Capítulo 13: Procesamiento de consultas

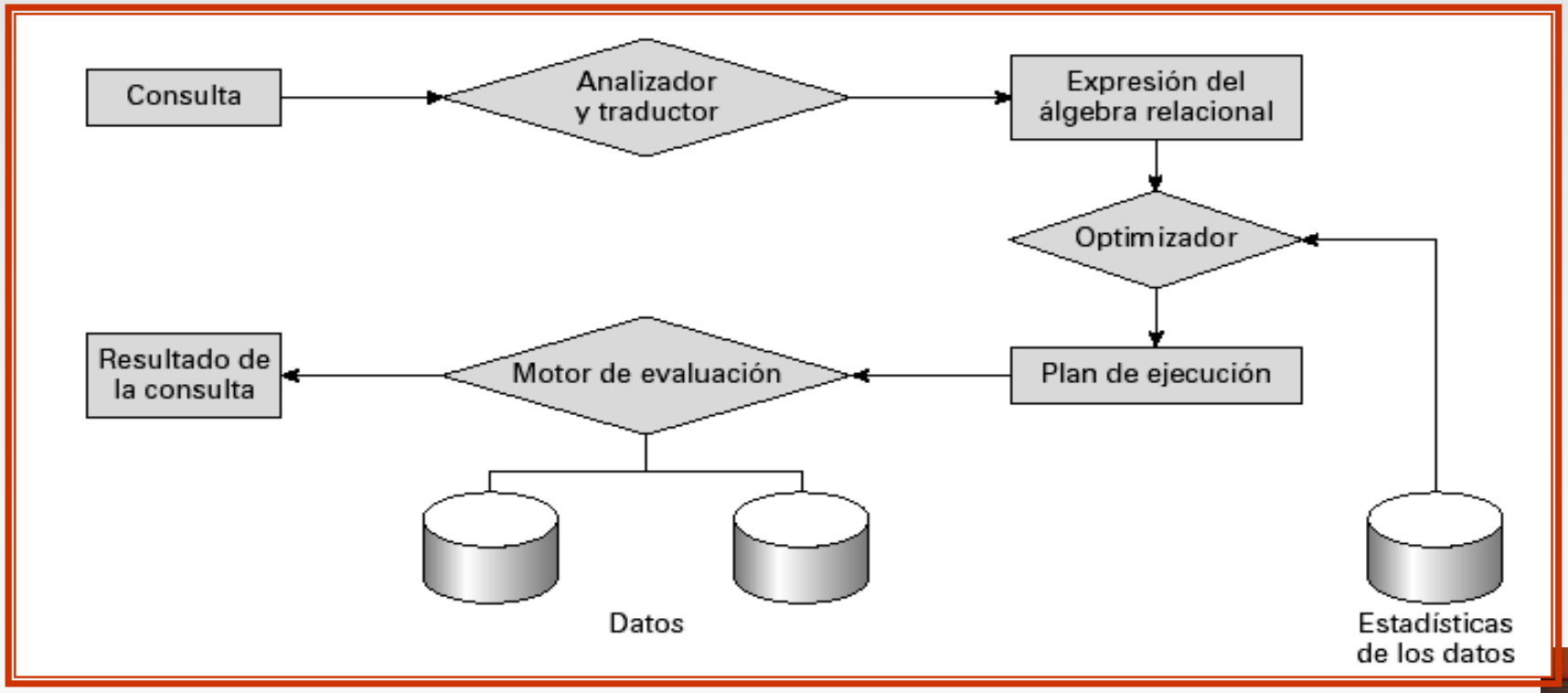
Fundamentos de Bases de datos, 5ª Edición.

Capítulo 13: Procesamiento de consultas

- Introducción
- Medidas del coste de consultas
- Operación selección
- Ordenación
- Operación reunión
- Otras operaciones
- Evaluación de expresiones

Pasos básicos en el procesamiento de consultas

1. Análisis y traducción
2. Optimización
3. Evaluación



Pasos básicos en el procesamiento de consultas (Cont.)

- Análisis y traducción
  - traducir la consulta en su forma interna. Esto se traduce entonces al álgebra relacional.
  - El analizador comprueba la sintaxis, verifica relaciones
- Evaluación
  - El motor de ejecución de consultas toma un plan de evaluación de la consulta, ejecuta ese plan y devuelve las respuestas a la consulta.

