2196-PS

A COMBINATORIAL OPTIMIZATION PROBLEM IN SQL

Luis Eduardo Muñoz (Departamento de Ingeniería de Sistemas, Universidad Tecnológica de Pereira, Colombia) - lemunozg@utp.edu.co
Francisco J. Moreno (Facultad de Minas, Escuela de Sistemas, Universidad Nacional de Colombia, Sede Medellín, Colombia) - fjmoreno@unalmed.edu.co
Jaime A. Echeverri (Departamento de Ingeniería de Sistemas, Universidad de Medellín, Colombia) - jaecheverri@udem.edu.co

In this paper, we analyze and solve a problem of combinatorial optimization using SQL. Traditionally this kind of problems are treated with efficient techniques in research operations area, such as linear programming, taboo search, neural networks, and genetic algorithms. We do not pretend to compete in performance with these techniques, our goal is to show how this kind of problems can be solved using a language like SQL, a language belonging to the database area. In this way, we can establish a link between these two areas. Moreover, we take advantage of SQL features rarely used like Common Table Expression clause. This clause can be used to simplify some SQL queries.

Keywords: Databases, SQL, research operations, combinatorial optimization.

UN PROBLEMA DE OPTIMIZACIÓN COMBINATORIA EN SQL

En este artículo se analiza y soluciona un problema de optimización combinatoria mediante SQL. Tradicionalmente este tipo de problemas se resuelven a través de técnicas propias del área de investigación de operaciones, como programación lineal, búsqueda tabú, redes neuronales y algoritmos genéticos, donde el principal objetivo es resolverlos eficientemente. En este artículo, no se pretende competir en rendimiento con estas técnicas, la intención es mostrar cómo este tipo de problemas pueden ser abordados mediante un lenguaje como SQL, un lenguaje propio del área de bases de datos, lo que permite establecer un punto de encuentro entre estas dos áreas. En el artículo se aprovecha la cláusula Common Table Expression, una cláusula poco usada de SQL, útil para simplificar la formulación de algunas consultas.

Palavras-chave: Bases de datos, SQL, investigación de operaciones, optimización combinatoria.

1. Introducción

En este artículo se analiza y soluciona un problema de optimización combinatoria mediante SQL sin considerar su opción procedimental PSM[1] [1]. Tradicionalmente este tipo de problemas se resuelven a través de técnicas propias del área de investigación de operaciones [2], como programación lineal, búsqueda tabú, redes neuronales y algoritmos genéticos, donde el principal objetivo es resolverlos eficientemente [3]. Estas soluciones son a menudo implementadas en lenguajes procedimentales como Java, C++ o en herramientas especializadas como GIPALS [4].

En este artículo no se pretende competir en rendimiento con estas técnicas, la intención es mostrar cómo este tipo de problemas pueden ser abordados mediante un lenguaje como SQL, un lenguaje propio del área de bases de datos, con el fin de establecer un punto de encuentro entre estas dos áreas.

La conexión entre este tipo de lenguajes e investigación de operaciones ha sido poco explorada[2] [5]. Esto sugiere la concepción de operadores especializados en SQL para resolver este tipo de problemas, como lo evidencia el problema de optimización combinatoria tratado en la Sección 3.

El artículo también aprovecha características de SQL poco usadas, como la cláusula *Common Table Expresión* (CTE) [6], útil para simplificar la formulación de algunas consultas. Adicionalmente, en el problema tratado se muestra una aplicación del conjunto potencia, traído de la teoría de conjuntos [7].

El artículo se estructura así: en la Sección 2 se ejemplifica la cláusula CTE. En la Sección 3 se presenta el problema de optimización combinatoria y en la Sección 4 se presentan las conclusiones y trabajos futuros.

2. La cláusula CTE

En esta sección se ejemplifica la cláusula CTE mediante un problema que sirve como preámbulo al problema de optimización combinatoria que se analiza en la Sección 3.
El problema se plantea en [8]. Se tienen varias cestas (*bins*) que contienen productos (*items*). Cada cesta se identifica por un código. Por ejemplo, en la cesta con código 1 pueden haber 5 Manzanas y 2 Naranjas, en la cesta 2 pueden haber 2 Manzanas y 3 Naranjas.

