Complete los valores de profundidad del directorio y las profundidades locales.

iii. A continuación se enumeran una serie de operaciones sobre el archivo, efectuadas, cada una de ellas, con el archivo en el estado representado en el punto ii. y una serie de posibles consecuencias. Indique qué consecuencias implica cada una de estas operaciones.
Operaciones: insertar el registro con clave
a) 293 b) 166 c) 35 d) 110 e) 49.

Consecuencias:

1. Duplica la cantidad de entradas del directorio.
2. Duplica la profundidad del directorio.
3. Reduce la profundidad del directorio.
4. Produce modificaciones en el directorio, sin cambiar de tamaño.
5. No produce ningún cambio en el directorio.
6. No puede ser satisfecha. (Sugiera una forma de modificarlo a fin de que pueda ser satisfecha.)
7. Lleva la profundidad del directorio a 5.
8. Entre las consecuencias del punto anterior hay una que es "imposible" (no existe ninguna operación que al aplicarse sobre este archivo produzca esa consecuencia). ¿Cuál es? Dar los motivos.

NOTAS BIBLIOGRÁFICAS

El texto de T. J. Teorey [198] trata a fondo los problemas de selección de estructuras físicas para implementar bases de datos. También contiene información útil sobre este tema, y sobre bases de datos en general, el libro de G. Wiederhold [213] (hay traducción al español). Sobre estructuras de datos en general existen una amplia bibliografía, de la cual destacamos la magnífica obra de D. E. Knuth [141].

El artículo original sobre el árbol-B es el de Bayer-McCreight [18]. Otros artículos interesantes son el de Comer [69] y el de Held-Stonebraker [127].

En el caso de hashing extensible, el artículo original es el de Fagin y otros [87].

CAPÍTULO 7
PROCESAMIENTO DE CONSULTAS

7.1. INTRODUCCIÓN

Dada una consulta en un lenguaje como SQL, el sistema de base de datos debe de alguna manera generar las tuplas de la respuesta. De la eficacia de los métodos usados en esta tarea dependerá en gran medida la performance total del sistema. Un lenguaje de consulta relacional da al usuario el poder de especificar en forma sucinta consultas cuya implementación puede ser muy costosa a menos que el sistema elija cuidadosamente el método más eficiente para procesarlas. La diferencia en costo entre un buen y un mal método puede ser de varios órdenes de magnitud.

EJEMPLO 7.1: Consideremos el esquema relacional del Capítulo 5, donde teníamos:

Alumno (Anúmero, Anombre)
Examen (Anúmero, Materia, Efecha, Nota).

Supongamos que debemos procesar la consulta "hallar los nombres de los alumnos que sacaron más de 8 en algún examen", o sea, en SQL:

```
select Anombre
from Alumno, Examen
where Alumno.Anumero = Examen.Anumero
  and Nota > 8;
```

Traduciendo esta consulta al álgebra relacional, obtenemos

$$\pi_{Anombre}(\sigma_{Nota>8}(\sigma_{J=3}(Alumno \times Examen)))$$

Supongamos que existen 5000 tuplas de Alumno, almacenadas a 20 por bloque físico, y 100.000 tuplas de Examen, almacenadas a 10 por bloque. El método ingenuo de evaluar esta expresión algebraica consiste en formar el producto cartesiano *Alumno* × *Examen* y luego aplicarle las selecciones y la proyección. El cómputo del producto requiere leer 250 × 10.000 = 2.500.000 bloques y escribir otros tantos, generando un resultado intermedio de 2.500.000 bloques que luego debe ser barrido para seleccionar las tuplas relevantes; el costo total en cantidad de accesos a bloque resulta de alrededor de 7.500.000.

Una estrategia más sofisticada observará que el producto cartesiano seguido de la selección es en realidad una operación de junta, que puede realizarse en forma mucho más eficiente. Supongamos que una relación de n bloques puede ser puesta en orden de acuerdo con el valor de un atributo con costo $n\log_2 n$. Ordenamos las relaciones Alumno y Examen de acuerdo con el valor de Anúmero; el costo del ordenamiento es de 250log250 + 10.000log10000 aproximadamente igual a 140.000. Ahora podemos implementar la junta barriendo secuencialmente ambas relaciones en forma similar a una operación de "merge" de archivos; cuando encontramos un par de tuplas que concuerdan en número de alumno y satisfacen la condición de que la nota sea mayor que 8, emitimos el nombre del alumno. Suponiendo que la cuarta parte de los alumnos van a satisfacer la condición, el costo de este paso es de 10.250 accesos de lectura y 50 accesos de escritura. El costo total resulta así de alrededor de 150.000 accesos, 50 veces menor que el costo del método ingenuo. Notemos que el costo se podría haber reducido aun más si hubiéramos combinado el ordenamiento de la relación *Examen* con la selección de las tuplas con nota mayor que 8; si suponemos que un décimo de las tuplas satisfacen esta condición, el resultado de este paso es una relación *Examen* con sólo 1000 bloques en lugar de 10.000, y el costo total se reduce a 141.000 accesos.

