Crítica a los cursores de SQL y propuesta de su eliminación

Ing. Hugo Ryckeboer, Ing. Osvaldo Spositto, Ing. Alfonso Palomares
Departamento de Ingeniería e Investigaciones Tecnológicas,
Universidad Nacional de La Matanza,
Florencio Varela 1903
B1754JEC San Justo, Argentina
cursores@unlam.edu.ar

Abstract

Using an exhaustive analysis of possible program structures we show that SQL cursors are not necessary to compute tables. We highlight the incompatibility between cursors and the descriptive and functional syntax style. The use of cursors also destroys the potential parallelism in execution. We show how arbitrary restrictions in the structure of the SQL language bound the languages expressive power.

Key words: data bases, cursors, programming theory, ANSI-sql,

Resumen

A través de un análisis exhaustivo de estructuras de programas se muestra que los cursores de SQL son innecesarios para elaborar tablas. Se resalta la incompatibilidad entre los cursores y el estilo descriptivo y funcional de la sintaxis y que su uso destruye el potencial paralelismo en la ejecución. Se señalan restricciones arbitrarias a la generalidad que limitan el poder expresivo del lenguaje SQL.

Palabras claves: bases de datos, cursores, teoría de programación, ANSI-sql,

1. INTRODUCCIÓN

El lenguaje SQL no sólo administra y recupera datos almacenados sino que a través de cierta capacidad de cálculo y de especificación de procesos ofrece soluciones casi integrales, eliminando en muchos casos el uso de lenguajes de programación tradicionales. El hecho de especificar lo que se pretende y no el modo detallado de obtenerlo implica un salto conceptual y permite calificarlo con toda propiedad como uno de los así llamados lenguajes de cuarta generación.

Por otra parte el motor de ejecución puede introducir un alto grado de paralelismo en la operación, característica que las construcciones de los lenguajes de programación habituales no ofrecen o las que hay no son sencillas de usar. También a través de las subconsultas se incorpora a su concepción descriptiva un estilo funcional, que expresa lo que desea sin efectuar asignaciones intermedias.

Es difícil decir que es SQL en este momento, tomamos como referencia ese común denominador de todas las implantaciones, inspirado en el SQL92. Ese SQL no ofrece una solución total por insuficiencia en los recursos que provee. Por otra parte, ha incorporado un recurso, los cursores, que no armoniza con su paradigma fuertemente inclinado al paralelismo en la ejecución.

Muchos tutoriales [1,4] y suscriptores a listas dedicadas al SQL, desaconsejan el uso de cursores, sin aportar un fundamento.

En este trabajo pretendemos fundamentarlo, mostrar cómo evitarlos sistemáticamente, delimitando los pocos casos en los cuales su uso es actualmente inevitable y señalar las pocas construcciones que debieran agregarse al SQL para hacer de él un lenguaje completo, en el sentido de que sea capaz de expresar lo mismo que los lenguajes imperativos habituales. Si esto se hiciera se aumentaría simultáneamente la capacidad expresiva del lenguaje en su modalidad descriptiva y la eficiencia en la paralelización de las operaciones.

Finalmente señalaremos las líneas nuevas de investigación que hemos detectado como fruto de este análisis.

2. PROGRAMAS QUE OPERAN SOBRE TABLAS.

El paradigma de SQL está orientado a una visión relacional de los datos y de los resultados. Toda operación describe la elaboración de una tabla a partir de las tablas preexistentes. Como caso extremo se elabora una sola tupla y eventualmente a ésta conteniendo un solo campo.

En esa visión relacional se disponía de las operaciones básicas y algunas derivadas del álgebra relacional: selección, proyección y ensamble. Rápidamente se incorporó una capacidad de cálculo, expresiones entre componentes de columnas y operaciones sobre los elementos de una columna, con lo cual muchas necesidades podían ser resueltas íntegramente dentro del lenguaje SQL. Mucho se tardó en incorporar funciones definidas por el usuario, difícil de entender porqué, puesto que compilar/interpretar un trozo de programa no es mucho más complejo que hacerlo con una expresión arbitraria.

En este contexto, los cursores ofrecen al programador la posibilidad de recibir una tupla por vez y someterla a las operaciones que considera conveniente. Si esto ocurre dentro del contexto de un lenguaje de programación incorpora toda la capacidad expresiva de este último.

Consideramos que la serialización de las operaciones que introduce el cursor es perjudicial y nos proponemos poner en evidencia que existe un camino alternativo mucho más conforme al espíritu del SQL, conservando la predisposición para un elevado paralelismo y el estilo funcional propio del lenguaje.

Como método para demostrar la no necesidad de los cursores, hemos partido de una taxonomía de las posibles acciones que se pueden realizar sobre una tabla. Las hemos caracterizado al modo del paradigma imperativo. Su modo de operar sobre tablas serían las iteraciones. De ser traducidos literalmente hubieran introducido cursores. Luego ponemos en evidencia que la capacidad expresiva del SQL no los necesita. Después estudiamos las combinaciones de las acciones elementales y su traducción al SQL. Suponemos que el programador puede describir las tablas sobre las cuales opera recurriendo a las construcciones SQL que corresponden al álgebra relacional. A partir de esa tablas mostraremos cómo programaría un programador con mentalidad secuencial propia del paradigma imperativo. Mostraremos por un análisis exhaustivo de casos como es factible realizar siempre la conversión siempre y cuando el SQL reciba un enriquecimiento que oportunamente destacaremos. Versiones posteriores al SQL92 han recogido algunas de ellas y fabricantes de motores han tomado antes la iniciativa en ese sentido.