Dados un producto y una cantidad específica solicitada, se deben listar las cestas necesarias para satisfacer dicha cantidad, dando prioridad a las cestas con menor código.
Para guardar la información de las cestas y sus productos se dispone de una tabla bodega (*Warehouse*):

---

[1] *Persistent Stored Modules*.
[2] Aunque SQL:2003 incorporó una serie de funciones (covarianza, percentiles, regresiones etc.) de gran ayuda en el campo de la estadística.

```
CREATE TABLE Warehouse
(item VARCHAR(10) NOT NULL,
 bin INTEGER NOT NULL,
 qty INTEGER NOT NULL CHECK (qty >= 0),
 PRIMARY KEY (item, bin)
);
```

item representa el nombre del producto, *bin* el código de la cesta y *qty* el número de unidades de ese producto en la cesta. Una muestra de datos de la tabla Warehouse se presenta en la Tabla 1.

Tabla 1. Muestra de datos de la tabla Warehouse.

item	bin	qty
Manzana	1	5
Manzana	2	0
Manzana	3	2
Manzana	5	9
Manzana	6	1
Manzana	7	6
Naranja	1	2
Naranja	3	3
Naranja	4	5
Naranja	8	1

Supóngase que se solicita una cantidad de 10 Manzanas. De acuerdo con la muestra de datos de la Tabla 1, la respuesta correspondiente se muestra en la Tabla 2.

Tabla 2. Resultados del Problema 1 para 10 Manzanas.

BIN	QTY
1	5
3	2
5	3

Nótese que la cesta 2 no hace parte de la respuesta porque contiene 0 Manzanas y de la cesta 5 se extraen sólo 3 Manzanas (de las 9 que posee) para completar las 10 Manzanas solicitadas.

Una solución para este problema, formulada en SQL y donde se usa una autoreunión (*self-join*), se presenta a continuación [8].

Consulta 1.

```
SELECT W1.bin, CASE WHEN :cant_solicitada >= SUM(W2.qty)
                    THEN W1.qty
                    ELSE :cant_solicitada - (SUM(W2.qty) - W1.qty)
```

```
                    END qty_extracted
FROM Warehouse W1 INNER JOIN Warehouse W2
ON W1.item = W2.item AND W1.bin >= W2.bin
WHERE W1.item = ':pdto_solicitado' AND W1.qty > 0
GROUP BY W1.bin, W1.qty
HAVING SUM(W2.qty) - W1.qty < :cant_solicitada
ORDER BY W1.bin;
```

De otro lado, con el fin de ilustrar la cláusula CTE se puede plantear la siguiente solución alternativa:

Consulta 2a.

```
SELECT bin, CASE WHEN (qty + (SELECT COALESCE(SUM(qty),0)
                              FROM Warehouse W2
                              WHERE item = ':pdto_solicitado ' AND
                                    W2.bin < W1.bin)) > :cant_solicitada
             THEN (:cant_solicitada - (SELECT COALESCE(SUM(qty),0)
                                       FROM Warehouse W2
                                       WHERE item = ':pdto_solicitado ' AND
                                             W2.bin < W1.bin))
             ELSE qty
             END qty_extracted
FROM Warehouse W1
WHERE qty > 0 AND item = ':pdto_solicitado ' AND
      :cant_solicitada > (SELECT COALESCE(SUM(qty),0)
                          FROM Warehouse W2
                          WHERE item = ':pdto_solicitado ' AND W2.bin < W1.bin)
ORDER BY bin;
```

Esta solución se puede simplificar. Nótese que la expresión:

```
(SELECT COALESCE(SUM(qty),0)
FROM Warehouse W2
WHERE item = ':pdto_solicitado ' AND
      W2.bin < W1.bin)
```

Se repite tres veces. Por medio de la cláusula CTE, la consulta se puede reescribir así:

Consulta 2b.

```
WITH (SELECT COALESCE (SUM(qty), 0)
     FROM Warehouse W2
     WHERE item = ':pdto_solicitado ' AND W2.bin < W1.bin)
AS X(aux)
SELECT bin, CASE WHEN (W1.qty + X.aux) > :cant_solicitada
```

```
                THEN  (:cant_solicitada - X.aux)
                ELSE W1.qty
                END qty_extracted
 FROM Warehouse W1, X
 WHERE qty > 0 AND W1.item = ':pdto_solicitado ' AND :cant_solicitada > X.aux
ORDER BY bin;
```

La consulta 2 evita la autoreunión de la consulta 1 mediante subconsultas correlacionadas. Sin embargo el rendimiento de la consulta 2 en los Sistemas de Gestión de Bases de Datos (SGBD) tiende a ser inferior frente a una solución equivalente basada en reuniones. Por ejemplo en el SGBD Oracle, mediante su herramienta TKPROF, se analizó el rendimiento de ambas consultas con las siguientes muestras de datos. Dos productos (Manzana y Naranja) y

- pocos datos: 50 cestas con cantidades entre 0 y 9; y
- muchos datos: 5000 cestas con cantidades entre 0 y 9.

Las consultas se probaron con una cantidad solicitada de 10 Manzanas.
Para analizar el rendimiento se consideraron las siguientes tasas. Una información detallada sobre éstas se puede ver en [9].

Tasa 1. Bloques leídos vs. Filas procesadas (*blocks read to rows processed*). Esta tasa indica de una manera general, el costo relativo de una consulta. Mientras más bloques tienen que ser accedidos en relación con las filas retornadas, la fila traída será mucho más costosa. Se debe procurar un valor menor a 10, sin embargo valores entre 10 y 20 son aceptables. Un valor mayor a 20, indica problemas de rendimiento.

Tasa 2. Análisis sintáctico vs. Ejecución (*parse count over execute count*). Idealmente, el conteo de *parsing* debe ser cercano a uno. Si este valor es alto en relación con el conteo de ejecuciones, la sentencia entonces se analizó sintácticamente varias veces sin necesidad. El valor que se debe procurar es 1.

Tasa 3. Filas traídas vs. Traídas (*rows fetched over fetches*). Esta tasa indica el nivel en que la capacidad del *array fetch*[3] se usó. Una tasa cercana a 1, indica que hubo poco procesamiento a través de *arrays*, lo que significa que este aspecto se puede mejorar.

Tasa 4. Lecturas de disco vs. Lecturas lógicas (*disks reads to logical reads*). Esta tasa muestra el porcentaje de veces en las que el SGBD no encontró las filas solicitadas en el *buffer* de la zona de caché y por lo tanto trajo los bloques desde disco. Generalmente se procura que esta tasa no esté por encima de 10%.

Para las dos consultas se obtuvieron los resultados de la Tabla 3. Las figuras 1, 2, 3 y 4 muestran los resultados para las tasas que evidenciaron diferencias en rendimiento.

Tabla 3. Tasas obtenidas para las consultas.

[3] *Array fetch* es una característica de Oracle que recupera más de una fila de una consulta por cada *fetch*.

	Tasa 1	**Tasa 2**	**Tasa 3**	**Tasa 4**	**Cantidad de Datos**
VALOR NORMAL	Menor que 10	1 (ó cercano a 1)	Mientras mayor mejor	Menor del 10%	
Consulta 1	18	1	1	0,027777778	**Muchos**
	23	1	1	0,02173913	**Pocos**
Consulta 2	18108,5	1	1	2,76113E-05	**Muchos**
	334	1	1	0,001497006	**Pocos**

