD

ID_WORD	LANG	DESCR
0001111	es	automotor
0001234	en	number doors
0001234	es	cantidad puertas
1110111	es	color
1110123	es	azul
2220000	en	brand
2220000	es	marca
2220011	en	phone
2220021	es	mecánico

WORD (continuación)

ID_WORD	LANG	DESCR
2220022	es	revisión
2220023	es	dueño
2220024	es	aceite
5000001	es	alógeno
5000002	es	nombre
5000003	es	fecha
5000004	es	monto
5000005	es	dirección
6660011	es	foto

STRING

ID_STRING	LANG	DESCR
7700001	**	hyundai
7700002	**	toyota
7700021	**	Juan Piña
7700022	es	200 Sur del Palo de Guayaba
7700022	en	200 South of the Guayaba tree
7700024	**	Pedro Mecánico
7700025	es	3 kms al este del Guayabero
7700025	en	2 miles east from Guayabero

Figura 4

Para obtener la base de datos que corresponde al documento XML de la Figura 2 es deseable codificar los nombres de los atributos y de las hilera de la base de datos. Usar códigos para los nombres de los atributos permite internacionalizar la base de datos pues se puede usar varios lenguajes diferentes, como por ejemplo español e inglés, tanto para las consultas SQL en que se mencionen los atributos como para los valores de datos almacenados. Por ejemplo, el código "2220011" corresponde a la palabra en inglés "phone" en la tabla "WORD" pero no está en español; "2220022" corresponde a la "revisión" que está en español pero no está en inglés y la cantidad de puertas código "0001234" sí está en ambos lenguajes. En la Figura 4 se muestra la relación "WORD" que contiene las palabras que se usan para describir los productos o servicios almacenados en la base de datos. Una palabra que aparece en la tabla "WORD" puede usarse para describir tanto una entidad como cualquiera de sus atributos o cualidades. Aunque hay cierto orden en este ejemplo, en la práctica los códigos "ID_WORD" son números diferentes y no es necesario que sean consecutivos o que estén agrupados de acuerdo a un patrón.

La tabla "STRING" tiene una estructura similar a la de "WORD", pues ambas incluyen la columna "[LANG]" en la que se indica el idioma en que está escritra la palabra o la hilera; por eso las llaves para estas tabla son [ID_STRING+LANG] y [ID_WORD+LANG] respectivamente. En el caso de que una hilera sea al misma en cualquier idioma, se puede usar "**" como valor en la columna "[LANG]"; para el español se usa la hilera "es" y para el inglés "en".

```
<entity ID_ENT="2220222">
    <record REL_ID="0000" REL_SUB="0000" REL_TYPE="e" ATTRIB="0001111" VAL="">
        <attrib REL_TYPE="s" ATTRIB="2220000" VAL="7700001" />

        <record REL_ID="0101" REL_SUB="0000" REL_TYPE="e" ATTRIB="2220023" VAL="">
            <attrib REL_TYPE="s" ATTRIB="5000002" VAL="7700024" />
            <attrib REL_TYPE="i" ATTRIB="2220011" VAL="77538888" />
            <attrib REL_TYPE="s" ATTRIB="5000005" VAL="7700025" />
        </record>

        <record REL_ID="0201" REL_SUB="0000" REL_TYPE="e" ATTRIB="2220024" VAL="">
            <attrib REL_TYPE="d" ATTRIB="5000003" VAL="20051302" />
        </record>
    </record>
</entity>
```

Figura 5

Al usar los códigos de las tablas "WORD" y "STRING" en el documento XML de la Figura 2 se obtiene el documento XML de la Figura 5.

```
  ERAV
+---------+---------------+---------+----------+
| ID_ENT  |      REL      | ATTRIB  |   VAL    |
+---------+---------------+---------+----------+
| 1110111 | (0000)(0000):e | 0001111 |        0 |\
| 1110111 | (0000)(0000):s | 2220000 |  7700002 | | _ toyota
| 1110111 | (0000)(0000):i | 0001234 |        2 | |
| 1110111 | (0000)(0000):a | 1110111 |  1110123 |/
+---------+---------------+---------+----------+
| 2220222 | (0000)(0000):e | 0001111 |        0 |\  _ hyundai
| 2220222 | (0000)(0000):s | 2220000 |  7700001 |/
+---------+---------------+---------+----------+

  automotor                                          automotor
+---------+--------+------------------+-------+   +---------+---------+
| ID      | marca  | cantidad puertas | color |   | ID      | marca   |
+---------+--------+------------------+-------+   +---------+---------+
| 1110111 | toyota |                2 |  azul |   | 2220222 | hyundai |
+---------+--------+------------------+-------+   +---------+---------+
```