Al convertir programas iterativos a SQL no incluiremos consideraciones de eficiencia por cuanto consideramos que éstas se pueden delegar en los algoritmos de optimización de los motores de bases de datos. También en esta versión nos limitamos a servicios de consulta que si bien fabrican tablas finales o intermedias no modifican las originales. Las actualizaciones se pueden conceptualizar como construcción de tablas nuevas que modifican o sustituyen a las recibidas.

Programas que producen resultados visibles, en pantalla o impresos se pueden descomponer en la preparación del material en forma de una tabla seguido de un proceso que la exterioriza.

También se puede entender que se fabriquen tablas intermedias, por ejemplo una selección compleja puede elaborar primero las claves de los elementos de interés operando sobre los atributos que influyen en la selección y luego seleccionar a estos con todos sus atributos.

Es por ello que supondremos para asemejarnos a una consulta SQL que se parte de una tabla existente (tal vez una vista) y se culmina con otra, la cual, es utilizada en otra etapa o una vez maquillada, se exhibe. En una primera visión podríamos decir que disponemos de variables que almacenan tablas y cada proceso asigna una nueva tabla y finalmente mostraremos que esas tablas

intermedias no deben ser citadas explícitamente ya que SQL las puede incorporar a modo de subconsulta. Nuestro análisis comienza con una taxonomía de los procesos sobre tablas.

3. TAXONOMÍA DE LOS PROCESOS

El hecho de ser las tablas conjuntos restringe enormemente las operaciones que se puede hacer con uno de sus elementos. Toda acción que privilegia a algunos elementos respecto de otros rompe la homogeneidad intrínseca del conjunto y equivale a particionar ese conjunto, o sea esa tabla, en partes, de modo tal que en cada parte se respete la homogeneidad de los elementos.

Bajo esta óptica resaltan dos grandes categorías de proceso: Por un lado aquellos que calculan a partir de los atributos de un elemento atributos nuevos. Esta misma acción se puede repetir en cada elemento y por lo tanto preservan la homogeneidad en el procesamiento. Estos procesos los denominaremos *horizontales*.

Luego están los que elaboran un resultado representativo de todos sin interesar el orden en que se los suministre, o sea, que evalúan una función simétrica de los datos. Sin embargo merecen ser subdivididos por cuanto existen funciones que son generalización a conjuntos de operaciones binarias, el ejemplo más sencillo es la sumatoria. Estos cálculos hacen intervenir una sola vez a cada dato en cualquier orden y pueden computarse a partir de resultados parciales sobre subconjuntos del conjunto base,. A estas clase de procesos los denominaremos *verticales*, a los restantes, o sea , a los que implican múltiple operación sobre al menos una parte de los datos, *algorítmicos*. Estos últimos tienen apariencia de dar un trato desigual a los elementos. Esta desigualdad es sólo aparente. Con razonamientos matemáticos se puede demostrar que el resultado no depende de las decisiones arbitrarias por las cuales le dieron a ciertos elementos un mayor protagonismo que a otros. Si los protagonistas hubieran sido otros, el resultado final no hubiera cambiado.

3.1 Procesos horizontales

Teniendo a la vista un elemento (una tupla) puede decidir extenderla con campos adicionales. Estos cálculos hacen intervenir a los campos ya conocidos. También incluiremos aquí consultas con unicidad a otras tablas. En su forma más sencilla estos cálculos se expresan como expresiones arbitrarias con operadores propios del tipo de datos que intervienen en la misma. A nivel de SQL92 se dispone de alternativas con el CASE. No hay constructor de iteraciones, ni forma de tener una biblioteca de cálculos frecuentes. Hay empresas que proveen motores de bases que ya subsanaron esta limitación.

Dado el carácter de conjunto que tiene una tabla, no hay forma de distinguir entre sus elementos, por lo cual, lo que hace con uno de ellos lo debe hacer con todos por igual, esta afirmación no queda invalidada por el hecho de que el proceso repetido sobre todos los elementos contuviera alguna bifurcación. Por otra parte el cálculo hecho para una tupla no interfiere con el cálculo hecho para otra. Aquí se destaca el elevado paralelismo que puede haber en los procesos horizontales.

3.2 Procesos verticales

En estos utiliza los elementos para calcular un valor representativo de todo el conjunto. En cálculos sencillos esto se consigue inicializando previamente una variable la cual es actualizada con cada tupla procesada. Así por ejemplo la sumatoria de los valores de un campo se construye por sucesivas sumas en las cuales es argumento y destino la variable que contendrá al final la sumatoria buscada.

Este tipo de cálculo se presta para su distribución, así una sumatoria se puede elaborar a partir de dos sumatorias parciales. Esta característica no es casual, al tratarse de un conjunto, sólo tienen una clara semántica las operaciones simétricas respecto de sus operandos. Esa simetría surge de la conjunción de las propiedades asociativas y conmutativas, característica que tienen las funciones provistas: MAX, AVG… En su versión más sencilla, tal como está implantado en SQL92 interviene un solo atributo, pero no se ve motivo para imponer esta limitación. En operaciones más complejas

el resultado podría ser una tupla y no un valor aislado como sería un cálculo de ejes de inercia de un cuerpo descrito mediante un conjunto de masas posicionadas.