Pasos básicos en el procesamiento de consultas: Optimización

- Una expresión del álgebra relacional puede tener muchas expresiones equivalentes
  - Por ejemplo, $\sigma_{saldo<2500}(\Pi_{saldo}(cuenta))$ es equivalente a $\Pi_{saldo}(\sigma_{saldo<2500}(cuenta))$
- Cada operación de álgebra relacional se puede evaluar empleando uno de los diferentes algoritmos
  - Igualmente, una expresión del álgebra relacional se puede evaluar de muchas maneras.
- La expresión comentada que especifica la estrategia de evaluación detallada, se denomina **plan de evaluación**.
  - Por ejemplo, puede usar un índice sobre *saldo* para encontrar las cuentas con *saldo* < 2.500,
  - o puede llevar a cabo el rastreo de la relación completa y eliminar las cuentas con *saldo* ≥ 2.500

Pasos básicos: Optimización (Cont.)

- **Optimización de consultas**: Entre todos los planes de evaluación equivalentes, seleccionar el de menor coste.
  - El coste se estima empleando información estadística del catálogo de la base de datos
    - por ejemplo, el número de tuplas de cada relación, el tamaño de la tuplas, etc.
- En este capítulo se estudia
  - Cómo medir los costes de las consultas
  - Algoritmos para evaluar operaciones de álgebra relacional
  - Cómo combinar algoritmos para las operaciones individuales, a la hora de evaluar una expresión completa
- En el Capítulo 14
  - Se estudia cómo optimizar consultas, esto es, cómo encontrar un plan de evaluación con coste estimado bajo

Medidas del coste de consultas

- El coste se mide generalmente como el tiempo total transcurrido para responder la consulta
  - Numerosos factores contribuyen al coste del tiempo
    - *accesos a disco*, *CPU*, o incluso las *comunicaciones* de la red
- El típico acceso a disco es el coste predominante y es también relativamente fácil de estimar. Se mide teniendo en cuenta
  - Número de búsquedas * coste medio de búsqueda
  - Número de bloques leídos * coste medio de lectura de bloque
  - Número de bloques escritos * coste medio de escritura de bloque
    - El coste para escribir un bloque es mayor que el coste para leerlo
      - el dato es leído de nuevo, después de ser escrito, para asegurar que la escritura se ha realizado correctamente

Medidas del coste de consultas (cont.)

- Por simplicidad se emplea sólo el *número de transferencias de bloques de disco*, y el *número de búsquedas* como la medida del coste
  - t_T – tiempo para transferir un bloque de datos
  - t_S – tiempo de una búsqueda
  - Coste de transferir for b bloques más S búsquedas
    $$b * t_T + S * t_S$$
- Se ignora por simplicidad los costes de CPU
  - En sistemas reales se tiene en cuenta el coste de CPU
- La estimación de costes no incluye el coste de escribir el resultado final en disco
- Algunos algoritmos pueden reducir las ES de disco utilizando espacio adicional
  - La cantidad de memoria real disponible para el búfer depende de otras consultas concurrentes y otros procesos del SO, que solo se conocen durante la ejecución.
    - Se suelen utilizar estimaciones de caso peor, suponiendo disponible la mínima cantidad de memoria necesaria para la operación.
- Los datos necesarios puede que ya se encuentren en el búfer, evitando acceso a discos.
  - Pero resulta difícil tenerlo en cuenta para la estimación de coste

Operación selección

- **Explorador de archivo** – algoritmos de búsqueda que localizan y recuperan registros que cumplen una condición de selección.
- Algoritmo **A1** (*búsqueda lineal*). Explorar cada bloque del fichero y comprobar todos los registros para ver si cumplen la condición de selección.
  - Coste estimado = b_r bloques trasferidos + 1 búsqueda
    - b_r indica el número de bloques que contienen registros de la relación *r*
  - Si la selección se hace sobre un atributo clave, puede terminar si se encuentra el registro
    - Coste = (b_r /2) bloques trasferidos + 1 búsqueda
  - La búsqueda lineal se puede aplicar independientemente de
    - la condición de selección o
    - la ordenación de los registros en el fichero o
    - la disponibilidad de índices

Operación selección (cont.)