Figura 6

En la Figura 6 se muestra cómo usar el documento XML de la Figura 5 para poblar la tabla "ERAV" usando los códigos definidos en las tablas "WORD" y "STRING" de la Figura 4. En aquellos casos en que un valor no aplica, como ocurre para las entidades y registros cuyos tuples aparecen marcados con el atributo REL_TYPE="e", el valor del campo "VAL=0" se deja en cero, que resulta más cómodo que utilizar un valor nulo. Cada renglón del "ERAV" marcado con la etiqueta REL_TYPE="e" corresponde a un renglón en el documento XML marcado con la etiqueta <record.../> y para cada atributo del documento XML marcado con la etiqueta <attrib.../> hay un tuple con el valor de ese atributo. La columna "[ATTRIB]" del "ERAV" indica cuál es el nombre del atributo y siempre es una referencia a la tabla "WORD". En la columna "[VAL]" está la referencias a la tabla "STRING" que sirve para indicar en forma codificada cuál es el valor del atributo descrito, o también puede estar el valor del atributo si ese valor se puede expresar como un número, como ocurre con el atributo "cantidad de puertas" cuyo valor es "2" (código "0001234").

En este ejemplo, para el automotor "toyota" se han registrado 3 atributos: "marca" [código "2220000"], "cantidad puertas" [código "0001234"] y "color" [código"1110111"]. Para el "hyundai" no se registra más que la marca. Esta flexibilidad del "ERAV" es la que permite almacenar información diversa para la misma entidad. De todas formas, siempre es posible extraer de la base de datos los registros que corresponden a un tipo de entidad, pues es posible seleccionar aquellos registros que corresponden a una marca seleccionando los valores "ATTRIB=2220000" ["marca"] y "VAL=7700001" ["hyundai"].

Para cada atributo de la entidad "[automotor]" existe un renglón en el "ERAV". La llave que amarra todos los datos de una misma entidad es "[ID_ENT]", que contiene un código numérico único que identifica a la entidad. El

nombre de la entidad siempre tiene valor "[REL]" igual a cero; por eso, en la Figura 6 el primer registro para al entidad "1110111" ["toyota"] tiene el valor de "[REL]" y de "[VAL]" en cero y el campo "[ATTRIB]" indica que la entidad es "automotor" [código 0001111""]. El orden relativo de las columnas de los atributos no queda registrado en el "ERAV" pues no importa si el atributo "color" aparece antes o después del atributo "cantidad puertas".

ERAV

ID_ENT	REL	ATTRIB	VAL	
2220222	(0000)(0000):e	0001111	0	\ _ hyundai
2220222	(0000)(0000):s	2220000	7700001	/
2220222	(0101)(0000):e	2220023	0	--> dueño
2220222	(0101)(0000):s	5000002	7700024	nombre
2220222	(0101)(0000):i	2220011	77538888	teléfono
2220222	(0101)(0000):s	5000005	7700025	dirección

Figura 7

En la Figura 7 se muestran los tuples del "ERAV" que se obtienen a partir de los renglones del documento XML que están dentro del bloque <record> que contiene los datos del dueño del vehículo "hyundai" de la Figura 5 (o de la Figura 3). Como este bloque tiene un renglón con la etiqueta <record> y otros 3 renglones marcados con la etiqueta <attrib>, en total quedan 4 tuples en el "ERAV": cada tuple del "ERAV" corresponde a un único renglón del documento XML.

En el "ERAV" se pueden almacenar varios tipo de datos porque los valores almacenados pueden ser de muchos tipos diferentes: enteros, reales, hileras, objetos binarios BLOB (Binary Large OBject), fechas, montos, etc.:

e → (entity) → nombre de la entidad en "WORD"
a → (attribute) → valor del atributo en "WORD"

i → (int) → entero
f → (float) → punto flotante

t → (time) → tiempo
d → (date) → fecha
h → (hour) → hora
s → (string) → hilera en el "STRING"
b → (blob) → BLOB

Puede notarse en la Figura 4 y en la Figura 6 que se ha usado el mismo valor numérico "1110111" para el auto en la tabla "ERAV" y también para la palabra "color" "1110111" en la tabla "WORD". Esto no es problema pues siempre el contexto indica si el valor "1110111" representa la llave de la entidad o una palabra que se usa para describir atributos de entidades.

