1º CONTECSI Congreso Internacional de Gestión de la Tecnologia y Sistemas de Información
21-23 de Junio de 2004 USP/São Paulo/SP- Brasil

Una alternativa para consultas recursivas en SQL

Francisco Javier Moreno A.[1]
fjmoreno@unalmed.edu.co

Universidad Nacional de Colombia, Sede Medellín.
Dirección Postal: Facultad de Minas, Bloque M8 Of. 209. Medellín, Colombia.
Teléfono: 4255352

Palabras Claves: Consultas recursivas, árboles, SQL-99, consultas jerárquicas, recursión
Area: Bases de Datos.

Resumen

En este artículo se presenta una alternativa para la solución de consultas recursivas en SQL puro sin acudir al uso operadores especializados ni a lenguaje procedimental; esta es una alternativa que puede ser contemplada tanto desde puntos de vista de optimización como de Sistemas de Gestión de Bases de Datos (SGBD) que no posean operadores especializados para consultas recursivas.

1. Introducción

Desde su surgimiento a mediados de la década 1970 el lenguaje de consulta para bases de datos SQL evolucionó hasta convertirse en un estándar adoptado por el ANSI en 1986 [Web Site 2]. Revisiones posteriores (1989 y 1992) enriquecieron su poder expresivo y su última versión (1999) ha adicionado una serie de características objeto-relacionales (tipos de datos definidos por el usuario, métodos, herencia etc.) dando como resultado un lenguaje

[1] Profesor Departamento de Ingeniería de Sistemas e Informática, Universidad Nacional de Colombia Sede Medellín.

híbrido entre los de bases de datos y los lenguajes convencionales de programación orientados a objetos (C++, Java, etc.).

Desde 1996 con la inclusión al SQL de la opción procedimental (PSM, Persistent Stored Modules) [Melton 98] el lenguaje se vuelve computacionalmente completo (gracias a las estructuras clásicas de programación: secuencia, decisión e iteración). Se puede distinguir entonces entre el SQL "puro" y el SQL + PSM. El PSM se ha incorporado en Sistemas de Gestión de Bases de Datos como Oracle (PL/SQL) y SQL Server (TSQL) entre otros.

Sin embargo en muchos casos sólo se puede hacer uso de SQL puro por diversas razones (aspectos de optimización y SGBDs que no permiten desarrollar procedimientos almacenados como MYSQL; entre otras)

Cuando esto sucede es posible encontrarse con consultas de difícil solución utilizando sólo SQL puro, entre ellas destacan las de tipo recursivo, las cuales son presentadas en la sección 3. Varios SGBDs han dado solución a este problema introduciendo operadores para tal propósito -en Oracle, *connect by* [Oracle 02] y en DB2 *with recursive* [Web Site 3]. Atendiendo a este problema, el SQL-99 [Gulutzan 99] también ha adicionado un operador [Ullman 01] para tal propósito con el mismo nombre que su contraparte de DB2, aunque la versión de este operador en DB2 no posee todas las características que establece el estándar [Web Site 3].

En este artículo se presenta una variante para la solución de consultas recursivas en SQL puro sin acudir al uso operadores especializados ni a lenguaje procedimental; esta es una alternativa que puede ser contemplada tanto desde puntos de vista de optimización como de SGBDs que no posean operadores especializados para consultas recursivas.

En el resto del artículo siempre que se hable de SQL se refiere a SQL puro a menos que se exprese lo contrario.

El artículo se desarrolla así: en la sección 2 se presenta un caso de estudio, en la sección 3 se explica y ejemplifica el método propuesto para solucionar consultas recursivas, en la sección 4 se exponen algunas limitaciones del método y posibles formas de evitarlas y en la sección 5 se presentan algunas conclusiones y posibles trabajos futuros.

2. Caso de Estudio

Las consultas recursivas pueden surgir en muchas situaciones:

- Árboles genealógicos
- Descomposición estructural de una compañía en departamentos y estos a su vez en sub-departamentos etc.
- Listas de discusión (mensaje, respuesta, contra respuesta, etc.)
- Composición de productos basados en otros productos (el famoso problema de la explosión de partes [Date00]).

De hecho tal y como se expresa en [Date 00] una solución impráctica a este tipo de consultas es realizar una serie de reuniones de una tabla consigo misma *n* veces, donde *n* indica el nivel de descomposición o profundidad deseado, es decir, si se desea, por ejemplo, en el caso de un árbol genealógico determinar los bisnietos de un individuo, se debe realizar una triple reunión de la tabla consigo misma. Aparte de impráctico, porque el máximo nivel de descomposición se desconoce normalmente, el costo computacional de una *n auto reunión* es enorme.
Un ejemplo de este tipo de consulta se muestra en la sección 3.3.