- **A2** *(búsqueda binaria).* Aplicable si la selección es una comparación de igualdad sobre el atributo en que está ordenado el archivo.
  - Suponer que los bloques de una relación están almacenados ordenadamente
  - Coste estimado (número de bloques de disco a rastrear):
    - coste de localizar la primera tupla mediante una búsqueda binaria sobre los bloques
      - $\lceil \log_2(b_r) \rceil * (t_T + t_S)$
    - Si existen múltiples registros que cumplen la condición de selección
      - *Más costes de transferir* el número de bloques que contienen los registros que satisfacen la condición de selección
      - En el Capítulo 14 se verá como estimar este coste

Selecciones empleando índices

- **Exploración del índice** – algoritmos de búsqueda que emplean un índice
  - la condición de selección debe estar sobre la clave de búsqueda del índice.
- **A3** (*índice primario sobre clave candidata, igualdad*). Recuperar un solo registro que cumpla la condición de igualdad correspondiente
  - *Coste* = $(h_i + 1) * (t_T + t_S)$
- **A4** (*índice primario sobre ninguna clave, igualdad)* Recuperar múltiples registros.
  - Los registros estarán en bloques consecutivos
    - Donde b = número de bloques que contienen registros recuperados
  - *Coste* = $h_i * (t_T + t_S) + t_S + t_T * b$
- **A5** (*igualdad sobre la clave de búsqueda de índice secundario).*
  - Recuperar un solo registro si la clave de búsqueda es una clave candidata
    - *Coste* = $(h_i + 1) * (t_T + t_S)$
  - Recuperar múltiples registros si la clave de búsqueda no es una clave candidata
    - Cada registro recuperado *n* puede estar en un bloque diferente
    - Coste = $(h_i + n) * (t_T + t_S)$
      - ¡Puede ser muy caro!

Selecciones que implican comparaciones

- Se pueden implementar selecciones de la forma $\sigma_{A \le V}(r)$ or $\sigma_{A \ge V}(r)$ empleando
  - un explorador de archivo lineal o búsqueda binaria,
  - o empleando índices de las siguientes maneras:
- **A6** (*índice primario, comparación).* (Relación ordenada sobre A)
  - Para $\sigma_{A \ge V}(r)$ emplear índice para encontrar la prima tupla $\ge v$ y rastrear la relación secuencialmente desde allí
  - Para $\sigma_{A \le V}(r)$ sólo rastrear la relación secuencialmente hasta la primera tupla $> v$; no emplear índice
- **A7** (*índice secundario, comparación).*
  - Para $\sigma_{A \ge V}(r)$ emplear índice para encontrar el primer índice $\ge v$ y rastrear el índice secuencialmente desde allí, para encontrar los punteros a los registros.
  - Para $\sigma_{A \le V}(r)$ rastrear sólo las páginas hoja de índice, buscando punteros a los registros, hasta la primera entrada $> v$
  - En cualquier caso, recuperar los registros que son apuntados
    - requiere una E/S por cada registro
    - el explorador de archivo lineal puede ser más barato

Implementación de selecciones complejas

- **Conjunción:** $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \theta n}(r)$
- **A8** (*selección conjuntiva utilizando un índice).*
  - Seleccionar una combinación de θ_i y algoritmos A1 hasta A7 que resulte del menor coste para $\sigma_{\theta i}$ *(r)*.
  - Probar otras condiciones sobre tuplas, después de llevarlas a la memoria intermedia.
- **A9** (*selección conjuntiva utilizando índices de claves múltiples).*
  - Emplear, si están disponibles, índices (clave múltiple) combinados apropiadamente.
- **A10** (*selección conjuntiva mediante la intersección de identificadores).*
  - Requiere índices con punteros de registros.
  - Emplear el correspondiente índice para cada condición y tomar la intersección de todos los conjuntos de punteros de registros obtenidos.
  - Entonces, tomar los registros del archivo
  - Si algunas condiciones no tienen los índices apropiados, aplicar pruebas en memoria.