Costos de Procesamiento

El objetivo del procesamiento de consultas es la minimización del costo de la producción de una respuesta. El costo a minimizar se compone de distintos factores, entre ellos los siguientes:

- **Comunicaciones**: El costo de transmitir datos desde donde estén almacenados hasta donde se realice el cómputo y presentación de los resultados. Por ejemplo, en una base de datos geográficamente distribuida, el procesador de consultas tratará de usar en lo posible datos almacenados localmente en el lugar donde se hace la consulta y minimizar la cantidad de datos que deben ser transferidos mediante las líneas de comunicación. Como en este libro no trataremos los problemas de bases de datos distribuidas, ignoraremos de ahora en más los costos de comunicaciones.
- **Acceso a memoria secundaria**: El costo de transferir bloques de datos de memoria secundaria a memoria principal y viceversa. Los factores que influyen sobre este costo son: cantidad de tuplas a ser transferidas, agrupamiento de tuplas en bloques físicos, cantidad de espacio de buffers disponible en memoria principal, y velocidad de acceso de los dispositivos de memoria secundaria.
- **Tiempo de Procesador**: El costo de utilizar la unidad central de procesamiento.

Como ya hemos indicado, la tecnología actual implica que en la mayoría de los casos –pero no en todos– el tiempo de acceso a memoria secundaria dominará el tiempo de uso del procesador; por lo tanto, los sistemas de base de datos consideran ventajoso gastar tiempo de procesador en la búsqueda de estrategias que disminuyan la cantidad de bloques a acceder.

Etapas en el Procesamiento de Consultas

El primer paso en la ejecución de una consulta es su traducción a una forma interna que sea fácilmente manipulable. Por ejemplo, es común transformar una consulta en una expresión en álgebra o cálculo relacionales, representadas mediante un árbol de sintaxis. Las técnicas para realizar esta transformación son similares a las que utiliza el analizador sintáctico de un compilador. A la vez, el analizador verifica la corrección de la sintaxis de la consulta, valida los nombres de relaciones y atributos utilizados, etc.

Una vez realizado el análisis sintáctico, comienza el proceso llamado *optimización de consultas*. Éste se compone de dos fases: en la primera fase, se intenta transformar la forma interna de la consulta en otra equivalente pero menos costosa de ejecutar. Por ejemplo, dada una expresión en álgebra rela-

cional, se trata de transformarla en otra expresión equivalente que sea más fácilmente computable. Esta primera fase es independiente de las estructuras físicas de almacenamiento utilizadas. La segunda fase toma el resultado de la primera y genera, tomando en cuenta las estructuras físicas tales como índices y tablas de hashing, una estrategia detallada para la ejecución de la consulta; esta estrategia suele llamarse un *plan de acceso*.

7.2. TRANSFORMACIÓN DE EXPRESIONES ALGEBRAICAS

Como sabemos, una misma consulta puede expresarse en muchas formas distintas en álgebra relacional; algunas de éstas pueden ser substancialmente más fáciles de ejecutar que otras. El objetivo de la primera fase del proceso de optimización de consultas es la selección de una expresión algebraica equivalente a la dada y eficientemente computable. En esta sección presentaremos reglas que pueden usarse en este proceso de transformación.

Selección

Como regla general, trataremos de aplicar los operadores de selección tan pronto como sea posible; esto reducirá el tamaño de los resultados intermedios.

EJEMPLO 7.2: Con el mismo esquema del ejemplo anterior, consideremos la consulta: "¿qué alumnos obtuvieron más de 8 en un examen tomado por el Prof. Sandrini?". La expresión algebraica

$$\pi_1(\sigma_{Nota>8}(\sigma_{Pnom=Sandrini}(Alumno \times Examen \times Prof)))$$

requiere la construcción de un resultado intermedio de gran tamaño, la junta

$$Alumno \mid\times\mid Examen \mid\times\mid Prof.$$

De este resultado, sólo una fracción de las tuplas será utilizada para obtener la respuesta final. Si en cambio seleccionamos primero las tuplas de Examen con Nota>8 y las de Prof con Pnom=Sandrini, obtenemos un ahorro considerable. La expresión algebraica que corresponde, equivalente a la original, es:

$$\pi_1(Alumno \mid\times\mid \sigma_{Nota>8}(Examen) \mid\times\mid \sigma_{Pnom=Sandrini}(Prof)).$$

Junta Natural

Otra forma de reducir el tamaño de los resultados intermedios consiste en usar un ordenamiento óptimo de las operaciones de junta. Como la junta natural es asociativa, sabemos que, para tres relaciones cualesquiera R, S, y T, se cumple que

$$R \mid\times\mid (S \mid\times\mid T) = (R \mid\times\mid S) \mid\times\mid T.$$