Se plantea a continuación un método alternativo, para ello considérese la siguiente situación:
Supongamos una compañía se especializa en la venta de un producto determinado[2]. Cada vendedor puede afiliar a muchos otros vendedores y a su vez cada uno de éstos puede

[2] La situación puede ser generalizada para manejar múltiples productos pero no es relevante para el problema que se expone.

afiliar a otros. Un vendedor no tiene necesariamente que haber sido afiliado por otro vendedor pero en caso de que lo sea sólo puede ser afiliado por uno y sólo uno.

A cada vendedor se le lleva el conteo de cuantos productos (unidades) vende durante el mes. Aquellos vendedores que directamente o indirectamente (es decir a través de otros vendedores que ellos hayan afiliado) hayan afiliado a un mayor número de vendedores podrían tener una bonificación de acuerdo a las políticas de la compañía. En la figura 1 se muestra un modelo entidad – relación que representa la situación descrita:

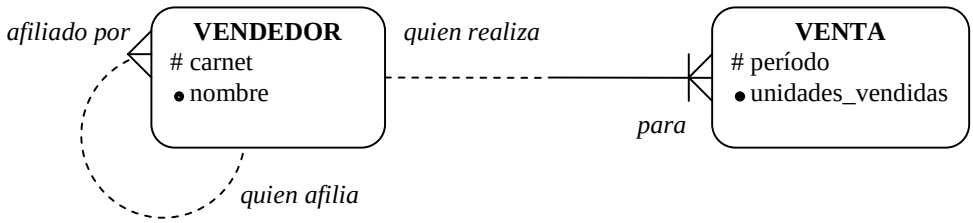

Figura 1: Modelo Entidad Relación

Dicho modelo implantado de manera relacional produce el esquema mostrado en las tablas 1 y 2.

En la figura 2 se presenta una muestra de datos, el atributo período significa el mes y el año en el que se realiza una venta, por ejemplo, 1-2003 significa enero de 2003, 2-2003 febrero de 2003[3] etc.

En dicha figura también se muestra una representación arbórea de la jerarquía de afiliación existente correspondiente a la muestra de datos de la tabla vendedor.

Las instrucciones de creación de tablas se pospondrán hasta la sección 3.2 ya que se hará necesaria la introducción de 2 columnas adicionales para el método aquí propuesto.

Relación Vendedor	
Columnas	**Características**
Carnet	Clave Primaria
Nombre	Obligatorio
afiliado_por	Clave foránea opcional hacia Vendedor

Tabla 1: Relación Vendedor

Relación Venta	
Columnas	**Características**
período	Hace parte de la clave primaria
unidades_vendidas	Obligatorio
cod_vendedor	Hace parte de la clave primaria. Clave foránea hacia vendedor

Tabla 2: Relación Venta

3. Método propuesto

Tomando como base el esquema mostrado en las tablas 1 y2, considérese la pregunta: ¿Cuántos afiliados (directos e indirectos) le ha llevado el vendedor con carnet 81 (Carlos) a la compañía? En este caso la respuesta es 6. Dicha consulta se puede resolver haciendo uso de múltiples reuniones o usando el operador *connect by* en Oracle como se muestra a continuación:

```
SELECT COUNT(*)-1 AS TOTAL
FROM vendedor
START WITH carnet = 81
CONNECT BY PRIOR carnet = afiliado_por;
```

[3] Por supuesto se puede crear alguna codificación para minimizar el tamaño de este campo, pero por simplicidad se dejará así.

En DB2 se puede hacer uso del operador *with recursive* proveniente del estándar SQL-99 así:

```
WITH temptab(carnet) AS
( SELECT v.carnet
  FROM vendedor v
  WHERE carnet = 81
  UNION ALL
  SELECT sub.carnet
  FROM vendedor sub, temptab super
  WHERE sub.afiliado_por = super.carnet
)
SELECT COUNT(*)-1 AS total
FROM temptab;
```

Sección de inicialización

Sección recursiva

Sección de resultados

Se propone ahora el siguiente método:

A cada vendedor se le asignará una clave *inteligente* llamada *ruta* la que permitirá de una manera "camuflada" mantener el camino de los ancestros a los que pertenece cada individuo. Las letras servirán para tal propósito[4]. Veamos en que consiste:

Vendedor

carnet	*nombre*	*afiliado_por*
13	Ana	null
23	León	13
81	Carlos	13
12	Héctor	23
14	Juan	23
5	Lisa	81
6	María	81
33	Pedro	81

Venta

período	*cod_vendedor*	*unidades_vendidas*
1-2003	13	100
1-2003	23	200
1-2003	81	300
1-2003	12	500
1-2003	6	100
1-2003	17	250
2-2003	13	300
2-2003	5	100