Algoritmos para selecciones complejas

- **Disyunción:** $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \theta n}(r)$.
- **A11** (*Selección disyuntiva mediante la unión de identificadores).*
  - Aplicable si *todas* las condiciones tienen índices disponibles.
    - De lo contrario, emplear rastreo lineal.
  - Emplear el correspondiente índice para cada condición y tomar la unión de todos los conjuntos de punteros de registros obtenidos.
  - Entonces, tomar los registros del archivo
- **Negación:** $\sigma_{\neg\theta}(r)$
  - Emplear un rastreo lineal sobre el archivo
  - Si muy pocos registros cumplen $\neg\theta$, y es aplicable un índice a θ
    - Encontrar los registros que lo cumplen empleando índice y tomarlos del archivo

Ordenación

- Se puede construir un índice sobre la relación y usarlo entonces para leer la relación de forma ordenada. Puede conducir a un acceso de bloque a disco por cada tupla.
- Para las relaciones que encajan en memoria, se pueden emplear técnicas como la ordenación rápida. Para relaciones que no encajan en la memoria, la **mezcla-ordenación externa** es una buena elección.

Mezcla-ordenación externa

Sea *M* el tamaño de la memoria (en páginas).

1. **Crear ejecuciones ordenadas**. Sea *i* a 0 inicialmente.

   Repetir lo siguiente hasta el final de la relación:

   (a) Leer *M* bloques de la relación en memoria

   (b) Ordenar los bloques de la memoria

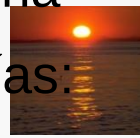
   (c) Escribir los datos ordenados de la ejecución R_i; aumentar *i*.

   Sea *N* el valor final de *i*

2. *Mezclar las ejecuciones (siguiente transparencia)…..*

Mezcla-ordenación externa(Cont.)

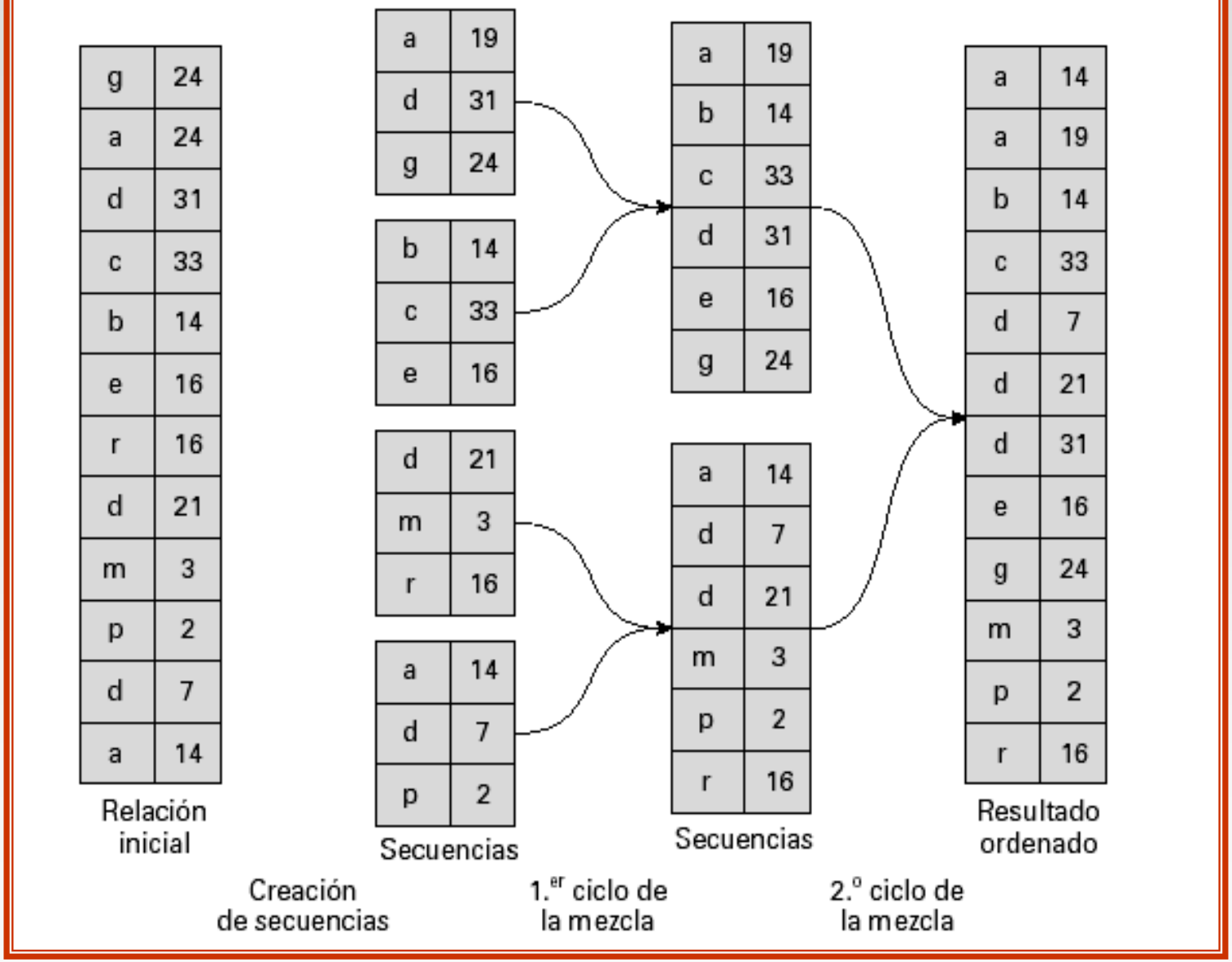
2. **Mezclar las secuencias (mezcla de N vías).** Se asume (por ahora) que N < M
    1. Emplear N bloques de memoria para las secuencias de entrada de la memoria intermedia y 1 bloque para la salida de la memoria intermedia. Leer el primer bloque de cada secuencia en su página de memoria intermedia
    2. **repetir**
        1. Seleccionar el primer registro (en forma ordenada) entre todas las páginas de la memoria intermedia
        2. Grabar el registro sobre la memoria intermedia de salida. Si la memoria intermedia de salida está llena, grabarlo sobre disco
        3. Borrar el registro de su página de la memoria intermedia de entrada.
           Si la página de la memoria intermedia se vacía **entonces**
           leer el siguiente bloque (si hay) de la secuencia en la memoria
    3. **hasta** que todas las páginas de la memoria intermedia estén vacías.