El costo de evaluar la expresión de la izquierda puede, sin embargo, ser muy diferente del de evaluar la de la derecha. Por ejemplo, consideremos la junta

$$Alumno \mid\times\mid Profesor \mid\times\mid \sigma_{Nota>8}(Examen).$$

Si realizamos primero la junta *Alumno* |×| *Profesor*, vamos a obtener simplemente el producto cartesiano de todos los alumnos con todos los profesores, ya que estas dos relaciones no tienen ningún atributo en común. Sólo una fracción de las tuplas en este resultado intermedio sobrevivirán la operación de junta con $\sigma_{Nota>8}(Examen)$. En cambio, si realizamos primero la junta $Profesor \mid\times\mid \sigma_{Nota>8}(Examen)$, el resultado tendrá la misma cardinalidad que el segundo operando (suponiendo que un solo profesor toma cada examen), y ésta es también la cardinalidad del resultado final (salvo por los alumnos que aparezcan duplicados), de modo que no hemos generado tuplas "inútiles". Notemos que como la junta es también conmutativa, podríamos igualmente haber tomado primero la junta $Alumno \mid\times\mid \sigma_{Nota>8}(Examen)$, con el mismo resultado.

Proyección

Así como la selección reduce la cantidad de tuplas que aparecen en los resultados intermedios, la proyección reduce el tamaño de cada tupla y por lo tanto el tamaño total de los resultados intermedios. Por lo tanto, es ventajoso también, en general, tratar de aplicar las proyecciones lo más temprano posible. Por ejemplo, en la expresión

$$\pi_1(Alumno \mid\times\mid (Profesor \mid\times\mid \sigma_{Nota>8}(Examen))),$$

sólo vamos a usar el atributo *Anúmero* de la junta *Profesor* $|\times| \sigma_{Nota>8}(Examen)$; por lo tanto, sólo necesitamos en la junta el atributo *Anúmero* más todo atributo común a *Profesor* y a *Examen;* con lo cual podemos usar

$$\pi_{Materia,Efecha}(Profesor)$$

en lugar de *Profesor,* y

$$\pi_{Anúmero,\ Materia,\ Efecha}\ (\sigma_{Nota>8}(Examen))$$

en lugar de $\sigma_{Nota>8}\ (Examen)$. Si lo deseamos, también podemos eliminar de la junta de *Profesor* y *Examen* todos los atributos salvo *Anúmero* antes de realizar la junta con *Alumno.*

Operaciones Booleanas

Tal como ocurre con la junta natural, las operaciones de unión y diferencia gozan de propiedades que permiten efectuar las transformaciones sugeridas por las siguientes identidades:

$$\sigma_F\ (R \cup S) = \sigma_F\ (R) \cup \sigma_F\ (S)$$

$$\sigma_F\ (R - S) = \sigma_F\ (R) - S = \sigma_F\ (R) - \sigma_F\ (S)$$

$$(R \cup S) \cup T = R \cup (S \cup T)$$

$$R \cup S = S \cup R.$$

7.3. Estimaciones del Costo de Procesamiento

Para seleccionar una estrategia de ejecución de una consulta entre las muchas posibles, el optimizador de consultas debe poder estimar el costo de una estrategia, medido en cantidad de accesos a memoria secundaria. Este costo depende por supuesto de las características de los datos; en primer lugar, es necesario conocer:

1. la cardinalidad de cada relación R, n_R;
2. la longitud (en bytes) de cada tupla de R, l_R;
3. para cada atributo A y cada relación R, la cardinalidad de $\pi_A(R)$, o sea, la cantidad de valores distintos de A que aparecen en R; llamemos a esta cantidad $V(A, R)$.

Usando esta información, podemos estimar el tamaño de cualquier resultado producido por una expresión algebraica. Comencemos con el producto cartesiano: la cardinalidad de $R \times S$ es fácilmente estimable como $n_R \times n_S$, y cada tupla del producto cartesiano ocupa $l_r + l_s$ bytes.

Supongamos que queremos estimar, dado n_R, la cardinalidad de

$$\sigma_{A=k}(R),$$

donde k es una constante. Si suponemos que los valores de A aparecen con distribución uniforme, o sea, cada valor de A aparece aproximadamente con la misma frecuencia, la cardinalidad de $\sigma_{A=k}(R)$ es aproximadamente igual a

$$n_R / V(A, R).$$

En general, esta suposición no es realista; por ejemplo, si R es *Examen* y A es *Nota,* sabemos que los valores de A **no** aparecerán uniformemente distribuidos, ya que algunas notas son mucho más comunes que otras. Dada la dificultad de obtener información confiable sobre la distribución de valores de un atributo, en la práctica los optimizadores de consultas tienden a usar la suposición de distribución uniforme.

La estimación del tamaño de una junta es algo más complicada que la del producto cartesiano. Sean R y S dos relaciones que tienen en común el conjunto de atributos X. Si $X = \emptyset$, la junta se reduce a un producto cartesiano; supongamos por lo tanto que X no es el conjunto vacío. Si X es una clave para una de las relaciones, digamos R, cada tupla de S concuerda con a lo sumo una tupla de R; por lo tanto, la cardinalidad de $R\ |\times|\ S$ está acotada superiormente por n_S. Si en cambio X no es una clave para ninguna de las dos relaciones, el problema es más complicado.