9	Luis	14	2-2003	39	50
17	Dora	6	2-2003	9	200
27	Iván	5	2-2003	17	50
39	Rosa	27		…	
	…				

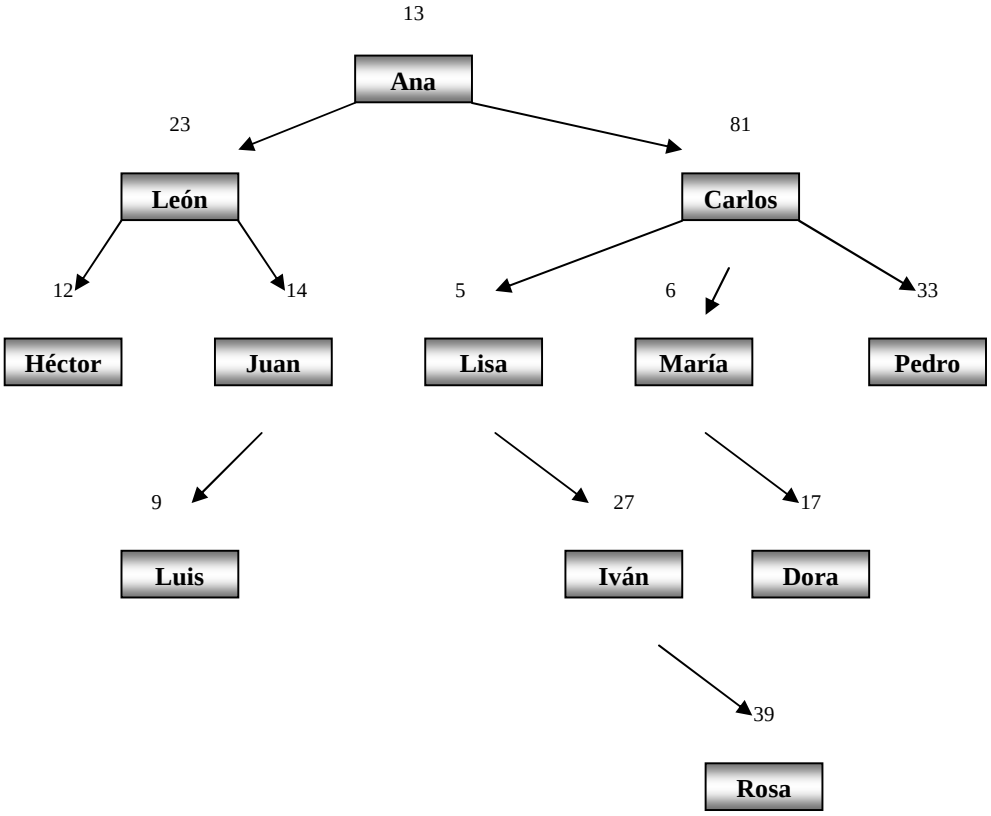

Figura 2: Muestra de datos y representación arbórea de la jerarquía de afiliación.

[4] Este sistema tal y como se presenta aquí posee algunas limitaciones que pueden ser resueltas, véase sección 4.

Al vendedor que da lugar al nacimiento de todo el árbol[5] (Ana en este caso) se le asigna la cadena con la letra *a*. A sus "hijos", León y Carlos se les asigna respectivamente las cadenas *aa* y *ab*. A los hijos de Carlos: Lisa, María y Pedro se les asignarán las cadenas *aba*, *abb* y *abc* respectivamente. La lista completa de asignaciones de claves inteligentes se muestra en la figura 3.

En las secciones siguientes se analizan la utilidad y algunos problemas de este sistema.

3.1 El aspecto de la inserción

En este punto se evidencia un problema: la inserción de los valores en la columna ruta. Por supuesto que el usuario final debe ser liberado del mantenimiento de dicha columna. Sería muy incómodo y además susceptible de errores realizar dicha inserción manualmente.
Suponiendo que no existe la opción de creación de disparadores (los cuales facilitarían el proceso de inserción) se añadirá a la tabla otra columna adicional por aspectos de facilidad (aunque no es estrictamente necesaria). Esta columna se denominará *próximo* y tiene el propósito de guardar para cada vendedor el valor correspondiente a la próxima letra que deberá asignársele al siguiente vendedor que él afilie directamente.

Vendedor

carnet	*nombre*	*afiliado_por*	*ruta*
13	Ana	null	a
23	León	13	aa
81	Carlos	13	ab
12	Héctor	23	aaa
14	Juan	23	aab
5	Lisa	81	aba
6	María	81	abb
33	Pedro	81	abc

[5] Podrían ser varios los vendedores los que dan lugar al nacimiento de sus respectivas jerarquías de afiliación.

9	Luis	14	aaba
17	Dora	6	abba
27	Iván	5	abaa
39	Rosa	27	abaaa

Figura 3: Adición de clave inteligente ruta a la relación vendedor

Por ejemplo supóngase que Carlos afilia a un nuevo vendedor llamado Tomás con carnet 85, como Carlos ya tiene 3 "hijos": Lisa, María y Pedro, con cadenas *aba*, *abb* y *abc* entonces el próximo hijo de Carlos tendrá como cadena *abd*. Por lo tanto *antes* de la inserción de Tomás la tabla Vendedor luce como se muestra en la figura 4.