Mezcla-ordenación externa (cont.)

- Si $i \geq M$, se requieren varios *ciclos* de mezclas.
  - En cada ciclo, se mezclan los grupos contiguos de M - 1 secuencias.
  - Un ciclo reduce el número de secuencias en un factor $M - 1$ y crea secuencias mayores por el mismo factor.
    - Por ejemplo Si M = 11y hay 90 secuencias, un ciclo reduce el número de secuencias a 9, cada 10 veces el tamaño de las secuencias iniciales
  - Se repiten los ciclos hasta que todas las secuencias se han mezclado en una.

Ejemplo: Ordenación externa empleando mezcla-ordenación



Mezcla-ordenación externa (cont.)

- Análisis de costes:
  - Número total de ciclos de mezcla requeridos: $\lceil \log_{M-1}(b_r/M) \rceil$.
  - Accesos a disco por la creación de la secuencia inicial, así como en cada ciclo es $2b_r$
    - para el ciclo final, no se cuenta el coste de escritura
      - se ignora el coste de escritura final de todas las operaciones, dado que la salida de una operación puede enviarse a la operación origen, sin grabarse en disco
    - Así, el número total de accesos a disco para ordenaciones externas:

$$b_r (2 \lceil \log_{M-1}(b_r / M) \rceil + 1)$$

  - Búsquedas: próxima transparencia

Mezcla-ordenación externa (cont.)

- Costes de búsqueda
  - Durante la generación de secuencias: una búsqueda para leer cada secuencia y una búsquedas para grabar cada secuencia
    - $2\lceil b_r / M \rceil$
  - Durante la fase de mezcla
    - Tamaño de memoria intermedia: b_b (leer/grabar b_b bloques)
    - Se necesitan $2\lceil b_r / b_b \rceil$ búsquedas por cada pasada de mezcla
      - Excepto en la final que no necesita grabar
    - El número total de búsquedas es:
      $2\lceil b_r / M \rceil + \lceil b_r / b_b \rceil (2\lceil \log_{M-1}(b_r / M) \rceil - 1)$

Operación reunión

- Diferentes algoritmos para implementar reuniones
  - Reunión en bucle anidado
  - Reunión en bucle anidado por bloques
  - Reunión en bucle anidado indexada
  - Reunión por mezcla
  - Reunión por asociación
- Elección basada en el coste estimado
- Los ejemplos hacen uso de la siguiente información
  - Número de registros de *cliente*. 10.000 *impositor*: 5000
  - Número de bloques de *cliente*. 400 *impositor*: 100