Supongamos, por simplicidad, que la intersección de R y S contiene un solo atributo, que llamaremos A, o sea, $X = \{A\}$. Supongamos, además, que los valores de A se hallan uniformemente distribuidos. Para cada tupla de R, existen

$$n_S / V(A, S)$$

tuplas en S con el mismo valor de A. Por lo tanto, la cardinalidad de R |×| S se puede estimar como

$$n_R\, n_S / V(A, S).$$

Notemos que podríamos igualmente usar la expresión

$$n_R\, n_S / V(A, R).$$

Si estas dos expresiones difieren significativamente, debido a una diferencia entre $V(A, R)$ y $V(A, S)$, es plausible suponer que una de las relaciones contiene sólo algunos de los valores de A que aparecen en la otra, y en este caso la menor de las dos estimaciones será más acertada. Si en cambio R y S tienen pocos valores de A en común, ambas estimaciones serán demasiado altas; necesitamos más información para poder hacer una estimación ajustada.

Uso de Índices

Como explicamos en el capítulo anterior, el uso de estructuras de datos adicionales, tales como índices y tablas de hashing, es esencial para obtener tiempos de respuesta adecuados en el procesamiento de consultas. Hasta el momento no hemos considerado el efecto de estas estructuras en la estimación de costos de procesamiento, pero es obvio que su presencia influye fuertemente en la selección de una estrategia de resolución de una consulta.

Para simplificar el problema, consideremos simplemente una consulta equivalente a una selección algebraica seguida de proyección, o sea, en SQL:

select $A_1, \ldots, A_n$
from R
where P_1 **and** P_2 **and** ... **and** P_n

donde los P_i's son condiciones sobre los atributos de R. El método que describiremos para hallar una estrategia de resolución de estas consultas está basado en el que utiliza el Sistema R. Hay dos factores que considerar:

- **Presencia de índices**: Si uno de los P_i's es de la forma $A = k$, y existe un índice sobre el atributo A, es probablemente ventajoso obtener primero todas las tuplas de R que satisfacen esta condición, usando el índice, y verificar cuáles de ellas satisfacen los demás P_i's. El costo de este plan de acceso, medido en cantidad de bloques, será el costo de acceder el índice más el costo de acceder las tuplas; este último varía ampliamente según como estén físicamente repartidas las tuplas en el disco, como veremos inmediatamente.
- **Tipo de índices**: Supongamos que el sistema permite almacenar relaciones usando agrupamiento, como lo definimos en la Sección 6.3, de modo que las tuplas de algunas relaciones pueden aparecer densamente cargadas en bloques y otras completamente esparcidas. En el ejemplo que dimos en la Sección 6.3, las tuplas de la relación Médico aparecen almacenadas en forma compacta, mientras que las de la relación Hospital se hallan esparcidas. Definamos un *índice de agrupamiento (clustering index)* como un índice sobre un atributo tal que las tuplas que tienen el mismo valor en ese atributo tienden a aparecer físicamente contiguas en el disco. Por ejemplo, un índice sobre el atributo *hosp* en la relación Médico de la Sección 6.4 es un índice de agrupamiento, ya que las tuplas de Médico con el mismo valor de *hosp* tenderán a aparecer en forma contigua. Otro ejemplo sería cualquier relación que esté almacenada físicamente en orden de cierto atributo A; un índice sobre A será entonces un índice de agrupamiento para esa relación. La presencia de un índice de agrupamiento nos permite obtener todas las tuplas con un valor dado en el atributo indexado, usando una cantidad total de accesos cercana al mínimo número de bloques necesario para almacenar esas tuplas. En cambio, un índice que no sea de agrupamiento puede requerir un número de accesos igual al número de tuplas a obtener.

En lugar de dar una descripción detallada del método de generación de un plan de acceso, veamos un ejemplo. Sea la consulta:

select *nombre*
from *Medico*
where *espec* = 'psiq' **and** *hosp* = 45
and *número* < 400;

O sea, los nombres de los psiquiatras que trabajan en el hospital número 45 y cuyo código es menor que 400. Supongamos que tenemos las siguientes estadísticas sobre la relación Médico:

- En cada bloque caben 20 tuplas.
- *V(hosp, Médico)* = 25.
- *V(espec, Médico)* = 20.
- *V(número, Médico)* = 5000.
- $n_{Médico}$ = 5000.

Supongamos que existen dos índices en forma de árboles-B: un índice de agrupamiento sobre el atributo *hosp* y uno no de agrupamiento sobre el atributo *espec*. Supongamos también que todos los atributos aparecen uniformemente distribuidos.