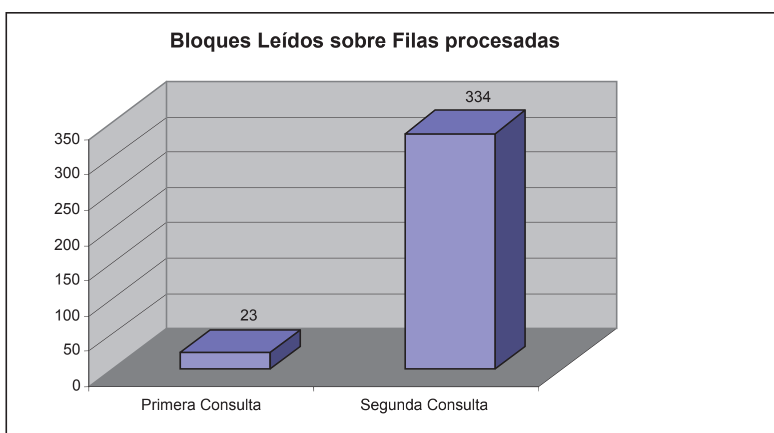

Figura 1. Bloques leídos vs. Filas procesadas con pocos datos.

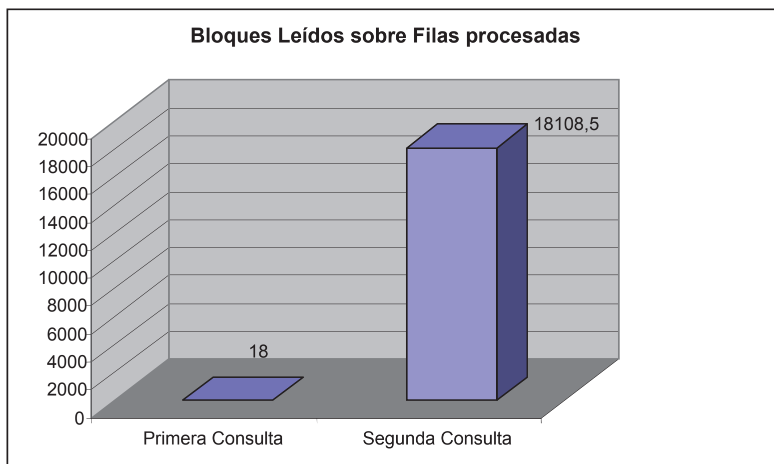

Figura 2. Bloques leídos vs. Filas procesadas con muchos datos.

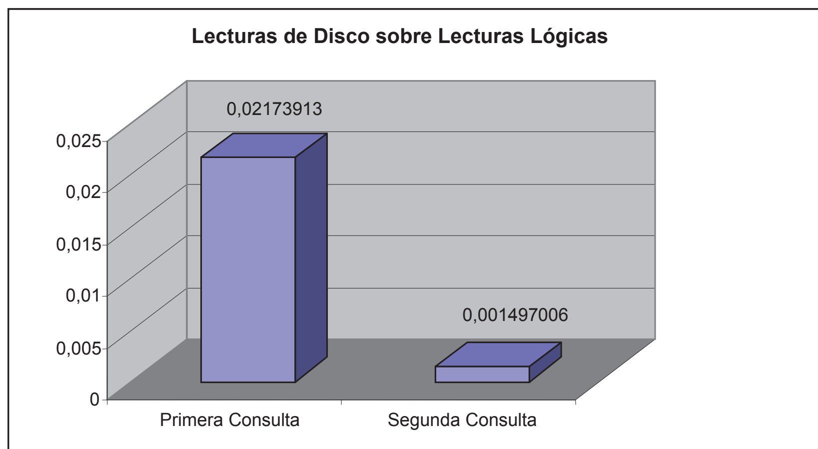

Figura 3. Lecturas de disco vs. Lecturas lógicas con pocos datos.