Vendedor

carnet	*nombre*		*afiliado_por*	*ruta*	*próximo*
13	Ana	null	a	c	
23	León	13	aa	c	
81	Carlos	13	ab	**d**	
12	Héctor	23	aaa	a	
14	Juan	23	aab	b	
5	Lisa	81	aba	b	
6	María	81	abb	b	
33	Pedro	81	abc	a	
9	Luis	14	aaba	a	
17	Dora	6	abba	a	
27	Iván	5	abaa	b	
39	Rosa	27	abaaa	a	

Figura 4: Representación arbórea de la jerarquía de afiliación

---

Nuevamente véase la sección 4. Para este ejemplo se asume una sola jerarquía presente en la tabla vendedor.

Luego de la inserción de Tomás las cosas estarán de la siguiente manera:
Una nueva fila para Tomás: (85, Tomás, 81, abd, a) y la fila de Carlos en su columna próximo cambia la *d* por la *e*. El resto de la relación continúa intacta.

3.2 Creación del esquema

La creación de la tabla, y de las inserciones correspondiente al esquema de la tabla 1 se muestran a continuación. PL/SQL [Urman 01] y SQL de Oracle son utilizados. La creación del esquema de la tabla 2 se muestra en la sección 3.3.2.

```
CREATE TABLE vendedor(
 carnet          NUMBER(5)  PRIMARY KEY,
 nombre          VARCHAR2(15)      NOT NULL,
 afiliado_por    NUMBER(5)  REFERENCES vendedor,
 proximo         CHAR(1)       NOT NULL,
 ruta            VARCHAR2(30)      NOT NULL UNIQUE --Clave alternativa
);
```

3.2.1 Reglas de inserción

- Todo registro que ingresa tiene su campo *próximo* igual a ‘a’.
- Cuando se va a ingresar un registro que es raíz de una jerarquía, su campo *afiliado_por* debe ser nulo y su campo *ruta* igual a ‘a’. Ejemplo:

```
INSERT INTO vendedor VALUES(13,'Ana', null,'a','a');
```

Para los registros no raíces, su campo ruta debe hacerse igual a la ruta de su registro padre (vendedor que lo afilia) concatenado con el valor del campo próximo y debe actualizarse el valor del campo próximo del registro padre a la siguiente letra. Se requiere entonces una inserción y una actualización para cada registro que se ingrese.

```
INSERT INTO vendedor VALUES(23,'León',13,'a',
(SELECT ruta||proximo FROM vendedor WHERE carnet=13));
UPDATE vendedor SET proximo = CHR(ASCII(proximo)+1) WHERE carnet=13;
```

Luego de realizar las inserciones y actualizaciones se obtiene la tabla presentada en la figura 4.

3.3 Consultas

Sea la consulta obtener los vendedores que han sido afiliados por vendedores que han sido afiliados por el vendedor con carnet 81 (es decir los "nietos" del vendedor con carnet 81) Sin utilizar las columnas ruta y próximo la respuesta clásica a esta consulta implica una reunión de la tabla consigo misma:

```
SELECT v2.carnet, v2.nombre
FROM vendedor v1, vendedor v2
WHERE v1.afiliado_por = 81 AND
v1.carnet = v2.afiliado_por;
```

Con el método propuesto se evita la reunión y la consulta se simplifica a:

```
SELECT carnet, nombre
FROM vendedor
WHERE ruta like 'ab__';  // dos underlines
```

Ver igualmente la nota 1 al final de la sección 3.3.1.

Una serie de consultas típicamente recursivas se resuelven haciendo uso de las columnas ruta y próximo en la tabla 3. Se presenta el enunciado de la consulta desde el punto de vista de la relación vendedor, el enunciado equivalente usando terminología de árboles [Horowitz 84] y la solución correspondiente.

3.3.1 Consultas basadas en la tabla vendedor

	Enunciado consulta	**Enunciado en términos de árboles**	**Consulta solución**
1	¿ Cuál es el total de vendedores de la compañía?	Total de nodos del árbol	SELECT COUNT(*) FROM vendedor;
2	¿Cuántos vendedores (directos e indirectos) le ha llevado un vendedor dado a la compañía? Ejemplo: Vendedor con carnet 81 (Carlos).	Total de nodos que conforman el subárbol cuyo nodo raíz es dado.	SELECT COUNT(*) FROM vendedor WHERE ruta LIKE 'ab_%'; Ver nota 1
3	Imprimir todos los vendedores (directos e indirectos) que ha afiliado un vendedor dado a la compañía. Ejemplo: Vendedor con carnet 81	Descendientes de un nodo dado.	SELECT carnet,nombre FROM vendedor WHERE ruta LIKE 'ab_%'; Ver nota 1
4	Imprimir todos los vendedores directos que ha afiliado un vendedor dado a la compañía. Ejemplo: Vendedor con carnet 81	Descendientes directos de un nodo dado.	SELECT carnet,nombre FROM vendedor WHERE ruta LIKE 'ab_';
5	Imprimir todos los vendedores indirectos que ha afiliado un vendedor dado a la compañía. Ejemplo: Vendedor con	Descendientes indirectos de un nodo dado.	SELECT carnet,nombre FROM vendedor WHERE ruta LIKE 'ab_%' AND Ruta NOT LIKE 'ab_'; Ver nota 4