Tenemos que elegir entre los dos índices disponibles para obtener el conjunto inicial de tuplas, un subconjunto del cual formará la respuesta. Supongamos que comenzamos usando el índice sobre *hosp* para extraer el conjunto de tuplas con *hosp* = 45. Como éste es un índice de agrupamiento, la cantidad de bloques a acceder será igual al cociente entre la cantidad de tuplas con *hosp* = 45 y la cantidad de tuplas por bloque. A este número deberemos sumar el costo de acceder el índice para obtener el costo total de procesamiento. Dada la suposición de uniformidad, la cantidad de tuplas con el número de hospital buscado es 5000/25 = 200. Como la cantidad de tuplas por bloque es 20, el costo en número de accesos resulta igual a 200/20 = 10. Notemos que a medida que obtenemos estas tuplas podemos ir verificando si satisfacen las otras dos condiciones, por lo que el costo total de la consulta usando este método es de 10 accesos más el costo de acceder al índice. Supongamos que el árbol-B contiene entre 5 y 10 punteros por nodo, y que cada puntero de un nodo terminal apunta al primer bloque del archivo que contiene registros con el valor asociado de *hosp*. La cantidad de punteros necesaria entonces es igual a la cantidad de valores distintos del atributo, o sea 25, lo que implica un mínimo de 3 nodos terminales y un máximo de 5, con lo que la profundidad del árbol será igual a 1, o sea que el costo de acceder el índice es de 2 accesos. El costo total de esta estrategia resulta entonces de 12 accesos.

Si en cambio usamos el índice sobre *espec*, el cálculo del costo es el siguiente. La cantidad de tuplas con especialidad psiquiatría será aproximadamente 5000/50 = 100. Como el índice no es de agrupamiento, la cantidad de bloques a acceder será también del orden de 100. Sin necesidad de calcular los accesos al índice, que serán 2 ó 3, vemos que es más conveniente por un orden de magnitud usar el índice sobre *hosp* en lugar del de *espec*. Notemos que, si ninguno de los índice hubiera sido de agrupamiento, hubiéramos llegado a la conclusión contraria, ya que el costo usando *espec* sería del orden de 100 contra 200 para *hosp*.

Finalmente, notemos que no es conveniente acceder primero la tuplas que satisfacen la condición *número < 400*, ya que esto requeriría una lectura secuencial de todas las tuplas de la relación, con un costo de 5000/20 = 250 accesos.

Generalizando a partir de este ejemplo, el método seguido en el Sistema R y otros sistemas similares en la optimización de consulta de este tipo se puede resumir como sigue. Consideremos todas la estrategias que consisten en una de las dos posibilidades siguientes:

- Elegir un P_i, obtener todas las tuplas que satisfacen P_i, y verificar cuáles de ellas satisfacen los demás predicados.
- Hacer un barrido de la relación completa verificando cuáles tuplas satisfacen todos los predicados.

Calculamos el costo de cada posibilidad tomando en cuenta la presencia de índices y si se trata o no de índices de agrupamiento, y finalmente elegimos como plan de acceso aquella que resulte de menor costo.

7.4. MÉTODOS DE CÁLCULO DE JUNTAS

La junta es probablemente la operación binaria más frecuente y su implementación ha sido cuidadosamente estudiada. En esta sección consideraremos varias estrategias alternativas para el cálculo de una junta natural: iteración ingenua, iteración por bloques, *sort-merge*, uso de índices, *hash*. Analizaremos también métodos aplicables a juntas múltiples y el índice de junta de mapa de bits.

Iteración Ingenua

Consideremos como ejemplo la junta natural

Hospital |×| *Personal*

y supongamos inicialmente que no disponemos de ningún índice. Digamos que la relación *Hospital* contiene 50 tuplas, y *Personal* contiene 5000. Supongamos que en cada bloque caben 20 tuplas de *Personal* y 25 de *Hospital*. Asumamos que ambas relaciones se hallan almacenadas físicamente en forma contigua, de modo que *Personal* ocupa 250 bloques y *Hospital* ocupa 2.

El método de iteración ingenua consiste simplemente en examinar todos los pares de tuplas posibles formados por una tupla de *Hospital* y una de *Personal*. Esto requiere examinar $5000 \times 50 = 250.000$ pares de tuplas. ¿Cuántos accesos a memoria secundaria serán necesarios? Esto depende de la estrategia que se siga. Por ejemplo, supongamos que leemos un bloque de la relación *Personal*, y para cada tupla que aparece en este bloque leemos secuencialmente toda la relación *Hospital*. En este caso, necesitamos $5000/20 = 250$ accesos para leer todos los bloques de *Personal*. Una lectura completa de *Hospital* requiere 2 accesos; como la cantidad de lecturas completas de *Hospital* es $n_{personal} = 5000$, el total de accesos a *Hospital* es de 10.000, y el costo total es 10.250 accesos.

Iteración por Bloques

Considerando el método de la sección anterior, es fácil observar que la siguiente modificación será ventajosa. Leemos un bloque de *Personal*; para este bloque de *Personal*, leemos todos los bloques de *Hospital* y hallamos *todas* las tuplas que participan en la junta en este par de bloques. De este modo, la cantidad de lecturas completas de *Hospital* es igual a la cantidad de *bloques* en *Personal*, no a la cantidad de tuplas; el costo total resulta entonces de $(5000/20) \times 3 = 750$ accesos.