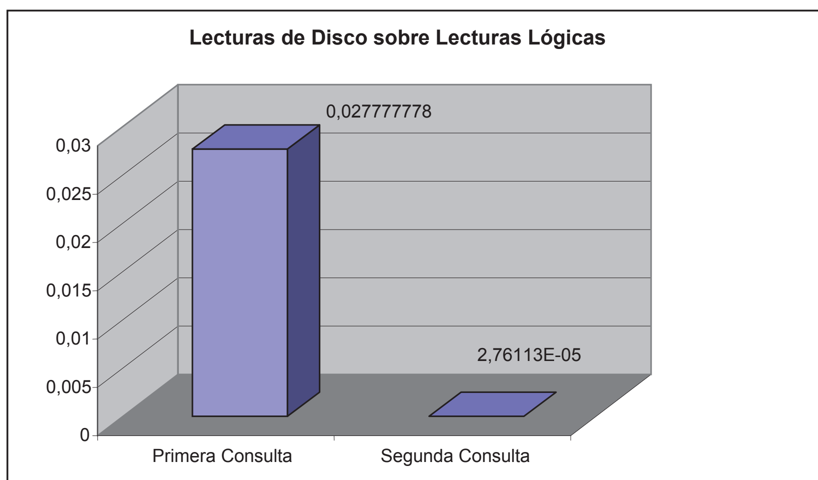

Figura 4. Lecturas de disco vs. Lecturas lógicas con muchos datos.

3. Un problema de optimización combinatoria

Continuando con el ejemplo de la Sección 2, supóngase ahora que se desea seleccionar el mínimo número de cestas que satisfagan una cantidad solicitada de un producto específico. Se deben generar todas las soluciones posibles.
Para solucionar este problema se usa el conjunto potencia, es decir, el conjunto de todos los subconjuntos de un conjunto. El conjunto potencia en SQL se puede obtener mediante una cláusula recursiva [10] o mediante múltiples uniones [11].

En Oracle se puede obtener el conjunto potencia mediante el operador POWERMULTISET [12]. Sin embargo, se deben usar características objeto-relacionales [13] que oscurecen un poco el lenguaje. Primero se debe crear un tipo de tabla anidada:

```
CREATE TYPE tnum AS TABLE OF NUMBER(6);
```

Esta sentencia crea un tipo de tabla anidada llamado *tnum* donde cada elemento de la tabla anidada es un número de máximo 6 dígitos.
Ahora mediante la siguiente consulta se puede obtener el conjunto potencia de los elementos 1, 2 y 3.

```
SELECT *
FROM TABLE(POWERMULTISET(tnum(1,2,3)));
```

El resultado se muestra en la Tabla 4. Oracle genera automáticamente un nombre de columna denominado COLUMN_VALUE.

Tabla 4. Conjunto potencia de {1, 2, 3} obtenido con POWERMULTISET.

COLUMN_VALUE
TNUM(1)
TNUM(2)
TNUM(1, 2)
TNUM(3)
TNUM(1, 3)
TNUM(2, 3)
TNUM(1, 2, 3)

A partir de este resultado se puede plantear una solución al problema enunciado. Se asigna un consecutivo a cada miembro del conjunto potencia así:

```
SELECT ROWNUM AS consecutivo, COLUMN_VALUE AS grupo
FROM TABLE(POWERMULTISET(tnum(1,2,3)));
```

ROWNUM es una pseudo columna de Oracle que enumera consecutivamente las filas retornadas por una consulta[4]. El resultado se muestra en la Tabla 5.

Tabla 5. Enumeración de los miembros del conjunto potencia.

CONSECUTIVO	GRUPO
1	TNUM(1)
2	TNUM(2)
3	TNUM(1, 2)
4	TNUM(3)
5	TNUM(1, 3)
6	TNUM(2, 3)
7	TNUM(1, 2, 3)

Ahora se "desanidan" los elementos del grupo correspondiente a cada consecutivo. Esta trasformación se logra así:

[4] En SQL estándar esto se logra con la función ROW_NUMBER.

```
SELECT consecutivo, g.COLUMN_VALUE AS bin
FROM (SELECT ROWNUM AS consecutivo, COLUMN_VALUE AS grupo
        FROM TABLE(POWERMULTISET(tnum(1,2,3)))
      ), TABLE(grupo) g
ORDER BY consecutivo;
```

Esta consulta genera los resultados de la Tabla 6.

Tabla 6. Desanidamiento de los elementos de cada miembro del conjunto potencia.