Reunión en bucle anidado

- Para calcular la reunión zeta $r \bowtie_\theta s$

  for each tupla t_r **in** r **do begin**
  for each tupla t_s **in** s **do begin**
  verificar el par (t_r, t_s) para ver si cumplen la condición de reunión θ
  si lo hacen, sumar $t_r \bullet t_s$ al resultado.
  end
  end

- r se denomina **relación externa** y s **relación interna** de la reunión.
- No se requieren índices y se pueden emplear con cualquier tipo de condición de reunión.
- Es costoso dado que examina cada par de tuplas en las dos relaciones.

Reunión en bucle anidado (cont.)

- En el peor de los casos, si hay suficiente memoria sólo para conservar un bloque de cada relación, el coste estimado es
  $$n_r * b_s + b_r$$
  accesos a disco, más
  $$n_r + b_r$$
  búsqueda
- Si la relación más pequeña encaja totalmente en memoria, emplearla como relación interna.
  - Reduce el coste a $b_r + b_s$ accesos a disco y 2 búsquedas.
- Suponiendo el caso peor de disponibilidad de memoria, el coste estimado es
  - con *impositor* como relación externa:
    - 5000 ∗ 400 + 100 = 2.000.100 accesos a disco,
    - 5000 + 100 = 5100 búsquedas
  - con *cliente* como relación externa
    - 10000 ∗ 100 + 400 = 1.000.400 accesos a disco y 10.400 busquedas
- Si la relación más pequeña (impositor) encaja totalmente en memoria, el coste estimado será de 500 accesos a disco.
- Es preferible el algoritmo de reunión de bucle anidado por bloques (siguiente transparencia).

Reunión en bucle anidado por bloques

- Variante de la reunión en bucle anidado, en que cada bloque de la relación interna es emparejado con cada bloque de la relación externa.

```
for each bloque B_r of r do begin
    for each bloque B_s of s do begin
        for each tupla t_r in B_r do begin
            for each tupla t_s in B_s do begin
                Comprobar si (t_r, t_s) cumple la condición reunión
                si lo hacen, añadir t_r • t_s al resultado.
            end
        end
    end
end
```

Reunión en bucle anidado por bloques (cont.)

- Estimación en el peor de los casos: $b_r * b_s + b_r$ accesos a bloques.
  - Cada bloque, en la relación interna *s*, es leído una vez por cada *bloque* en la relación externa (en vez de una vez por cada tupla en la relación externa)
- Mejor de los casos: $b_r + b_s$ accesos a bloques + 2 búsquedas.
- Mejoras para los algoritmos de bucles anidados y bucles anidados por bloques:
  - En los bucles anidados por bloques, se emplean *M* — 2 bloques de disco como unidad de bloqueo para las relaciones externas, donde *M* = tamaño de la memoria en bloques; emplear los dos bloques restantes para la relación interna y la salida de la memoria intermedia
    - Coste = $\lceil b_r / (M-2) \rceil * b_s + b_r$ accesos a bloques + $2 \lceil b_r / (M-2) \rceil$ búsquedas
  - Si el atributo de equirreunión forma una clave o relación interna, detener el bucle interno en la primera correspondencia
  - Rastrear el bucle interno hacia delante y hacia atrás, alternativamente, para hacer uso de los bloques restantes en la memoria intermedia (con la sustitución de LRU)
  - Emplear índice en la relación interna si es posible (siguiente transparencia)