3.3 Procesos algorítmicos

Almacenar la totalidad de los valores de ciertos campos con un objetivo similar al anterior pero delegado en un algoritmo de procesamiento no lineal o al menos no secuencial de los valores. Dado la indiferencia del algoritmo frente al orden de los elementos en su primer ciclo recorre los elementos secuencialmente. Luego algunos valores son consultados reiteradamente. El mayor protagonismo que dan a algunos elementos es sólo aparente. Se puede demostrar que el resultado final no depende de cual haya sido el protagonista. Cuando se trata de algoritmos de orden los razonamientos usan preponderantemente la propiedad transitiva de las relaciones de orden. Un ejemplo más algebraico es la inversión de una matriz donde en cada paso se elige un elemento como "pivote".

Son ejemplos:

- El cálculo de la mediana de una columna que entrega uno o dos valores.
- El cálculo de una cápsula convexa procesa dos o más columnas. Devuelve una cantidad variable de los elementos recibidos, desde unos pocos hasta la totalidad.
- La inversión de una matriz entrega una tabla de igual orden y cardinalidad pero con un campo numérico cambiado.

4. ANÁLISIS DE LOS PROGRAMAS QUE OPERAN SOBRE TABLAS

En los próximos párrafos consideraremos con un alto grado de abstracción los procesos enumerados y sus combinaciones. Los detalles del cálculo que se estuviera realizando no son de interés en esta clasificación. Hablaremos de funciones que de tuplas calculan tuplas. Es usual que tales funciones se puedan describir componente a componente mediante funciones de tuplas en campos elementales. El querer conservar algún campo de la tupla recibido se reduce a agregar una componente más en la tupla resultado que se construya por copia de un campo de la tupla original. Usaremos letras minúsculas empezando con la "`f`" para referirnos a tales funciones.

Por ejemplo si escribimos:

```
SELECT   x as a, x * y + z as b
```

Estamos construyendo una tupla de dos componentes, `a` y `b`, a partir de una de tres componentes, `x`, `y` y `z`. Podríamos escribir `a, b ← f( x, y, z )` o, si no queremos entrar en tanto detalle, `s ← f( t )` siendo `s` y `t` nombres genéricos de tuplas.

Se puede observar en el ejemplo que la componente `a` es una trascripción de `x`

El SQL92 impone una restricción a las funciones que se pueden calcular de este modo, permitiendo utilizar en su elaboración solamente ciertas funciones y operaciones predefinidas y un mecanismo de selección de alternativas.

Los proveedores de motores ya descubrieron que esa limitación es arbitraria y perjudicial para la expresividad del lenguaje y permiten escribir funciones utilizando también ciclos.

Aún así, sigue habiendo una limitación sintáctica, por cuanto las funciones provistas no pueden aportar varios campos de una vez. Un ejemplo práctico de tal necesidad sería una función que cambie el sistema de coordenadas en un problema geométrico.

En lo que sigue usaremos las siguientes convenciones:

$\|$ es el operador de concatenación de tuplas

$\bowtie$ es el operador de ensamble, llamado también de junta, ("`JOIN`")

Suponemos que las operaciones del álgebra relacional: selección, proyección, producto cartesiano, ensambles,… son operaciones disponibles en el motor de base de datos. Para no introducir más símbolos consideraremos el producto cartesiano como caso extremo del ensamble.

Con la idea amplia de función explicitado más arriba, la proyección se reduce a una función que consista en escribir sólo los nombres de los atributos que sobreviven.

La selección la consideramos una operación fundamental para evitar el envío de volúmenes demasiado grandes hacia el programa. Bien utilizada evitará el uso de selecciones dentro de la aplicación. Para que estas selecciones sean efectivas y evitar pasos intermedios es conveniente tener la misma capacidad de usar funciones arbitrarias en el `WHERE` como en el `SELECT`. Ambos remiten en su sintaxis a `<value expression>`, asegurando por lo tanto la igual capacidad expresiva.

El ensamble, simbolizado con $\bowtie$ es una operación si bien derivada fundamental en cuanto al poder expresivo, la cual por otra parte no hace más que revertir las proyecciones hechas a partir de una relación universal. El ensamble aumenta el tráfico de datos entre servidor SQL y cliente. Es de notar que un programador de tuplas en un lenguaje imperativo evita transferencias repetidas de partes comunes de las tuplas efectuando el ensamble en la memoria principal ocupada por el programa. Esto lo señalaremos nuevamente más adelante.

El lenguaje SQL provee operaciones de conjuntos. Las señalaremos genéricamente con mayúsculas `F`; `G`,… Aquí también señalamos una pobreza del lenguaje SQL. El usuario debiera ser capaz de definir nuevas funciones de conjunto e inclusive permitir algunas de resultado no atómico o que operen sobre varias columnas al mismo tiempo (por ejemplo cálculo de regresiones)

Nos imaginamos una sesión de SQL como una sucesión de acciones que de tablas engendran una nueva tabla. Sus variables, para almacenar resultados intermedios son del tipo tabla. No nos interesa si dos tablas tienen parte de su cálculo en común, lo consideramos problema del optimizador del motor.