Estos cálculos suponen que sólo tenemos espacio en memoria principal para mantener un bloque de cada relación; en la práctica, probablemente dispondremos de más espacio y podremos por ejemplo cargar la relación *Hospital*, que tiene sólo 2 bloques, totalmente en memoria principal. Si hacemos esto y luego leemos la relación *Personal* bloque por bloque secuencialmente, el costo se reduce a 2 accesos para la lectura inicial de *Hospital*, más 250 accesos para la única lectura de *Personal*, con un total de 252 accesos; una mejora notable sobre los 10.250 de la iteración ingenua. Intuitivamente, observamos que este método debe ser óptimo, ya que su costo es simplemente el de leer completamente ambas relaciones, y todo método debe al menos incurrir este costo. En caso de que ninguna de las dos relaciones pueda

ser cargada completamente en memoria principal, podemos modificar el método de modo que cargue tantos bloques como sea posible de una de las relaciones y, para cada conjunto de bloques de la primera relación, recorra completamente la segunda.

Método de *Sort-merge*

En caso que ninguna de las dos relaciones quepa completamente en memoria principal, otra estrategia eficiente consiste en ordenar las dos relaciones de acuerdo con el valor del atributo de junta *(sort)*, y luego calcular la junta mediante un simple barrido secuencial de ambas relaciones *(merge)*. Una vez ordenadas las relaciones, el costo del *merge* es simplemente la suma de la cantidad de bloques ocupada por cada relación (252 en el ejemplo); o sea que este método iguala al que almacena una relación totalmente en memoria principal en el caso en el que las relaciones ya están ordenadas de antemano. Si no es así, el costo adicional del ordenamiento es del orden de $B \log B$ para una relación que ocupa B bloques.

Uso de Índices

Supongamos ahora que tenemos un índice sobre el atributo *hosp* de la relación *Personal*. Volviendo a la iteración ingenua, podemos ahora modificarla del siguiente modo. Para cada bloque que leemos de *Hospital*, usamos el índice para extraer de *Personal* aquellos bloques que contienen tuplas con los valores de *hosp* que aparecen en este bloque de *Hospital*.

Aunque en este ejemplo este método resultará más costoso que la iteración por bloques, debido al costo extra de los accesos al índice, hay dos situaciones en las cuales su performance es muy superior. La primera es el caso en que las tuplas de *Personal* no se hallan almacenadas en forma contigua en bloques; por ejemplo, si aparecen entrelazadas con las tuplas de otra relación usando el método de agrupamiento. En este caso el uso del índice nos permite acceder sólo aquellos bloques que contienen tuplas de *Personal* ignorando los otros. La segunda situación ocurre si sólo una fracción de las tuplas de la segunda relación participan en la junta. Por ejemplo, supongamos que la relación *Personal* contiene información sobre personal de los hospitales de todo el país, mientras que *Hospital* solamente lista los hospitales de una ciudad dada. En este caso el índice nos permite acceder solamente los

bloques relevantes a la consulta e ignorar los demás. Las ventajas del uso de índices en estas situaciones son tan significativas que generalmente los sistemas consideran la posibilidad de crear un índice expresamente para el cálculo de una junta, si éste no existiera de antemano.

Método *Hash*

Se trata, en realidad, de una familia de métodos que se caracterizan porque es posible acotar la cantidad de comparaciones entre las tuplas de las relaciones intervinientes. En estos métodos no se requiere que las relaciones se encuentren o sean clasificadas, que posean algún tipo de índice o se encuentren limitadas por su tamaño. La idea es separar las tuplas de una relación que pueden ser juntadas con ciertas otras tuplas de la otra relación.

Un primer método a analizar es el llamado *Método Simple de Junta Hash*. Se distinguen dos fases bien definidas. La primera se denomina fase *constructiva* y consiste en armar una tabla hash en memoria con los datos de la primera relación, supuestamente la más pequeña, que llamamos R. Para esto se utiliza una función de hash h que se aplica a cada tupla, específicamente a los valores de los atributos sobre los que se realiza la junta. Asumiremos que las funciones de hash mapean los valores de clave a un espacio de direcciones de manera uniforme. Llamemos X al conjunto de atributos en función de los cuales se realiza la junta. La función h transforma valores de X en direcciones dentro de la tabla hash que designaremos con TH. En TH se pueden almacenar las tuplas completas o simplemente los *tid* (identificadores de tupla) y los correspondientes valores de X. Obviamente, se debe considerar algún método para la resolución de colisiones. En la segunda fase, denominada *exploratoria*, se aplica la función h a las tuplas de la otra tabla, que llamamos S. De esta manera es posible ubicar, en la tabla TH, y juntar con cada tupla de S, las correspondientes filas de R. En cuanto al costo de este método, éste resulta del orden de la suma de la cantidad de bloques de ambas relaciones, ya que cada una de ellas es leída una sola vez. Por ejemplo, para realizar la junta entre las tablas *Personal* y *Hospital* se requieren 252 accesos para su lectura. Obviamente, la tabla hash en memoria TH se construiría con la relación *Hospital*, lo que minimizaría las comparaciones para realizar la junta.