	carnet 81		
6	Imprimir todos los vendedores directa o indirectamente afiliados por un vendedor dado y que no han afiliado a otros vendedores. Ejemplo: Vendedor con carnet 81	Hojas descendientes de un nodo dado	SELECT carnet,nombre FROM vendedor WHERE ruta LIKE 'ab_%' AND proximo='a'; Ver nota 8
7	¿Cuál es la longitud de la cadena de afiliados más larga?	Altura del árbol	SELECT MAX(LENGTH(ruta)) FROM vendedor;
8	¿Cuáles vendedores no han conseguido afiliar a otros vendedores?	Hojas del árbol	SELECT carnet, nombre FROM vendedor v1 WHERE proximo = 'a';
9	¿Cuáles vendedores han afiliado al menos a otro vendedor?	Nodos no Hojas del árbol	SELECT carnet, nombre FROM vendedor v1 WHERE proximo <> 'a';
10	¿Cúal es el número máximo de vendedores afiliados directamente por un mismo vendedor?	Grado del árbol	SELECT MAX(ASCII(proximo)–97) FROM vendedor; Ver nota 2
11	Dado el carnet de 2 vendedores determinar si han sido afiliados *directamente* por el mismo vendedor Por ejemplo los vendedores con carnets 5 y 33.	Decir si 2 nodos son hermanos	SELECT 'Si' FROM vendedor a, vendedor b WHERE a.carnet=5 AND b.carnet=33 AND SUBSTR(a.ruta,1,LENGTH(a.ruta)-1)= SUBSTR(b.ruta,1,LENGTH(b.ruta)-1); Ver nota 3

12	Imprimir la ruta de afiliación que conduce hasta un vendedor determinado. Ejemplo: Ruta de afiliación hasta Iván (carnet 27)	Ancestros de un nodo	SELECT carnet, nombre FROM vendedor v1 WHERE (SELECT ruta FROM vendedor WHERE carnet = 27) LIKE v1.ruta || '_%'; Ver nota 5
13	Dado el carnet de 2 vendedores determinar los vendedores comunes en la cadena de afiliación de ambos. Ejemplo: Para Rosa(carnet 39) y Pedro (carnet 33).	Ancestros comunes a un par de nodos	SELECT carnet,nombre, ruta AS comun FROM vendedor v1 WHERE (SELECT ruta FROM vendedor WHERE carnet = 39) LIKE v1.ruta || '_%' AND (SELECT ruta FROM vendedor WHERE carnet = 33) LIKE v1.ruta || '_%';
14	Dado el carnet de 2 vendedores determinar el vendedor más cercano en la cadena de afiliación común a ambos Ejemplo: Para Rosa(carnet 39) y Pedro (carnet 33).	Ancestro más cercano de un par de nodos	Crear una vista con la consulta anterior llamada ancest_comunes y luego ejecutar: SELECT carnet,nombre FROM ancest_comunes WHERE LENGTH(comun) = (SELECT MAX(LENGTH(comun)) FROM ancest_comunes); Ver nota 6
15	Imprimir todos los vendedores ubicados en un mismo nivel en las cadenas	Imprimir todos los nodos de un mismo nivel	SELECT carnet, nombre FROM vendedor WHERE LENGTH(ruta) = 3;

	de afiliación. Ejemplo: Nodos del nivel 3		
16	Imprimir en orden de arriba abajo la jerarquía de afiliación	Imprimir un árbol por niveles desde la raíz	SELECT carnet, nombre, LENGTH(ruta) as nivel FROM vendedor ORDER BY 3; Ver nota 7
17	Imprimir en orden de abajo hacia arriba la jerarquía de afiliación	Imprimir un árbol por niveles desde las hojas	SELECT carnet, nombre, LENGTH(ruta) as nivel FROM vendedor ORDER BY 3 DESC;

Tabla 3: Consultas jerárquicas sobre la tabla vendedor

Notas

1. *ab* es la ruta que identifica a Carlos (carnet 81). Por supuesto ésta puede ser consultada desde la tabla vendedor mediante:

```
SELECT COUNT(*)
FROM vendedor
WHERE ruta LIKE (SELECT ruta||'_%' FROM VENDEDOR WHERE carnet=81);
```

2. Esta consulta merece especial atención:

Se está sacando provecho de la organización alfabética. Se obtiene el valor ASCII de la columna próximo y se le resta el valor 97 (código ASCII de la '*a*'). La diferencia mayor corresponderá al valor de la columna próximo que contenga la letra más alta, por lo tanto al vendedor que ha afiliado a más vendedores.

Sin esta característica la obtención de dicho valor basado en el uso de la columna ruta también posible, pero es más complejo.

3. Si la consulta devuelve un conjunto vacío, significa que los nodos no son hermanos.

4. Mediante el uso de los comodines _ y % se pueden obtener sólo los nietos o los bisnietos etc.; de un nodo. En el método propuesto por [Web Site 1] se requiere por ejemplo para obtener los nietos una subconsulta o una reunión los cuales son innecesarios aquí.