En los esquemas siguientes ilustramos el modo de operar de un programador en un lenguaje imperativo que quisiera construir una tabla a partir de otra. Introducimos un iterador no común en los lenguajes de programación: `foreach.` Con el operador `foreach` queremos resaltar la indiferencia frente al orden de evaluación. Justamente esto es lo que provee un proceso con cursores, acceso a un elemento por vez en un orden arbitrario. Que la tabla o vista que alimenta al cursor entregara los elementos en un orden particular no tiene ninguna importancia para quien quiera construir otra tabla, solamente serviría para una exhibición más inteligible de la misma.

Comenzamos el análisis con los procesos elementales para luego estudiar combinaciones de los mismos. Si llamamos **D** a la tabla dato y **R** a la tabla resultado, tendríamos las siguientes estructuras para los distintos procesos

4.1 Procesos elementales

Usamos la taxonomía del punto3 y damos sus equivalentes en SQL sin utilizar cursores.

4.1.1 Proceso de tipo horizontal

```
R ← ∅
foreach t of D do
    s ← f( t )
    agregar s a R
```

```
SELECT  f( t )
FROM  D
```

Esto admite una traducción directa a SQL si no tuviera limitación en las funciones.:

La `f( t )` se traduce tal como indicamos más arriba. Si `R` no es la pantalla falta una cláusula `INTO R` .Recuérdese que dijimos que `D` puede ser fruto de una expresión del álgebra relacional y por lo tanto constar de la cita de varias tablas e incluir cláusulas `WHERE` y `JOIN` o inclusive ser una subconsulta.

4.1.2 Proceso de tipo vertical

```
Inicializar r
foreach t of D do
    r ← f( t, r )
```

El resultado R se reduce aquí a una única tupla r.

Esto admite en muchos casos una traducción directa a SQL:

```
SELECT  F( t )
FROM  D
```

Aquí tenemos la principal limitación del SQL, el repertorio de las F está delimitado en la definición de lenguaje y no hay un mecanismo de definición de nuevas instancias. Esta pobreza se ha paliado agregando en algunas implantaciones nuevas funciones predefinidas, pero éste no es el camino correcto, debiera haber un mecanismo de extensión. Este es el principal defecto en el diseño del lenguaje. Es notable como en las versiones más modernas, incluyendo la versión de trabajo del SQL-2003 [3], se persiste en el mismo defecto, habiendo puesto como remedio un surtido más grande de funciones predefinidas.

La pobreza de funciones de conjuntos es la principal causa del uso actual de cursores.

Después de este diagnóstico, y para no reiterarlo, continuaremos el análisis suponiendo que tenemos un mecanismo para definir nuevas funciones y que esta limitación está superada.

4.1.3 Proceso de tipo algorítmico

```
n ← 0 ;
foreach t of D do
    n ← n + 1
    v[n] ← t
R ← Algoritmo( v, n )
```

SQL no ofrece una solución genérica. Tampoco los cursores. Hay versiones de SQL que permiten depositar el contenido íntegro de una tabla en una variable de tipo tabla. Después se debiera llamar a una función o procedimiento ajeno al SQL para que elabore un algoritmo.

El primer inconveniente de este modo de operar es la rotura de la estructura funcional.

Los algoritmos que apartamos en esta tercera variante consultan el vector v en una forma no secuencial. El cálculo de mediana es un ejemplo, entrega una o dos tuplas de un sólo campo sobre un R que consta de un sólo campo.

Aunque hay métodos lineales, éstos tienen coeficiente mayor que 1, lo que significa que algunos elementos son consultados varias veces como consecuencia de la evaluación parcial ya hecho además de ser desplazados. El direccionamiento directo es fundamental para lograrlo. Estos problemas no se encaran con cursores sino tal como muestra el esquema con ayuda de un vuelco en memoria de la totalidad de la información disponible. El depósito masivo de toda una tabla es mucho más eficiente que llamar sucesivamente a la función FETCH NEXT del cursor con el único propósito de depositarla en un arreglo. Ya se hizo notar que estos algoritmos en su primera fase hacen un barrido secuencial de los datos y recién en sus posteriores accesos actúan selectivamente.

La solución ideal sería que SQL permitiera incorporar funciones que de tablas den tablas. No queda tan clara la utilidad cuando hay un único cálculo algorítmico, pero cuando este sea repetitivo, su incorporación al lenguaje permitiría distribuir las diferentes ejecuciones. Fuera del cálculo de mediana no es frecuente este tipo de situaciones entre los usuarios del SQL.

4.2. Parametrización de Procesos

Es factible que estos procesos dependan de un parámetro. Esto se aplica particularmente cuando un proceso está anidado dentro de otro. El contenedor puede, al recorrer su propia tabla, enviar un parámetro al contenido

En los procesos horizontales el parámetro puede intervenir en uno o ambos modos siguientes:

(a) La especificación de D (b) El modo de efectuar el cómputo de s.

El parámetro estará simbolizado con p.

En el primer uso, el parámetro especifica el conjunto particular sobre el cual se quiere operar. Interpretado el parámetro cómo especificador de una tabla,

4.2.1 Proceso horizontal parametrizado

```
R ← ∅
foreach t of D( p )
do
  s ← f( t )
  grabar s en R
```

```
R ← ∅
foreach t of D do
  s ← f( t, p )
  grabar s en R
```

También se puede dar combinado.