CONSECUTIVO	BIN
1	1
2	2
3	1
3	2
4	3
5	1
5	3
6	2
6	3
7	1
7	2
7	3

Ahora se cambia el conjunto tnum(1,2,3) por el conjunto de códigos de cestas que poseen una cantidad mayor que cero del producto solicitado. Para ello se define una vista Desanidados:

```
CREATE VIEW Desanidados AS
SELECT consecutivo, g.COLUMN_VALUE AS bin
FROM
(SELECT ROWNUM AS consecutivo, COLUMN_VALUE AS grupo
 FROM TABLE(POWERMULTISET(CAST(MULTISET(
                    SELECT bin
                    FROM Warehouse
                    WHERE item = ':pdto_solicitado ' AND qty > 0) AS tnum)))
), TABLE(grupo) g;
```

Se usan los operadores CAST y MULTISET [13] de Oracle para facilitar la trasformación de la subconsulta (en negrilla) y para desanidar.
A partir de la muestra de datos de la Sección 2, para el producto Manzana, esta vista produce el conjunto potencia de los elementos (códigos de cestas) 1, 3, 5, 6 y 7, es decir, 31 miembros en total (no se incluye el conjunto vacío).

El proceso continúa con la obtención de la cantidad total del producto solicitado en cada miembro del conjunto potencia y con el total de cestas en cada miembro. Para ello se define la vista Candidatos:

```
CREATE VIEW Candidatos AS
SELECT consecutivo, SUM(qty) AS total_items, COUNT(*) AS total_bins
FROM Warehouse W, Desanidados P
WHERE W.bin = P.bin AND item = ':pdto_solicitado '
GROUP BY consecutivo
HAVING SUM(qty) >= :cant_solicitada;
```

Los resultados para una cantidad solicitada de 16 Manzanas se muestran en la Tabla 7.

Tabla 7. Vista Candidatos.

CONSECUTIVO	TOTAL_ITEMS	TOTAL_BINS
7	**16**	**3**
15	17	4
21	**20**	**3**
22	**17**	**3**
23	22	4
28	**16**	**3**
29	21	4
30	18	4
31	23	5

Es decir, mediante la vista Candidatos se obtienen los miembros del conjunto potencia (nueve) que pueden satisfacer la cantidad solicitada.

A partir de la vista Candidatos se obtienen los miembros del conjunto potencia con menor número de cestas (en negrilla en la Tabla 7), para ello se crea la vista Seleccionados:

```
CREATE VIEW Seleccionados AS
SELECT consecutivo
FROM Candidatos
WHERE total_bins = (SELECT MIN(total_bins)
                    FROM Candidatos);
```

Esta vista genera los resultados de la Tabla 8.

Tabla 8. Vista Seleccionados.

CONSECUTIVO
7
21
22
28

Ya se identificaron los miembros del conjunto potencia que conforman la solución. Ahora por medio de la vista Desanidados y de la tabla Warehouse, se obtienen las cestas de cada miembro del conjunto potencia candidato con su respectiva cantidad. Para ello se crea una vista Resultados:

```
CREATE VIEW Resultados AS
SELECT D.consecutivo AS grupo, D.bin AS bin, W.qty AS qty
FROM Desanidados D, Seleccionados S, Warehouse W
WHERE D.consecutivo = S.consecutivo
AND W.bin = D.bin AND item = ':pdto_solicitado ';
```

Para el ejemplo con 16 Manzanas, esta vista genera los resultados de la Tabla 9.

Tabla 9. Vista Resultados.

GRUPO	BIN	QTY
7	1	5
7	3	2
7	5	9
21	1	5
21	5	9
21	7	6
22	3	2
22	5	9
22	7	6
28	5	9
28	6	1
28	7	6

El paso final logra que en cada miembro (grupo) la suma sea igual a la cantidad solicitada. Para ello se plantea la siguiente consulta: se imprime el código de cada grupo (miembro del conjunto potencia), el código de la cesta y su cantidad, teniendo en cuenta que *en cada grupo* a la cesta que posee mayor cantidad se le extrae la cantidad necesaria para satisfacer la cantidad solicitada (si hay empates se selecciona la cesta con mayor código).

```
WITH (SELECT MAX(bin)
        FROM Resultados
        WHERE grupo = R.grupo AND qty = (SELECT MAX(qty)
                                          FROM Resultados
                                          WHERE grupo = R.grupo))
AS X(aux)
SELECT grupo, bin,
          CASE WHEN bin != X.aux
          THEN qty
           ELSE
```

```
          :cant_solicitada - (SELECT COALESCE(SUM(qty),0)
                               FROM Resultados
                               WHERE grupo = R.grupo AND bin != X.aux
                               )
           END cant
FROM Resultados R, X
ORDER BY grupo,bin;
```

Para el ejemplo con 16 Manzanas se obtienen los resultados de la Tabla 10.