5. En forma alternativa a LIKE se puede usar también la función INSTR equivalente a la función POSITION del SQL estándar.

6. El uso de la vista no es necesario ya que se puede colocar directamente en la cláusula *from* (inline views).

7. La columna nivel podría ser usada junto con la función LPAD rellenando de blancos a la izquierda para crear un efecto de impresión gráfico que semeje a un árbol (horizontal).

8. En el método posicional propuesto por [Web Site 1] la solución a esta consulta implica una subconsulta correlacionada. Con el método propuesto aquí se evita dicha subconsulta.

3.3.2 Consultas que involucran la tabla venta

La creación de la tabla y las inserciones correspondientes al esquema de la tabla 2 se muestran a continuación.

```
CREATE TABLE venta(
 periodo                VARCHAR2(7),
 cod_vendedor           NUMBER(5)  REFERENCES vendedor,
 unidades_vendidas      NUMBER(6)  NOT NULL,
 PRIMARY KEY(periodo, cod_vendedor)
);

INSERT INTO venta VALUES('1-2003', 13, 100);
INSERT INTO venta VALUES('1-2003', 23, 200);
```

Etc.

En este caso se tiene una clave foránea (cod_vendedor) que referencia a la clave primaria de vendedor (carnet). Sin embargo se verá que desde el punto de vista de algunas consultas[6] puede ser más beneficioso colocar la clave foránea referenciando a la clave alternativa de vendedor ruta.

Consideremos la siguiente consulta por ejemplo:

"Obtener el total de unidades vendidas por todos los vendedores afiliados directa o indirectamente por un vendedor dado incluyendo todos los períodos, por ejemplo para el vendedor con carnet 81 (Carlos)".

De acuerdo al esquema actual, una solución para esta consulta es:

```
SELECT SUM(unidades_vendidas)
FROM vendedor, venta
WHERE carnet = cod_vendedor AND
ruta LIKE(SELECT ruta||'%' FROM vendedor WHERE carnet = 81);
```

Resultado 850. Sin embargo supongamos que la tabla venta cambia la clave foránea y se construye así:

```
DROP TABLE venta;
CREATE TABLE venta(
 periodo              VARCHAR2(7),
 ruta_vendedor        VARCHAR2(30)      REFERENCES vendedor(ruta),
 unidades_vendidas    NUMBER(6)  NOT NULL,
 PRIMARY KEY(periodo, ruta_vendedor)
);
```

Se realizan las inserciones correspondientes:

[6] Pero quizás no desde el punto de vista de almacenamiento ya que la columna ruta tiende a ser más larga que la columna carnet, se hace necesario estudiar este aspecto de acuerdo a la naturaleza de los datos y establecer un punto de balance entre almacenamiento y ciertas consultas.

```
INSERT INTO venta VALUES('1-2003', 'a', 100);
INSERT INTO venta VALUES('1-2003', 'aa', 200);
INSERT INTO venta VALUES('1-2003', 'ab', 300);
```
Etc.

La inserción de los valores para esta clave foránea podría manejarse por medio de un disparador y el usuario final podría seguir trabajando con los carnets y así evitar errores potenciales. Ahora la consulta queda así:

```
SELECT SUM(unidades_vendidas)
FROM venta
WHERE ruta_vendedor LIKE(SELECT ruta||'%' FROM vendedor WHERE carnet = 81);
```

Resultado 850. Nótese como de esta forma se evita una reunión. Esta es una de las bondades de utilizar como clave foránea la ruta y no el carnet.
Las consultas que se presentan en la tabla 4 trabajan con la clave foránea ruta_vendedor.

	Enunciado consulta	**Enunciado en términos de árboles**	**Consulta solución**
1	Total de unidades vendidas por todos los vendedores afiliados directa o indirectamente por un vendedor dado incluyendo todos los períodos pero sin incluirlo a él. Por ejemplo para el vendedor con carnet 81 (Carlos).	Sumatoria de todos los valores[7] que conforman un subárbol dado sin incluir la raíz del subárbol	SELECT SUM(unidades_vendidas) FROM venta WHERE ruta_vendedor LIKE (SELECT ruta||'_%' FROM vendedor WHERE carnet = 81); Resultado: 550

[7] Aquí los valores del árbol se refieren a las unidades vendidas al realizar la reunión (join).