4.2.2 Proceso vertical parametrizado

En los procesos de tipo vertical el parámetro puede intervenir solamente en la especificación de D

```
Inicializar r
foreach t of D( p ) do
  r ← f( t, r )
```

Cuesta imaginar que intervenga en el cómputo. Por ejemplo ¿Cómo sería una sumatoria parametrizada (y no en la cantidad de sumandos)? De todos modos no molestaría que lo hiciera.

4.2.3 Proceso algorítmico parametrizado

Aquí podríamos imaginar un proceso como el cálculo de percentiles que admite el valor de éste como parámetro. Pero un algoritmo que los calculara simultáneamente sería siempre más eficiente que reiterados cálculos de valores únicos.

La especificación del conjunto de datos sobre el cual correr el algoritmo es otro uso del parámetro.

```
n ← 0 ;
foreach t of D( p ) do
    n ← n + 1
    v[n] ← t
R ← Algoritmo( v, n, p )
```

4.3 Comentario sobre la trama de los procesos

Es de notar que hablando con esta generalidad en estos procesos elementales no hay instrucciones de selección, esto se puede razonar en detalle, por ejemplo para un proceso de tipo horizontal.

```
foreach t of D do
    s ← f( t )
    grabar s en R
```

Toda selección que afecta al cómputo de s es parte integrante de lo que genéricamente hemos llamado "f". Esto a lo más indica que debe haber un lenguaje algorítmico para describir estos cómputos. Una selección que influya en que s exista o en la cantidad de campos hace carente de sentido o imposible la existencia de un resultado para agregar a R.

Una selección que haga condicional la grabación hace también condicional el cálculo de s pues no tiene sentido calcular algo que no se piensa almacenar.

Finalmente si ambas líneas son condicionales resulta que la tabla D de partida tuvo que ser mejor descripta con una cláusula WHERE mejor especificada para que no aparezcan en ella tuplas no deseadas. El SQL es ortogonal en su <search condition>, de modo tal que lo que se puede especificar en una bifurcación también se puede especificar en un WHERE.

Sólo al componer procesos aparecen las acciones condicionadas. La selección aparece cuando el resultado de una consulta influye en que se realice o no la segunda consulta.

4.4 Anidamiento de procesos en ciclos

Un primer modo de anidamiento es tener un ciclo `foreach` dentro de otro. Si los valores del ciclo exterior no parametrizan a la tabla del ciclo interior se trata de un producto cartesiano de tablas, caso extremo del ensamble. Si por el contrario, si lo parametrizan se trata de un ensamble. El orden de los dos constructores de ciclos se podría invertir, adecuando la descripción de tabla parametrizada sin cambiar el resultado. Si se lo hace con un "`JOIN … ON …`" la condición expresada no cambia.

```
foreach t1 of D1 do
    s1 ← f1( t1 )
    foreach t2 of D2(s1) do
```

equivale pues a

```
foreach t of D1 ⋈ D2 do
```

Al poder parametrizar los procesos elementales arriba detallados resulta factible encerrar a cualquiera de ellos en un ciclo que provea sucesivos valores de parámetro para los mismos.

Esta y otras igualdades expresivas presuponen que toda función de la cual disponga el sistema se pueda usar tanto en la claúsula `SELECT` como en la cláusula `WHERE`.

4.4.1 Ciclos sobre procesos horizontales

Un ciclo que abarca un proceso horizontal produciría en cada iteración una tabla. Esto es incompatible con el planteo original de que el resultado sea una única tabla. El razonamiento de homogeneidad, en este caso aplicado al parámetro, deja pocas opciones: se debe reunir tablas en cantidad variable en una sola y que ello no dependa del orden de evaluación.

Operaciones asociativas y conmutativas entre conjuntos son la unión, intersección y diferencia simétrica. La última es de dudosa semántica aplicada a tuplas.

```
foreach t1 of D1 do
    R ← ∅
    foreach t2 of D2(t1) do
        s ← f1( t1 , t2 )
        agregar s‖t1‖t2  a R
    Aqui debo indicar que hago con cada
R
```

Si se trata de la unión basta con adelantar la sentencia `R ← ∅` un lugar, y la unión en el sentido que le da SQL es automática. Pero una vez sacada esta instrucción tenemos dos `foreach` anidados que según lo visto en 4.4 se confunden en 1. Si se trata de la intersección se está efectuando una división y esta es expresable en el álgebra relacional. Lo más general sería conservar los valores junto con los valores de los parámetros que le dieron lugar, comportamiento similar al de la unión.