ERAV (campos y sub-campos)

ID_ENT	REL	ATTRIB	VAL
	(ID)(SUB):T(ype)		

Figura 8

En la Figura 8 están los 4 sub-campos del atributo "[REL]" del "ERAV". El valor almacenado en el sub-campo "TYPE" indica el tipo de datos almacenado en la columna "[VAL]" del "ERAV" ([entity↔blob]). El sub-campo "[ID_ENT]" amarra a todos los registros de una entidad, lo que permite que los valores almacenados en el sub-campo "[REL.ID]" no sean únicos para entidades diferentes (por eso se puede usar el mismo valor "0101" para el sub-campo "[REL.ID]" del "toyota" y del "hyundai"). La raíz de la jerarquía, que es el registro con los datos de la entidad "toyota", siempre tiene el valor del sub-campo "[REL.ID]=0000".

El sub-campo "[REL.SUB]" sirve para definir cuál de todos los sub-registros es el ancestro inmediato. Para el registro de "mecánico" el ancestro inmediato es "toyota", la raíz de la jerarquía, cuyo valor es "[REL.SUB]=0000". Cada sub-registro de "revisión" tiene por ancestro inmediato a un "mecánico", por lo que el valor del sub-campo "[REL.SUB]" para los sub-registros del primer "mecánico" es "[REL.SUB]=0101" y para los del segundo es "[REL.SUB]=0102". En el "ERAV" no queda registrado el orden relativo de un sub-registro hijo respecto a sus hermanos, pero es posible agregar el sub-campo "[REL.SEC]" para registrar ese dato.

ERAV (continuación)

ID_ENT	REL(ID)(SUB):T	ATTRIB	VAL	1110111 --> automotor
1110111	(0101)(0000):e	2220021		--> mecánico
1110111	(0101)(0000):s	5000002	7700021	nombre
1110111	(0101)(0000):i	2220011	88887753	teléfono
1110111	(0101)(0000):s	5000005	7700022	dirección
1110111	(0201)(0101):e	2220022		--> revisión
1110111	(0201)(0101):d	5000003	20080923	fecha
1110111	(0210)(0101):f	5000004	250.55	monto
1110111	(0202)(0101):e	2220022		--> revisión
1110111	(0202)(0101):d	5000003	20081025	fecha
1110111	(0202)(0101):d	2220024	386.12	aceite
1110111	(0202)(0101):f	5000004	256.55	monto
1110111	(0202)(0101):e	2220022		--> revisión
1110111	(0202)(0101):d	5000003	20081112	fecha
1110111	(0202)(0101):f	5000004	260.33	monto

Figura 9

Como se muestra en la Figura 9, para almacenar todos los datos de la jerarquía de valores es necesario identificar tanto a los registros como a los valores almacenados en cada registro. Todos estos valores tiene el mismo identificador "1110111" para "toyota". El campo "[VAL]" está vacío (nulo o en cero) para cada renglón del "ERAV" que describe una entidad o un sub-registro: estos son los renglones marcados con el descriptor de tipo "REL.TYPE='e'". Luego están los valores para ese registro y debe haber un renglón por cada atributo. Por ejemplo, si un sub-registro tiene 5 campos, en total se usarán 6 registros para almacenar sus datos en el "ERAV": 1 para describir la entidad o registro, y 5 renglones más, para almacenar el valor para cada uno de los 5 atributos. El valor almacenado en la columna "[ATTRIB]" siempre es una referencia a la tabla "WORD", que es en donde están los nombres de atributos y de relaciones. En la Figura 1 hay 9 sub-registros para el "toyota" por lo que es necesario que el "ERAV" contenga 9 sub-registros, identificados cada uno con un valor del sub-campo "[REL.ID]" diferente (0000 0101 0201 0202 0203 0102 0205 0301 0302).