2	Total de unidades vendidas por todos los vendedores afiliados directa o indirectamente por un vendedor dado en un período dado. Por ejemplo para el vendedor con carnet 81 (Carlos) en el período 1-2003	Sumatoria de todos los valores que conforman un subárbol dado incluyendo la raíz del subárbol pero restringiendo los nodos con una condición adicional.	SELECT SUM(unidades_vendidas) FROM venta WHERE periodo = '1-2003' AND ruta_vendedor LIKE (SELECT ruta||'%' FROM vendedor WHERE carnet = 81); Resultado: 650
3	Total de unidades vendidas por todos los vendedores afiliados directa o indirectamente por un vendedor dado en un período dado pero sin incluirlo a él. Por ejemplo para el vendedor con carnet 81 (Carlos) en el período 1-2003	Sumatoria de todos los valores que conforman un subárbol dado sin incluir la raíz del subárbol pero restringiendo los nodos con una condición adicional.	SELECT SUM(unidades_vendidas) FROM venta WHERE periodo = '1-2003' AND ruta_vendedor LIKE (SELECT ruta||'_%' FROM vendedor WHERE carnet = 81); Resultado: 350
4	Imprimir el total de unidades vendidas *por cada* vendedor de la compañía incluyendo todas las unidades de los vendedores afiliados por él directa o indirectamente contemplando todos los períodos.	Sumatoria de todos los valores para *cada uno* de los subárboles incluyendo la raíz de cada uno.	SELECT carnet,nombre, (SELECT SUM(unidades_vendidas) FROM venta WHERE ruta_vendedor LIKE (SELECT ruta||'%' FROM vendedor WHERE carnet = v1.carnet)) AS total

			FROM vendedor v1; Ver figura 5

Tabla 4: Consultas jerárquicas que involucran la tabla venta

carnet	*nombre*	*total*
13	Ana	2150
23	León	900
81	Carlos	850
12	Héctor	500
14	Juan	200
5	Lisa	150
6	María	400
33	Pedro	
9	Luis	200
17	Dora	300
27	Iván	50
39	Rosa	50

Figura 5: Resultado de la consulta 4, de la tabla 4

4. Desventajas y mejoras al método

El método presenta algunos aspectos que pueden ser mejorados:

- Desde el punto de vista de *análisis* del modelo, la presencia de las columnas ruta y próximo no son “naturales”. Este es un factor menor pero debe ser advertido en una etapa temprana y debe ser bien documentado su propósito.
- Almacenamiento: Es claro que las columnas ruta y próximo gastan espacio adicional. Si la cantidad de datos a manejar es enorme, esto deberá ser considerado. La columna *próximo*

tal y como fue propuesta gasta un único carácter, pero aun así puede llegar a ser un gasto considerable.

- Limitaciones de representación del árbol:

Un lector atento habrá notado que en el método planteado un nodo sólo puede tener como máximo 26 hijos (las letras del alfabeto). En el caso de estudio esto equivale a decir que un vendedor sólo puede afiliar directamente máximo a 26 vendedores. Una forma de eliminar esta restricción es:

En vez de manejar los hijos de un nodo como *aa, ab, ac, …, az* se puede adicionar un entero positivo antes de la segunda letra así: *a1a, a1b, a1c, … , a1z, a2a, a2b, a2c, …, a2z, a3a,* a3b, *…, a3z,…, a10a, a10b, a10z, a11a,* etc. de esta manera un nodo puede tener potencialmente infinitos hijos. Por supuesto la cantidad de almacenamiento aumenta de esta manera pero permite seguir utilizando el método propuesto.

Otra aspecto tiene que ver con la altura del árbol, ya que estará limitada por el máximo número que se pueda representar en una cadena de caracteres (columna ruta). Este problema es más fácil de solucionar, por ejemplo se podría usar una segunda cadena de caracteres (ruta2) y concatenarlas en las consultas cuando sea necesario. De todas formas los límites actuales para cadenas de carácteres son altos en los SGBDs (2000, 4000 y más).

Respecto al número de jerarquías (árboles) que se pueden representar en una tabla, esto es también fácil de manejar, supóngase los datos mostrados en la figura 6.

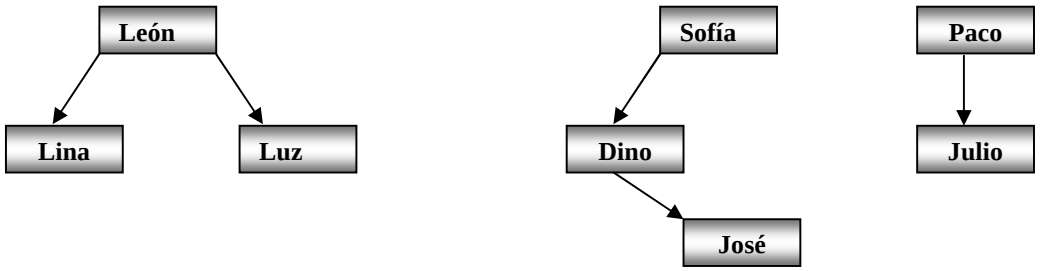

Figura 6: Diversas jerarquías de afiliación

En este caso se tienen 3 jerarquías, “lideradas” por León, Sofía y Paco. Para solucionar el problema de representación en la columna ruta, a cada jerarquía se le asigna un número que se concatena a la columna ruta, tal y como lo muestra la figura 7.