Tabla 10. Resultados finales para el Problema 2 con 16 Manzanas.

GRUPO	BIN	QTY_EXTRACTED
7	1	5
7	3	2
7	5	9
21	1	5
21	5	5
21	7	6
22	3	2
22	5	8
22	7	6
28	5	9
28	6	1
28	7	6

4. Conclusiones y Trabajos Futuros

Los resultados de la Sección 2 sugieren que en Oracle las consultas correlacionadas tienden a ser inferiores en rendimiento frente a una alternativa basada en reuniones. Mecanismos, como transformaciones semánticas y sintácticas, para optimizar las consultas correlacionadas se deben investigar. Un punto de partida son los trabajos de [14], [15] y [16]. Igualmente se debe confrontar su rendimiento en otros SGBD como SQL Server y DB2.
La solución propuesta para el problema de la Sección 3, sugiere el diseño de operadores especializados en SQL para abordar este tipo de problemas, es decir, operadores que ofrezcan mayor expresividad para el campo de la investigación de operaciones en SQL. La expresividad se refiere a la presencia de operadores muy "poderosos" que realicen procesos complejos [17], lo que deriva en consultas concisas. También se debe evaluar la solución propuesta frente a las técnicas tradicionales usadas en investigación de operaciones.

Bibliografía

[1] Melton J., *Understanding SQL Stored Procedures: A Complete Guide to SQL/PSM*, Morgan Kaufmann, 1998.
[2] Taha H., *Investigación de Operaciones*, Prentice Hall, México, 1998.

[3] Cook W., Cunningham W., Pulleyblank W., Schrijver A., *Combinatorial Optimization*; John Wiley & Sons, 1997.
[4] *Entorno de Programación GIPALS.*
http://www.freedownloadmanager.org/es/downloads/GIPALS_-_Linear_Programming_Environment_26067_p/
Visitado Enero de 2010.
[5] Celko J., *SQL for Smarties Advanced SQL Programming*, Elsevier/Morgan Kaufmann, 2005.
[6] Naumann F., Häussler M., *Declarative Data Merging with Conflict Resolution*, Proceedings of the International Conference on Information Quality, Cambridge, 2002.
[7] Kolman B., Busby R.C., *Estructuras de Matemáticas Discretas para la Computación*, Prentice-Hall Hispanoamericana, 1997.
[8] DBMS Enero de 1998. *Making Things Secure*.
http://www.dbmsmag.com/9801d06.html
Visitado Mayo de 2009.
[9] Harrison G., *Oracle SQL High-Performance Tuning*, Prentice Hall, 2000.
[10] Finkelstein S., Mattos N., Mumick I.S., Pirahesh H., *Expressing Recursive Queries in SQL*, ANSI Document X3H2-96-075rl, 1996.
[11] Celko J., Comunicación privada, Enero de 2009.
[12] Discusión Forums. *Subset Query not Subqueries*.
http://forums.oracle.com/forums/message.jspa?messageID=1133405#1133405
Visitado Noviembre de 2009.
[13] Oracle Corp., *Oracle DB Application Developer's Guide Object Relactional Features*, Oracle Corp., 2007.
[14] Kim W., *On Optimizing an SQL-like Nested Query*, ACM Transactions on Database Systems (TODS), v.7 n.3, 1982.
[15] Muralikrishna, M., *Improving Unnesting Algorithms for Join Aggregate Queries in SQL*, Proceedings of the VLDB Conference, 1992.
[16] Rao J., Ross K. A., *Reusing Invariants: A new Strategy for Correlated Queries*, Proc. of ACM SIGMOD Conf. on Management of Data,1998.
[17] Gilman L., Rose A. J., *Apl: An Interactive Approach*, John Wiley & Sons, 1984.