El sub-campo "REL.TYPE" indica el tipo de dato almacenado en la columna "[VAL]". Cuanto el sub-campo "REL.TYPE='s'" el valor está almacenado en la tabla "STRING". Por ejemplo, en la Figura 9, el nombre y la dirección son hileras de longitud variable almacenadas en la tabla global de hileras "STRING" y el teléfono es un número entero (código de tipo 'i') que está almacenado directamente en el "ERAV" (también los valores de los atributos fecha y monto, código de tipo 'd' y 'f', están almacenados en el "ERAV"). En la Figura 6, el valor del color está tiene "REL.TYPE='a'" por lo que está codificado con el número "1110123", que es el código del color "azul" en la tabla "WORD".

Como "mecánico" es el primer sub-registro hijo de "toyota", su valor para el campo "[REL.ID]" es "0101" y el valor para su campo "[REL.SUB]" es "0000", pues la raíz de la jerarquía anidada es el registro "toyota" cuyo registro descriptivo tiene esos dos campos en cero ("[REL.ID]=0000" y "[REL.SUB]=0000"). El primer registro de "revisión" tiene identificación "[REL.ID]=0201" y es el primer hijo del sub-registro "mecánico" identificado con "[REL.ID]=0101", por lo que ese registro de "revisión" indica con su valor "[REL.SUB]=0101" que desciende directamente del sub-registro "mecánico" "0101". El campo "[REL.SUB]" indica cuál es el registro del que un renglón del "ERAV" desciende directamente; de esta manera es posible definir con exactitud la jerarquía de la relación anidada de la base de datos.

El truco usado pare almacenar los valores jerárquicos en la relación "ERAV" no es nuevo pues se encuentra en libros de texto básicos como [AHU-1984]: consiste en registrar para cada hijo quien es su padre, lo que se logra con el sub-campo"[REL.SUB]".

4. ALMACENAMIENTO EFECTIVO EN EL "ERAV"

En la Figura 6 junto con la Figura 9 se muestra una forma de almacenar los datos en el "ERAV", pero requiere que sistema administrador de la base de datos permita que el campo "[VAL]" mantenga valores de varios tipos diferentes; por ejemplo, el número entero "88887753" es un número teléfono mientras que el valor de punto flotante "250.55" que es un monto. Otra forma de manejar esta diversidad de tipos es almacenar los valores en una hilera en formato hexadecimal, usando una codificación como el formato BinHex usado en algunos computadores [Wiki-2004], en una tabla "ERAV_VAL" que incluya una columna que indique el tipo de dato. En este caso, sería necesario implementar varias funciones para transformar cada valor hexadecimal en el tipo adecuado, tomando en cuenta que en algunas máquinas los números se almacenan del byte más significativo al menos significativo, y en otras al revés (Big Endian vs Little Indian). También es factible almacenar los valores BLOB como una gran hilera en "ERAV_VAL". Además es posible fusionar en una sola las tablas "STRING" y "WORD", indicando si el valor almacenado pertenece a una u otra tabla con el signo del número de referencia.

Es posible ahorrar espacio al codificar los sub-campos de "[REL]" en un sólo número entero de 32 bits, usando una cantidad relativamente pequeña de bits para almacenar el valor de cada sub-campo:

14 bits → [REL.ID] → identificador de sub-registro
14 bits → [REL.SUB] → identificador del padre
4 bits → [REL.TYPE] → tipo de dato (entity↔blob) "[0..15]" → "(e a i f t d h s b)"