Si las relaciones no caben en memoria principal, el Método GRACE, basado en el algoritmo de la máquina de base de datos del proyecto japonés de Quinta Generación, se presenta como una alternativa eficiente.

GRACE se lleva a cabo en dos etapas, para lo cual usa dos funciones de hash. La primer función, que se denomina *función de particionamiento*, la designamos con h_1 y se utiliza para particionar las relaciones R y S en subconjuntos disjuntos. Si M es la cantidad de particiones seleccionada, h_1 aplicado a los componentes de X de cualquier tupla produce un valor entre 0 y M – 1. En la segunda etapa se juntan las correspondientes particiones utilizando el Método Simple anterior. El algoritmo siguiente usa M buffers para manipular las tuplas procesadas durante la primer etapa:

Etapa 1

i. Leer R. Para cada tupla t de R calcular $h_1(t[X])$, lo que resulta en un número de buffer. Grabar la tupla en el buffer en memoria y cuando alguno se complete grabarlo al disco. Después de leer toda R grabar el contenido de los buffers al disco. De esta manera R queda particionada en subconjuntos R_i, con $0 \le i \le M - 1$.

ii. Leer S y hacer lo mismo que en el paso 1, generando una partición para S de subconjuntos S_j, con $0 \le j \le M - 1$.

Etapa 2

Para i = 0, ..., M – 1

iii. Leer R_i y construir con sus tuplas una tabla hash TH en memoria, utilizando una función de hash h.

iv. Leer S_i y aplicar a cada una de sus tuplas la misma función h del paso iii. Para cada tupla de S_i, verificar si en TH existe alguna tupla juntable. Si es así, emitir una tupla del resultado y luego continuar con la siguiente tupla de S_i.

En este método, la cantidad de bloques a leer se duplica con respecto al Método Simple; pero todavía resulta más económico que el Sort-Merge analizado anteriormente.

7.5. JUNTA DE MÚLTIPLES RELACIONES

Una manera de resolver una consulta que involucra la junta de múltiples relaciones $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_k$, con $k > 2$, es traducir la expresión a una se-

cuencia de juntas binarias. Cada junta, por su parte, puede ser implementada usando diversos algoritmos, tales como los descriptos anteriormente. Existen diversas alternativas de evaluación de la junta múltiple, en función de las distintas formas de aplicación de paréntesis. Por ejemplo, si k = 4, tenemos $(R_1 |\times| R_2) |\times| (R_3 |\times| R_4)$, $(R_1 |\times| (R_2 |\times| R_3)) |\times| R_4$, $(R_1 |\times| R_3) |\times| (R_2 |\times| R_4)$, etc., expresiones que, en virtud de las propiedades conmutativa y asociativa de la junta natural, sabemos que son equivalentes.

Para evitar grabar, y luego volver a leer, los resultados intermedios de las juntas, se utiliza una técnica conocida como *pipeline*. De acuerdo con esta técnica, los resultados intermedios de una operación son utilizados inmediatamente en la siguiente junta. De esta manera, sólo el resultado final es grabado al disco. Obviamente, de todas las expresiones equivalentes, algunas son más susceptibles que otras de beneficiarse con la aplicación de la estrategia de pipeline. En general, son las que se encuentran anidadas en paréntesis de izquierda a derecha, por ejemplo las del tipo $((R_1 |\times| R_2) |\times| R_3) |\times| R_4$.

A continuación describiremos el algoritmo que implementa la estrategia de pipeline, para k = 4:

for each tupla t en R_1
  recuperar tuplas en R_2 que sean juntables con t en Q_1
  juntar t con cada tupla recuperada de R_2 obteniendo $T_1 = t \cdot Q_1$
  for each tupla t_1 en T_1
    recuperar tuplas en R_3 que sean juntables con t_1 en Q_2
    juntar t_1 con cada tupla recuperada de R_3 obteniendo $T_2 = t_1 \cdot Q_2$
    for each tupla t_2 en T_2
      recuperar tuplas en R_4 que sean juntables con t_2 en Q_3
      juntar t_2 con cada tupla recuperada de R_4 obteniendo $T_3 = t_2 \cdot Q_3$
      agregar T_3 al resultado final
    endfor
  endfor
endfor

7.6. MÉTODO DE ÍNDICE DE JUNTA CON MAPAS DE BITS

Este método realiza la junta por medio de índices que utilizan como representación vectores de bits comprimidos. Permite juntar múltiples tablas, lo que resulta de aplicación, particularmente, en ambientes de soporte de decisiones (Data Warehousing).

El método resulta de la combinación de las siguientes técnicas:

i) *Índices de Junta*

Un índice de junta, para relaciones R y S, se define como una relación binaria que identifica cada tupla resultado de la junta, por medio de punteros a las respectivas tuplas de las relaciones R y S. Se puede considerar entonces un índice de junta como una forma especial de vista materializada.