4.4.2 Ciclos sobre procesos verticales

El planteo de partida es similar

```
foreach t1 of D1 do
    Inicializo r
    foreach t2 of D2(s1) do
        r ← f1( r, t1 , t2 )
    Aqui debo indicar que hago con cada
r
```

Valen las conclusiones anteriores si `r` se considera una relación de una sola tupla. El hecho de ser un escalar abre la posibilidad de efectuar con todos ellos un proceso vertical, caso que estudiaremos más adelante. Con la primera alternativa caemos en la construcción que en SQL usaría la cláusula `GROUP BY:`

```
R ← ∅
foreach t1 of D1 do
    Inicializo r
    foreach t2 of D2(t1) do
        r ← f1( r, t1 , t2 )
    guardar t1‖r en R
```

se traduce como

```
SELECT t1, F1( t1, t2 )
FROM  D1 JOIN D2 ON … t1 …
GROUP BY t1
```

4.4.2 Ciclos sobre procesos algorítmicos

Siguiendo los planteos anteriores:

```
foreach t1 of D1 do
    n ← 0 ;
    foreach t2 of D2( t1 ) do
        n ← n + 1
        v[n] ← t2
    R ← Algoritmo( v, n, p )
    Aqui debo indicar que hago con cada
R
```

Tiene una problemática similar a la de los procesos horizontales, se dispone de múltiples tablas y es necesario fijar un modo de reunirlos en una única tabla. Lo que si desaparece es la idea de reunir los dos ciclos, puesto que el ciclo interior no sería adyacente sintácticamente al exterior sino quedaría separado por la llamada a función que computa al algoritmo.

```
SELECT
FROM D1 JOIN Algoritmo( SELECT D2.t2 FROM D2 WHERE   D1.t1   )
```

En cambio con un proceso horizontal hubiéramos tenido:

```
SELECT
FROM  D1  JOIN  ( SELECT D2.t2  FROM D2 WHERE    D1.t1   )
```

Está última puede reescribirse sin el uso del segundo SELECT cómo ya hemos indicado.

Con cualquiera de estos procesos si en su realización dependieran de un parámetro, y los cómputos se ejecutan para varios valores del mismo, se crearía una tabla con un campo adicional dedicado al parámetro, el cual integrará la clave.

Actualmente los programadores depositan las tablas a ser procesadas algorítmicamente. La regularidad de las representaciones en memoria principal hace que esta área se pueda considerar un vector y a eso hemos aludido en nuestros esquemas. No molestaría que fueran listas u otro tipo de estructura.

El hecho de necesitar un almacenamiento para empalmar con un algoritmo escrito en un lenguaje procedural destruye el carácter funcional de las consultas. Lo que se necesita son funciones que de tablas den tablas.

4.5 Anidamiento de procesos en procesos

Los procesos que hemos considerado se esquematizan con un ciclo, a diferencia del caso anterior estos ciclos ya tienen un objetivo y lo que debemos destacar como aporte novedoso es de qué manera puede contribuir el ciclo interior al exterior.

4.5.1 Proceso horizontal dentro de otro horizontal

En un esquema que opera registro a registro corresponde que el ciclo interior colabore con el exterior en cada registro que elabora. El único modo es entregarlo tal como está para que sea incorporado sin más en el ciclo exterior o enriquecido por éste con elementos que el ciclo interior no elabora.

Arrancando desde una óptica distinta llegamos exactamente a lo que se hacía en el punto 4.4.1.

4.5.2 Proceso vertical dentro de otro horizontal

Aquí corresponde hacer una distinción: este proceso vertical se realiza antes o después del proceso horizontal. Expresamos esquemáticamente las dos posibilidades:

```
R ← ∅
foreach t1 of D1 do
    s1 ← f1( t1 )
    Inicializar r
    foreach t2 of D2(s1) do
        r ← f2( r, t2 )
    agregar s1‖r a R
```

```
R ← ∅
foreach t1 of D1 do
    Inicializar r
    foreach t2 of D2(t1) do
        r ← f2( r, t2 )
    s1 ← f1( t1, r )
    agregar s1‖r a R
```

Si se realiza antes (esquema de la izquierda) llegamos a un resultado similar al alcanzado precedentemente, existen otros campos ya calculados al cual se anexa. Si se realiza después puede llevar a múltiples citas de un mismo cómputo lo cual el optimizador lo evitará con una subconsulta

```
SELECT s1, F2( t2 )
FROM D1 JOIN D2 ON …f1( t1 )…
GROUP BY s1
```

```
SELECT f1( t1, r ), r
FROM ( SELECT D1.*, F2( t1 ) AS
r
        FROM D1 JOIN D2 ON …
        GROUP BY t1 )
```

4.5.3 Proceso algorítmico dentro de otro horizontal

Nuevamente nos encontraremos que no hay forma de expresar la colaboración de cada proceso algorítmico. Diagnosticado con mayor precisión podemos afirmar que SQL permite construir tablas de complejidad arbitrariamente grande mientras sepamos explicar como construirla registro a registro, pues eso es lo que figura dentro de un SELECT. Todos los casos en que una tabla se construye por unión de otras en número variable el lenguaje sólo puede expresarlo por un barrido con cursores incorporando los aportes de una tabla por vez. Tablas construídas mediante algoritmos no secuenciales sobre los datos se comportan como tablas almacenadas en el sistema.

Con este diagnóstico global dejamos de incluir los procesos algorítmicos dentro de este análisis.

4.5.4 Proceso horizontal dentro de proceso vertical

```
Inicializar r
foreach t1 of D1 do
    R ← ∅
    foreach t2 of D2(t1) do
        s ← f( t1, t2 )
        Grabar s‖t1‖t2 en R
    Aqui debo indicar como cada R incide
en r
```

Y nos encontramos con el problema propio de todos los anidamientos de procesos horizontales en otros. Si tuplas de todas las R inciden del mismo modo no hubiera sido necesario la subdivisión y se trata de un único proceso vertical sobre una tabla ensamblada, si las tuplas de un mismo R inciden de un mismo modo, se puede tratar como un proceso vertical dentro de otro vertical, o simplemente expresarlo en la fórmula con la cual la t actualizan la r. si todo R tiene una única

incidencia en r entonces seguro que son procesos verticales dentro de otro vertical. Estos los veremos en el punto siguiente.