Aunque es posible almacenar en una sola relación el contenido de "STRING" y "WORD", resulta más simple mantenerlas separadas para facilitar al formulación de consultas SQL. La solución más sencilla para almacenar los valores del "ERAV" es confiar en que el sistema administrador de bases de datos usa eficientemente el espacio de almacenamiento disponible; no vale la pena manchar el diseño con micro eficiencias. Por eso, en la mayor parte de los caso lo que mejor conviene es que el "ERAV" incluya varias columnas, una para cada posible tipo de dato. Por ejemplo, si se quiere usar datos de tipo fecha y hora, en la tabla habrá una columna "[VAL_DATE]" para la fecha y otra "[VAL_HOUR]" para la hora. Solo una de las columnas "[VAL]" tendrá su valor no nulo

```
-- Entidad Relación Atributo Valor
CREATE TABLE [ERAV] (
    [ID_ENT]     INTEGER   NOT NULL, -- <K+++> <E> Entidad
--  [REL] NOT NULL,                              <R> Relación
      [REL_ID]   SMALLINT  NOT NULL, -- <+K++>
      [REL_SUB]  SMALLINT  NOT NULL, -- <++K+>
      [REL_TYPE] CHAR(01)  NOT NULL,
    [ATTRIB]     INTEGER   NOT NULL, -- <+++K> <A> Atributo
--  [VAL] NOT NULL, -- Solo uno no es NULL     <V> Valor
      [VAL_INT]    INTEGER,  -- entero o referencia a "WORD"
      [VAL_FLOAT]  FLOAT,    -- punto flotante

      [VAL_TIME]   TIMESTAMP, -- tiempo
      [VAL_DATE]   DATE,      -- fecha
      [VAL_HOUR]   TIME,      -- hora

      [VAL_BLOB]   BLOB,      -- valor binario enorme

    CONSTRAINT [ERAV_KEY] UNIQUE
      ( [ID_ENT],[REL_ID],[REL_SUB],[ATTRIB] )
);
```

Figura 10

En la Figura 10 están los campos que componen el "ERAV". El campo "[REL_TYPE]" indica el tipo de renglón o el tipo de datos del atributo. "[VAL_INT]" es un número entero que se puede usar para almacenar un valor entero o para indicar a cuál palabra en la tabla "WORD" se hace referencia. No existe el campo "[VAL_STRING]" porque todas las hilera se pueden almacenar en "STRING" y proveer 2 formas de hacer lo mismo complica las cosas. La llave para esta relación es la concatenación del los campos [ID_ENT] + [REL_ID] + [REL_SUB] + [ATTRIB].

El lector cuidadoso ya habrá descubierto que esta versión del "ERAV" no cumple con la forma normal de Boyce-Codd pues existe una dependencia funcional que no es llave de la relación: "[ID_ENT+REL_ID]→REL_SUB". Esto explica la duplicación de valores que se nota en la columna "[REL]" de la Figura 9. Este pequeño detalle se puede solucionar usando una tabla adicional que contenga esos 3 campos { ID_ENT REL_ID REL_SUB }, lo que permite remover del "ERAV" el atributo "REL.SUB". Esta normalización impediría empacar el campo "[REL]" en un sólo número entero de 32 bits y también se hace más complicada la explicación de cómo está constituido el "ERAV". Es fácil encontrar contextos en que lo mejor es remover esta dependencia funcional parcial, pero por lo menos para esta exposición queda bonito que todos los datos de un documento XML se pueden representar usando solo una relación en la base de datos relacional.

5. CONSULTAS SQL DEL "ERAV"

Como ya se comentó previamente, el "ERAV" permite realizar consultas SQL con gran flexibilidad, pues el resultado SQL siempre es un tabla rectangular tomada de los valores almacenados en el "ERAV". En algunas ocasiones es importante que la estructura jerárquica de los valores almacenados quede representada en el resultado de la consulta, pero en la mayor parte de las ocasiones estos no es lo necesario.

```
-- Todos los autos 'toyota' que tengan revisiones con un costo entre 250 y 260
SELECT *
FROM   ERAV
WHERE  ID_ENT IN (
    SELECT m.ID_ENT
    FROM   ERAV m
        -- 'automotor'
        INNER JOIN ERAV e1  ON m.ID_ENT  = e1.ID_ENT   AND e1.REL_TYPE = 'e'
        INNER JOIN WORD a1N ON e1.ATTRIB = a1N.ID_WORD AND a1N.DESCR = 'automotor'
        -- ATTRIB.'marca' = 'toyota'
        INNER JOIN ERAV a2  ON m.ID_ENT  = a2.ID_ENT   AND a2.REL_TYPE = 's'
        INNER JOIN WORD a2N ON a2.ATTRIB = a2N.ID_WORD AND a2N.DESCR = 'marca'
        INNER JOIN STRING a2V ON m.VAL_INT = a2V.ID_STRING AND a2V.DESCR = 'toyota'
        -- ATTRIB.'monto' >= 250
        INNER JOIN ERAV a3  ON m.ID_ENT  = a3.ID_ENT   AND a3.REL_TYPE = 'f'
        INNER JOIN WORD a3N ON a3.ATTRIB = a3N.ID_WORD AND a3N.DESCR = 'monto'
                                                       AND a3.VAL_FLOAT >= 250
        -- ATTRIB.'monto' <= 260
        INNER JOIN ERAV a4  ON m.ID_ENT  = a4.ID_ENT   AND a4.REL_TYPE = 'f'
        INNER JOIN WORD a4N ON a4.ATTRIB = a4N.ID_WORD AND a4N.DESCR = 'monto'
                                                       AND a4.VAL_FLOAT <= 260
);
```