Vendedor

carnet	*nombre*	*afiliado_por*	*ruta*	*próximo*
1	León	null	0-a	c
2	Lina	1	0-aa	a
4	Luz	1	0-ab	a
8	Sofía	null	1-a	c
9	Dino	8	1-aa	a
3	José	9	1-ab	a
5	Paco	null	2-a	b
6	Julio	5	2-aa	a

Figura 7: Diversas jerarquías de afiliación

Con este sistema se pueden manejar teóricamente infinitos árboles. Por supuesto que mientras más árboles existan se requerirán números al inicio de la ruta con mayor número de dígitos, pero en una situación real, consideremos 1 millón de árboles, habrá entonces árboles cuya ruta inicia por 6 dígitos lo cual puede ser manejable. Obviamente el costo de almacenamiento y de mantenimiento se incrementa, pero las consultas propuestas pueden seguir siendo utilizadas[8].

- La formación de ciclos, la cual debe ser evitada en el modelo propuesto. Desde el punto de inserción no hay problema, la actualización y la clave foránea controlan esta posibilidad, sin embargo una actualización posterior podría dañar la tabla y crear ciclos.

5. Conclusiones y trabajos futuros

[8] Algunas de ellas requerirán algunos “ajustes”, por ejemplo la consulta 4 de la tabla 3, entre otras.

Se ha presentado un método alternativo para el manejo de consultas recursivas en SQL. De esta forma se pueden resolver con relativa facilidad y con un tiempo de respuesta rápido problemas que de lo contrario requerirían acudir a lenguaje procedimental o a operadores especializados.

Se planea realizar un estudio de rendimiento entre el uso de *connect by* y el método propuesto. Igualmente dicha comparación serviría para establecer un posible mecanismo de trasformación entre las dos notaciones.

Deben también tratarse los casos de eliminación y actualización (¿qué hacer con los vendedores que fueron afiliados por un vendedor que ya no trabajará en la compañía?) de empleados y hacer uso de "letras" que puedan volver a ser utilizadas.

El método propuesto podría ser utilizado incluso en otros dominios donde la utilización de árboles sea necesaria, por ejemplo en las jerarquías de representación de XML (representación DOM).

Otras variantes y mejoras, como las expuestas en la sección 4 pueden igualmente ser introducidas y métodos de compresión de datos u otras formas de representar las rutas podrían ser diseñados para ahorrar gasto de almacenamiento.

Otras variantes han sido propuestas igualmente: [Celko 00,03] propone un modelo denominado "anidado de árboles" en el cual es complejo manejar ciertas consultas, por ejemplo para hallar la altura del árbol se requiere una reunión de la tabla consigo misma mas un operación de grupo (GROUP BY); en el método aquí propuesto no se requiere ni la reunión ni el agrupamiento (Véase consulta 7 en la tabla 3). Igualmente hallar los descendientes directos de un nodo implica en su modelo una consulta con reunión mas una subconsulta correlacionada, confróntese con la consulta 4 en la tabla 3.

[Moreau 00] presenta un método de notación de ruta posicional, con similitudes con el que aquí se propone, pero no maneja bien ciertas consultas ya que requieren subconsultas o reuniones que pueden ser evitadas (véase consultas 5 y 6 en la tabla 3, entre otras). Se planea realizar nna comparación de rendimiento entre estos diversos métodos.

6. Bibliografía

6.1 Libros

[Celko 00] Celko J.; "*SQL for Smarties*", Morgan Kaufmann, 2000.

[Celko 03] Celko J.; "*Trees & Hierarchies in SQL*", Morgan-Kaufmann, 2003.

[Date 00] Date C.J.; "*Introducción a los Sistemas de BD***",** Prentice Hall, 2000.

[Gulutzan 99] Gulutzan P, Pelzer T.; **"***SQL-99 Complete Really***",** R & B Books, 1999.

[Horowitz 84] Horowitz E.; "*Fundamentals of data structures in Pascal*", Computer Science Press, 1984.

[Melton 98] Melton J. "*Understanding SQL's Stored Procedures : A Complete Guide to SQL/PSM*",
Morgan Kaufmann, 1998.

[Moreau 00] Moreau T., Ben-Gan I.; "*Advanced Transact-SQL for SQL Server 2000*", APress, 2000.

[Oracle 02] Oracle Corporation; "*Oracle9i SQL Referente Release 2 (9.2)*". Oracle. 2002.

[Ullman 01] Ullman J., Widom J.; "*A First Course in Database Systems*", Prentice Hall, 2001.

[Urman 01] Urman S. "*Oracle9i PL/SQL Programming*", McGraw-Hill Osborne Media, 2001

6.2 Recursos en Internet

[Web Site 1] Título: "*Maintaining Hierarchies*", Itzik Ben-Gan
http://www.sqlmag.com/Articles/Index.cfm?ArticleID=8826&pg=1

[Web Site 2] Título: "*Breve Historia de SQL*"
http://www.htmlpoint.com/sql/sql_04.htm

[Web Site 3] Título: "*Migrating Recursive SQL from Oracle to DB2 UDB*", Torsten Steinbach
http://www106.ibm.com/developerworks/db2/library/techarticle/0307steinbach/0307steinbach.html