4.5.5 Proceso vertical dentro de procesos verticales

```
Inicializar r
foreach t1 of D1 do
    Inicializar s
    foreach t2 of D2(t1) do
        s ← f1( s, t1, t2 )
    r ← f2( r, s, t1 )
```

Esta combinación se puede traducir con una subconsulta que fabrique los pares s, t_1 que sirven para evaluar r.

```
SELECT  F2( s, t )
FROM  ( SELECT F1( D1.t1, D2.t2 ) s, D2.t1 .t
        FROM D1 JOIN  D2 ON ... D1.t1 ...
        GROUP BY D1.t1 )
```

Este esquema de reducción se puede utilizar para procesos verticales más anidados. Las sucesivas subconsultas anidadas tendrán cada vez más ensambles y listas de agrupamiento más extensas.

4.5.6 Procesos condicionados por el resultado de otros

Aunque dentro de un proceso elemental no hay sentencias condicionales, es factible que un proceso completo quede subordinado al resultado de otro. Esto es de particular interés cuando el subordinante y el subordinados son interiores a un ciclo tanto puro como gobernando un proceso. A su vez puede manejarse la condición por un valor, fruto de un cálculo vertical como por una tabla, fruto de un cálculo horizontal. Los procesos algorítmico pueden producir tanto uno u otro.

Empezando por las condiciones gobernadas por un valor:

```
R ← ∅
foreach t1 of D1 do
    Inicializar r1
    foreach t2 of D2( t1 ) do
        r1 ← f1( r1, t1, t2 )
    if  φ( r1 )
        Inicializar r2
        foreach t3 of D3( t1 , r1 )
do
            r2 ← f2( r2, t1, t3, r1 )
        agregar t1‖r1‖r2 en R
```

Por supuesto que muchos parámetros que hemos agregado como posibles en los casos concretos no aparecen.

Un modo sistemático de reducirlo es calcular primero una relación de los pares (t_1 , r_1) para los cuales debe realizarse el segundo cálculo y usar este como tabla de entrada del segundo proceso.

```
foreach t1 of D1 do
    Inicializar r1
    foreach t2 of D2( t1 ) do
        r1 ← f1( r1, t1, t2 )
    if  φ( r1 )
        grabar t1‖r1 en R1
R ← ∅
foreach t1‖r1 of R1 do
    Inicializar r2
    foreach t3 of D3( t1 , r1 ) do
        r2 ← f2( r2, t1, t3, r1 )
```

```
agregar t1‖r1‖r2 en R
```

En SQL es muy sencillo arrancar un cálculo a partir de una resultado intermedio, basta con escribir a éste como una subconsulta en el FROM

```
SELECT aux.*, F2( aux.*, D3.t3.)
FROM ( SELECT D1.*, F1( D1.t1, D2.t2 )
       FROM D1 JOIN D2 ON ... .D1.t1 ...
       GROUP BY D1.t1
       HAVING φ( r1 )  ) aux  JOIN D3 ON  aux.t1  aux.r
```

Si la condición es gobernada por una tabla, cabe preguntarse que se puede preguntar sobre ella, cualquier propiedad que se traduzca en un sí o no puede transformarse en un valor (proceso vertical) que lo represente. Así preguntar si una tabla es vacía equivale a preguntar si la cardinalidad es 0. Sin embargo como la pregunta por vacío existe en SQL bajo la modalidad EXIST podemos analizarla un momento. Otras preguntas vinculadas a la naturaleza de conjunto que tienen las tablas se pueden reducir a esta pregunta, así preguntar si una está contenida en otra equivale a preguntar si la resta es vacía. Veamos un esquema genérico basado en conjunto vacío

```
…
foreach t of D1 do
    Proceso horizontal H que
determi
     na un R a partir de un D2( t
)
    if  R > ∅
        un proceso P contribuye al
          resultado buscado
```

Determinamos los valores de t1 para los cuales se quiere realizar el segundo proceso, seguido de otro ciclo donde efectivamente se computan

```
aux ← ∅
foreach t of D1 do
    Proceso H que determina un R a partir de un D2( t )
    if  R > ∅
        grabar t en aux
…
foreach t of aux do
    un proceso P contribuye al resultado buscado
```

El ensamble de D_1 y el proceso que construye R no contiene registros si R es vacío, tomando su proyección sobre los elementos de D_1 se tiene un resultado que permite saber con un IN si un cierto t debe ser procesado o no en la segunda parte. El proceso H no aporta nada al resultado ya que en la práctica no recibe una evaluación completa.

```
SELECT ...
FROM  D1
WHERE  t  IN  ( SELECT ... )
```

Si la pregunta fuera por vacío se puede negar la condición "IN ..."

4.5.7 Múltiples procesos dentro de otro

Esta situación se puede equiparar a lo estudiado en *4.4.6* considerando que el primer proceso tomó la decisión de que todos los argumentos eran válidos. Observen que con la máxima generalidad consideramos que el resultado del primer proceso no sólo había servido para tomar la decisión sino que estaba a disposición del segundo proceso.