Figura 11

En la Figura 11 está la consulta que obtiene todos los autos "toyota" que tengan revisiones con un costo entre 250 y 260. La estrategia para hacer una consulta consiste en pegarle al registro principal "m" del "ERAV" cada uno de los atributos por los que se quiere calificar los valores seleccionados. En el caso de esta consulta, hay 4 partes bien definidas en la consulta, cada una de las que sirve para agregar mediante JOIN cada uno de los atributos que se necesitan para calificar la entidad: "automotor", "toyota", "monto>=250" y "monto<=260" (en este caso, no se ha requerido que el sea el mismo mecánico quien hace las revisiones). Si el "ERAV" solo tuviera datos sobre automotores, no sería necesario calificar los registros para que correspondan únicamente a la entidad "automotor" pero en el caso general contendrá datos sobre muchos objetos diversos. Se puede formular la consulta en cualquiera de los lenguajes cuyas nombres de atributo estén en la tabla "WORD"; es posible mezclar lenguajes, porque con el JOIN de SQL se cubren todas las posibilidades.

En aquellos casos en que es necesario reflejar la jerarquía de los datos, quien realiza la consulta debe incluir el camino en los registros y sub-registros que desea recorrer. En aquellos casos en que una jerarquía está representada de manera diferente para entidades diferentes, ocurrirá que el resultado de la consulta incluirá solo los valores de la jerarquía que fue descrita al realizar la consulta. En los ejemplos aquí expuestos se han usado identificadores "[REL_ID]" con valores no consecutivos, pero en la práctica puede resultar más sencillo usar valores en secuencia.

Se puede argumentar que el "ERAV" es un ejemplo de relaciones anidadas [Colomb-2008] y de ahí concluir que las consultas SQL serían muy costosas pues para resolverlas se requieren muchas operaciones JOIN. Debido a que

el identificador de cada entidad "ID_ENT" amarra todos los tuples que corresponden a una misma entidad, posiblemente un buen optimizador de consutlas SQL pueda mitigar este problema. Algunos programadores que prefieren no usar XML les gusta decir que las consultas SQL son resueltas con mucho más eficiencia que las consultas XQuery [DTCO-2003].

El manejo de los campos nulos representa otro desafío para el "ERAV", pues requiere de un esfuerzo adicional al hacer la consulta, esfuerzo que se deriva de la forma en que SQL maneja los atributos nulos. Lo natural al representar información jerárquica es que muchos atributos simplemente no existan. El estándar SQL incluye el atributo NULL para manejar este caso, pese a que algunos autores han mostrado las ventajas de usar más de un valor nulo, para distinguir estos 2 casos: "No se Aplica" y "Desconocido" [Codd-1987]. Al representar en el "ERAV" los datos, son las reglas del SQL son las que se usan y por eso esa es la forma en que se manejan los valores nulos. Debido a que estas jerarquías generalmente se usan para almacenar datos descriptivos de una entidad, esta forma de manejo de los nulos es apropiada en casi todos los casos, pues nunca ocurre que un campo llave es nulo, como se ve en la Figura 10.

Es popular la idea de almacenar en la base de datos algunos valores que permiten parametrizar las aplicaciones. Por ejemplo, si el impuesto de ventas o impuesto al valor agregado se almacena en la base de datos, cuando cambia no hace falta recompilar programas pues todos lo obtienen de la base de datos. Este tipo de valor se puede almacenar en el "ERAV" de manera natural.