Cada puntero puede representarse como un valor de una clave, una dirección física, o cualquier otro valor que identifique en forma unívoca a cada tupla en una relación.

El algoritmo para ejecutar la junta de R con S por medio de este método se reduce a:

a) Leer el índice de junta y recuperar los punteros a pares de tuplas juntables (x,y).

b) Concatenar los pares (x,y) hallados, formando tuplas z y almacenarlas en la relación resultado T.

El costo más importante en este método, una vez construido el índice, es el que surge de recuperar las tuplas juntables. Luego, el mejor caso resulta cuando ambas relaciones, R y S, están ordenadas o agrupadas en función de los atributos en base a los cuales se realiza la junta. El costo resulta del orden de la suma de la cantidad de bloques que ocupan R y S. En el peor caso, el orden asciende a la suma de la cantidad de tuplas de R y S.

ii) *Índices de mapas de bits*

Sea una relación R con N tuplas o filas y un determinado atributo o columna A, tal que $|dom(A)|$ (el cardinal del dominio de A) sea M. Un índice de mapa de bits es un conjunto de M vectores o cadenas binarias de longitud N. Cada fila de la tabla se asocia con el mismo número de bit en cada cadena. En particular, en la cadena que corresponde al valor que la fila tiene en la columna A, ese bit es 1 y 0 en las restantes. Así, por ejem-

plo, si se trata de una relación con datos de personas con 1000 filas y el atributo sobre el que se define el índice es Sexo, el índice de mapa de bits estará integrado por dos vectores o cadenas. Una cadena se utilizará para el valor "Varón" y otra para el valor "Mujer". El primer bit de la cadena correspondiente a "Varón" será 1 si la primera fila de la relación corresponde a un varón y, en ese caso, el primer bit de la cadena correspondiente a "Mujer" será 0. Veamos otro ejemplo: se registran las ventas de los productos de una empresa en las distintas sucursales. Un índice de mapa de bits tendrá tantas cadenas de bits como sucursales tenga la empresa. Si la fila 331 de la relación de ventas corresponde a un producto que se vendió en la sucursal 14, luego la cadena 14 tendrá un 1 en la posición 331. Las demás cadenas, correspondientes a las otras sucursales, tendrán 0 en la posición 331.

Existen diversas técnicas para compresión de datos, las que permiten reducir drásticamente el espacio necesario para almacenar estos índices. Una ventaja del uso de los índices de mapa de bits radica en la reducción de las entradas/salidas al dispositivo de almacenamiento. Otra ventaja se debe a que las combinaciones de predicados AND, OR y NOT se pueden ejecutar muy eficientemente, en paralelo, utilizando las instrucciones de los procesadores de 64 bits o más.

Un caso particular es el método denominado **junta estrella.** Este método permite realizar la junta de una relación central, denominada *tabla de hechos*, con múltiples relaciones descriptivas, denominadas *tablas de dimensiones*. Las claves primarias de las tablas de dimensiones son las claves foráneas de la tabla de hechos y, consecuentemente, las juntas se realizan sobre esos atributos. Para cada posible junta se define entonces un índice de junta en el que se incluyen, para cada fila en una tabla de dimensiones, una cadena de bits comprimida. Esta última representa el conjunto de filas en la tabla de hechos que se juntan con la correspondiente fila en la tabla de dimensiones.

EJERCICIOS

E1. Sean las siguientes tres relaciones:

$R(A, B, C)$, $S(B, C, D)$ y $T(D, E)$

Asumir que:

$n_R = 10.000$, $n_S = 20.000$ y $n_T = 30.000$
$V(A, R) = 100$, $V(B, R) = 200$, $V(C, R) = 200$,
$V(B, S) = 400$, $V(C, S) = 500$, $V(D, S) = 200$,
$V(D, T) = 100$, $V(E, T) = 200$
Un bloque puede alojar 50 tuplas de cualquier relación.
Se pide estimar el tamaño de la junta $R \bowtie S \bowtie T$.

E2. Dada la relación $R(A, B, C)$ y la consulta

```
select *
from R
where A > 10 and B = 10 and C = 5;
```

Asumir que:
$n_R = 10.000$, $b_R = 1000$ (cantidad de bloques de r)
$V(A, R) = 90$, $V(B, R) = 150$, $V(C, R) = 150$.

Suponer que existe un índice de agrupamiento para A e índices de no agrupamiento para B y C.
¿Cuáles de las siguientes opciones requieren la lectura de la mínima cantidad de bloques para la consulta?:

a) leer la relación R íntegramente a memoria.
b) realizar primero la selección A > 10.
c) realizar primero la selección B = 10.
d) realizar primero la selección C = 5.

E3. Sea la siguiente base de datos:

Empleado (nro_e(4), nom_e(32), domicilio(24), ciudad(10), edad(2), dni(8), tel(12), sueldo(6))
Trabaja_en(nro_e, nro_depto)
Departamento(nro_depto(4), nom_depto(20), ubicación(20), ciudad(10))
Gerencia(nro_e, nro_depto, desde(8), hasta(8))