4.5.8 Reglas generales de reducción

Para documentar mejor estos esquemas abstractos hemos elaborado ejemplos concretos.[2] De este análisis exhaustivo de casos podemos establecer una estrategia general de reducción de procesos anidados

1) Secuencias de procesos dentro de un ciclo, sean de ejecución condicionada o no, se resuelven por una secuencia de ciclos con el mismo control ejecutando un solo proceso interior por vez.
2) Múltiples anidamientos de procesos horizontales, que contribuyen a una única relación objetivo, se deben encarar por unión de múltiples procesos con un solo proceso horizontal anidado.
3) Un proceso horizontal anidado pierde su anidamiento si se lo programa sobre un ensamble.
4) Un proceso vertical anidado se elimina mediante una subconsulta.
5) Una pregunta que condiciona una elaboración al resultado de otra se resuelve mediante una subdivisión del proceso en dos, en una tabla auxiliar se elaboran los elementos habilitados para el segundo proceso. Finalmente ambos se pueden unir sintácicamente recurriendo a la construcción `IN` evitando el agregar momentáneamente una tabla auxiliar al catálogo.

5. CUANDO EL USO DE CURSORES ES INEVITABLE

Hemos detectado un solo caso en el cual los cursores son inevitables y ello corresponde a una situación completamente distinta a la analizada. Nuestro análisis estudió las elaboraciones que comienzan con una vista, armada tal vez con ayuda de varias tablas, a partir de la cual se quiere computar una nueva tabla.

En toda vista la cantidad de tablas que intervienen es perfectamente conocida por quien la escribe. ¿Qué pasa si las tablas sobre las cuales quiero actuar las describo por alguna característica? Podría no saber a priori, o no querer contabilizarlo, cuantas aparecerán. Esto es factible recurriendo a la meta-información almacenada en la misma base.

Obsérvese que el argumento de un `FROM` es un conjunto concreto de tablas, aún recurriendo a subconsultas. Se puede ampliar la cantidad pero quién la escribe sabe a cuantas tablas afecta su consulta. También la `UNION` cita una cantidad concreta. Podríamos decir que las tablas intervienen como constantes. No creemos que tenga sentido mantener el recurso cursores para esta única finalidad. Los ejemplos que hemos encontrado son acciones de unión entre contenidos de campos homónimos de muchas tablas y sólo durante tareas de remodelación de bases de datos. Aún usado en esta situación los cursores dispararían la acción sobre cada tabla en secuencia frenando una posible ejecución paralela si las tablas involucradas residieran en más de un procesador.

6. RESUMEN, CONCLUSIONES Y EXTENSIONES

A lo largo de este análisis hemos señalado los aspectos en los cuales debiera ser enriquecido el SQL92, punto de referencia para muchos motores. Estos son:

- Proveer funciones sin ninguna limitación algorítmica. Éstas en tres en modalidades, todas ellas capaces de entregar tuplas con varios atributos.
    - A partir de los campos de una tupla para elaborar nuevos campos de esa tupla.
    - A partir de una o más columnas de una tabla elaborar nuevos campos con funciones simétricas.
    - A partir de una tabla engendrar otra tabla o una tupla en forma algorímica.
- Extender el segundo mecanismo detallado en el punto anterior a operadores solo asociativos a ser utilizados en contextos de ORDER BY.
- Agregar un mecanismo para uniones genéricas de tablas.

Todas las características enunciadas aumentarían simultáneamente la expresividad del lenguaje y el paralelismo en la ejecución y permitirían conservar la característica descriptiva y funcional para

cada consulta. Una vez hecha esas mejoras el uso de cursores, que destruye las características de lenguaje, se vuelve superfluo.

A partir de este trabajo hemos abierto varias líneas de trabajo adicionales:

- Este trabajo debiera ser extendido para empalmar con versiones más modernas de SQL (1999 y 2003), aún no plena y masivamente implantadas.
- También se intentará concretar una propuesta concreta de modificación del lenguaje analizando las modificaciones gramaticales necesarias.
- La experiencia docente sugiere que la redacción de sentencias SQL complejas no resulta intuitiva. De los esquemas de reducción discutidas surge la posibilidad de ofrecer un lenguaje alternativo que permita describir con mentalidad imperativa lo deseado y lo traduzca al SQL facilitando el uso de los motores existentes.
- En varios casos hemos encontrado formas alternativas de expresar un requerimiento en SQL, no así en el sentido inverso, donde la expresión imperativa admite menos alternativas. Esto sugiere que alrededor de ella se podría tener una forma normal interna, punto de partida para encarar optimizaciones en la evaluación, de modo tal que el estilo de redacción de la consulta no influyera en la eficiencia de la misma.

AGRADECIMIENTOS

Al Ing. Jorge Doorn por haber leído y hechos valiosas sugerencias para este artículo.

REFERENCIAS

[1] Chigrik, Alexander "*Using SQL Server Cursors"* www.mssqlcity.com/Articles/General/UseCursor.htm

[2] Grupo Cursores "*Informe técnico N°2*" del en Depto. Ingeniería Univ. Nac. de la Matanza.

[3] Melton Jim (ed.) "*(ISO-ANSI Working Draft) Foundation*", August 2003, pág. 505

[4] Thayer, Matías "*Cursores en SQL Server*" http://maestrosdelweb.com/editorial/cursql