6. BIBLIOTECA ERAV2XML()

Una forma de utilizar el "ERAV" que puede resultar ventajosa es contar con 2 verbos, xml2erav() y erav2xml() que sirvan para transformar datos XML a formato tabular y viceversa. Una implementación C++ de estas rutinas está disponible en la red:

http://www.di-mare.com/adolfo/p/ERAV/ERAV.zip
http://www.di-mare.com/adolfo/p/ERAV/index.htm

Esta implementación incluye un analizador léxico y un analizador sintáctico para transformar un documento XML en una lista de tuples. Se ha usado la biblioteca estándar de C++ para almacenar en memoria el contenido completo del documento "ERAV" usando hileras std::string<> y listas std::list<>. Los diccionarios para codificar los nombres y las hileras se han implementado usando el std::map<> de la STL (Standard Template Library).

```
<entity ID_ENT="2220222">
    <record ATTRIB="1110111">
        <attrib REL_TYPE="s" ATTRIB="2220000" VAL="7700001" />

        <record ATTRIB="2220023">
            <attrib REL_TYPE="s" ATTRIB="5000002" VAL="7700024" />
            <attrib REL_TYPE="i" ATTRIB="2220011" VAL="77538888" />
            <attrib REL_TYPE="s" ATTRIB="5000005" VAL="7700025" />
        </record>

        <record ATTRIB="2220024">
            <attrib REL_TYPE="d" ATTRIB="5000003" VAL="20051302" />
        </record>
    </record>
</entity>
```

Figura 12

La rutina xml2erav() permite manipular documentos XML como el que se muestra en la Figura 12, que fue obtenido eliminando los campos "REL_ID" y "REL_SUB" del documento XML de la Figura 5, pues su valor se puede deducir por el contexto de anidamiento de los bloques <record> pues la estructura jerárquica del documento XML indica claramente la dependencia entre sub-registros. También puede omitirse el indicador REL_TYPE="e" para cada descriptor de sub-registro junto con el valor nulo VAL="". Por eso no es ambiguo re-escribir algunos de los renglones del documento XML como se muestra en la Figura 12.

La rutina erav2xml() permite transformar los valores tabulares "ERAV" almacenados en una lista C++ en un documento XML como el que se muestra en la Figura 12. La pareja erav2xml() y xml2erav() no pueden manejar documentos XML arbitrarios, pues en la implementación se ha asumido que los valores de los datos siempre están

codificados en como parámetros "ATTRIB" y "VAL", aunque no debiera ser muy complicado incorporarles la posibilidad de procesar el texto CDATA de un documento XML arbitrario. El analizador sintáctico que usado en la implementación de xml2erav() es relativamente simple, pues la gramática de XML es LL(1) lo que facilita escribir manualmente un reconocedor para ese lenguaje de marcas.

7. CONCLUSIÓN

Es posible implementar programas relativamente simples para trasladar los valores almacenados en la base de datos a un documento XML, o viceversa, utilizando, por ejemplo, el módulo de código abierto "expat", descrito en [Clark-2000] o el "Markup" de [Bryant-2003]. Sin embargo, en muchas ocasiones no se necesita tanta maquinaria para procesar un solo documento, sino que más bien se quiere usar todos los documentos XML como si fueran parte intrínseca de la base de datos [PVVGR-2004]. Para estos casos hay que escoger alguno de los procesadores XML para bases de datos a nivel comercial [Dayen-2001].

Aunque aumente significativamente la popularidad de XML en bases de datos, lograr realizar el mismo trabajo con menos recursos siempre es conveniente. Es más difícil hacer consultas en muchos documentos XML de estructura significativamente diferente, que son más complejos, porque están estructurados en una jerarquía XML viva y expresiva. Además, como cada sistema de gestión de bases de datos interpreta el mundo XML de una forma diferente, es relativamente difícil lograr que la aplicación no quede amarrada a un motor de bases de datos específico.

La ventaja principal de utilizar el "ERAV" para almacenar los valores de una jerarquía de entidades es que no se pierda la capacidad de usar SQL directamente para obtener información de la base de datos. Esta solución es adecuada para algunas aplicaciones que tienen estos requerimientos:

- Los datos a almacenar pueden ser representados con una jerarquía Entidad - Relación - Atributo - Valor.
- Las entidades cuyos datos hay que almacenar son estructuralmente equivalentes, pero difieren en cuanto a los atributos específicos que se quiere registrar.
- Se quiere mantener acceso SQL a los datos para reducir el costo de la aplicación, tanto al construirla como al darle mantenimiento.

Definitivamente el uso del "ERAV" no convierte en innecesaria la tecnología XML, pese a que es relativamente sencillos almacenar un documento XML en un repositorio de bases de datos "ERAV", cuya mayor cualidad es la facilidad con que se puede manipular mediante SQL.

Si la diversidad de atributos está localizada a unas cuantas entidades, puede ser preferible usar una solución "ERAV" a un híbrido SQL+XML; en especial, cuando los recursos computacionales no son muy abundantes como ocurre con sistemas embebidos o empotrados, puede resultar muy atractivo prescindir del XML sin limitar la funcionalidad de la aplicación: el "ERAV" es una solución tanto simple como portátil a muchas plataformas. Si se codifican los los atributos, las consultas SQL se pueden realizar en varios lenguajes (español, inglés, etc.).

Además, si se usa la representación de información "ERAV" basta una consulta SQL para trasladar los datos de las bases de datos operacionales a las bases de datos dimensionales, pues todos los datos están almacenados en formato relacional, lo que facilita alimentar las tablas de dimensiones y hechos que forman un sistema OLAP (Online Analytical Processing), que es una de las herramientas que permiten dar apoyo a los sistemas de gestión para la alta gerencia.

8. AGRADECIMIENTOS

Alejandro Di Mare aportó varias observaciones y sugerencias importantes para mejorar este trabajo, como también lo hizo Ronald Argüello.

REFERENCIAS

[AHU-1984] Aho, Alfred V. & Hopcroft, John E. & Ullman, Jefrrey D.: Data Structures and Algorithms, Addisson-Wesley Publishing Co., 1984.

[AS-2004] Androutsellis-Theotokis, Stephanos & Spinellis, Diomidis: A Survey of Peer-to-Peer Content Distribution Technologies, ACM Computing Surveys, 36(4):335-371, diciembre 2004. **http://www.spinellis.gr/pubs/jrnl/2004-ACMCS-p2p/html/AS04.html**

[Bryant-2003] Bryant, Ben: XML class for processing and building simple XML documents, 2003. **http://www.codeproject.com/KB/cpp/markupclass.aspx**

[Clark-2000] Clark, James: XML class for processing and building simple XML documents, Thai Open Source Software Center Ltd., 2000. **http://www.jclark.com/xml/expat.html**

[Codd-1987] Codd, E.F.: More commentary on missing information in relational databases (applicable and inapplicable information), ACM SIGMOD Record, Vol.16 No.1, 1987. **http://portal.acm.org/citation.cfm?id=24823**

[Colomb-2008] Colomb, Robert M.: The Nested Relational Data Model is Not a Good Idea, School of Information Technology and Electrical Engineering, The University of Queensland, Australia, 2008. **http://www.itee.uq.edu.au/~infs3200/_Readings/NRBadIdea.pdf**

[Dayen-2001] Dayen, Igor: Storing XML in Relational Databases, O'Reilly XML Resources, junio 2001. **http://www.xml.com/pub/a/2001/06/20/databases.html**

[DTCO-2003] DeHaan, David & Toman, David & Consens, Mariano P. & Ozsu, M. Tamer: A comprehensive XQuery to SQL translation using dynamic interval encoding, ACM SigMod 2003. **http://www.cs.uwaterloo.ca/~david/papers-sigmod03.pdf**

[Hipp-2000] Hipp, D. Richard: SQLite, Hwaci - Applied Software Research, 2000. http://www.SQLite.org

[Kay-2006] Kay, Michael: Blooming FLWOR - An Introduction to the XQuery FLWOR Expression, XQuery Tutorial, 2006. **http://www.xquery.com/tutorials/flwor/**

[PVVGR-2004] Pal, Shankar & Parikh, Vishesh & Zolotov, Vasili & Giakoumakis, Leo & Rys, Michael: XML Best Practices for Microsoft SQL Server 2005, Microsoft Corporation, abril 2004. **http://technet.microsoft.com/en-us/library/ms345115.aspx**

[Wiki-2004] BinHex, From Wikipedia, the free encyclopedia 2004. **http://en.wikipedia.org/wiki/Binhex**

[XQuery-2007] Clark, James: W3C XML Query (XQuery), Specification by 3 W3C Working Groups: XML Query & Schema & XSL Working Groups, enero 2007. **http://www.w3.org/XML/Query/**

Authorization and Disclaimer

Authors authorize LACCEI to publish the paper in the conference proceedings. Neither LACCEI nor the editors are responsible either for the content or for the implications of what is expressed in the paper.