Reunión en bucle anidado indexada

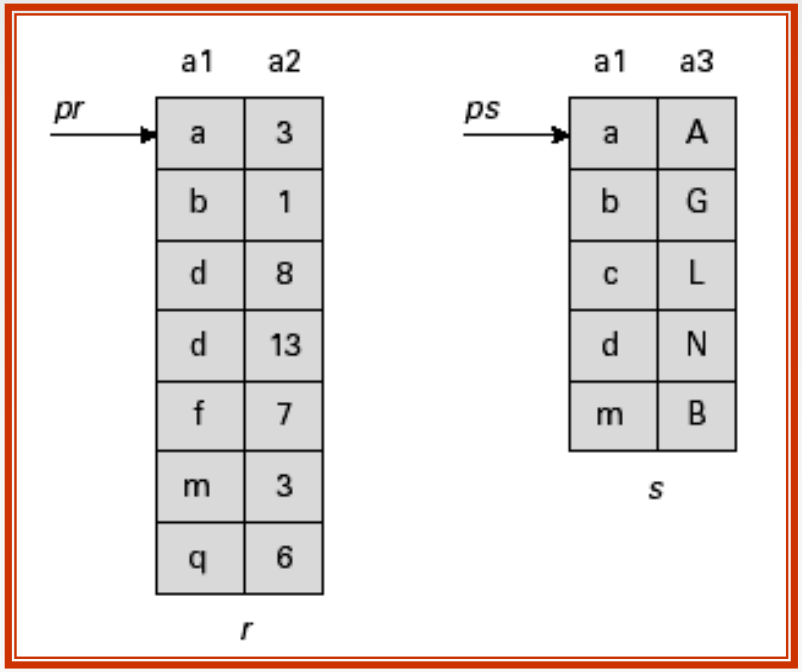
- Las búsquedas de índice pueden reemplazar los exploradores de archivos si
  - la reunión es una equirreunión o reunión natural y
  - está disponible un índice sobre el atributo de la reunión de la relación interna
    - Se puede construir un índice sólo para calcular una reunión.
- Por cada tupla t_r en la relación externa *r,* emplear el índice para buscar las tuplas en *s* que cumplan la condición reunión con la tupla t_r.
- Caso peor: la memoria intermedia sólo tiene espacio para una página de *r* y, por cada tupla en *r*, se realiza una búsqueda de índice sobre *s.*
- Coste de la reunión: $b_r + n_r * c$
  - Donde *c* es el coste de recorrer el índice y tomar todas las tuplas de *s* correspondientes, para una tupla de *r*
  - se puede estimar *c* como el coste de una sola selección sobre *s*, empleando la condición de reunión.
- Si están disponibles los índices sobre los atributos de reunión de *r* y *s*, emplear la relación con menos tuplas cono relación externa.

Ejemplo de costes de reunión en bucle anidado

- Calcular *impositor* ⋈ *cliente,* con *impositor* como relación externa.
- Supóngase que *cliente* tiene un índice primario de árbol B^+ sobre el atributo de reunión *nombre-cliente,* que contiene 20 entradas en cada nodo de índice.
- Dado que *cliente* tiene 10.000 tuplas, la altura del árbol es 4 y se necesita una acceso más para encontrar el dato actual
- *impositor* tiene 5.000 tuplas
- Coste de la reunión en bucle anidado por bloques
  - 400*100 + 100 = 40.100 accesos a disco + 2 * 100 = 200 búsquedas
    - suponemos el peor de los casos de memoria
    - puede ser significativamente menos, con más memoria
- Coste de la reunión en bucle anidado indexada
  - 100 + 5.000 * 5 = 25.100 accesos a disco y búsquedas.
  - El coste de CPU es probable que sea menor que el de la reunión en bucle anidado por bloques

Reunión por mezcla

1. Ordenar ambas relaciones sobre sus atributos de reunión (si no lo están ya).
2. Mezclar las relaciones ordenadas para reunirlas
   1. El paso de la reunión es similar a la fase de mezcla del algoritmo de la mezcla-ordenación.
   2. La principal diferencia es la gestión de valores duplicados en el atributo de reunión — cada par con el mismo valor sobre el atributo de reunión debe ser emparejado
   3. Algoritmos detallados en el libro



Reunión por mezcla (cont.)

- Sólo se puede usar para equirreuniones y reuniones naturales
- Cada bloque necesita ser leído sólo una vez (suponiendo que todas las tuplas, para cualquier valor dado de los atributos de reunión, encajan en memoria)
- Así, el costo de reunión por mezcla es:

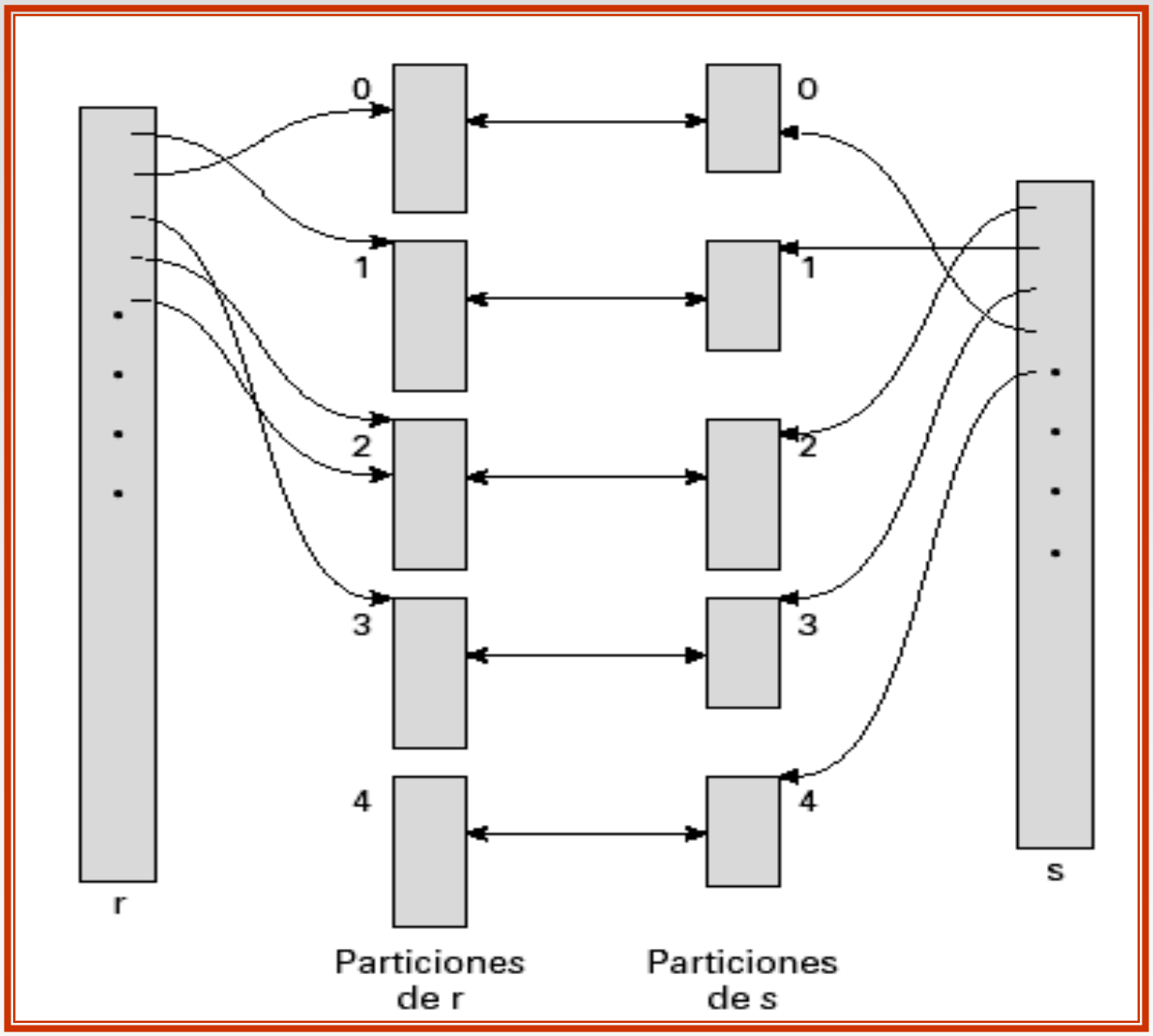
  $b_r + b_s$ accesos de bloques + $\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil$ búsquedas
  - + el coste de ordenación, si las relaciones están desordenadas.
- **reunión por mezcla híbrida:** Si una relación está ordenada y la otra tiene un índice secundario de árbol B^+ sobre el atributo de reunión
  - Mezclar la relación ordenada con las entradas de hojas del árbol B^+.
  - Ordenar el resultado sobre la dirección de las tuplas de la relación desordenada
  - Rastrear la relación desordenada en la dirección física, ordenar y mezclar con el resultado previo, para reemplazar las direcciones por las tuplas actuales
    - El rastreo secuencial es más eficiente que la búsqueda random

Reunión por asociación

- Aplicable a equirreuniones y reuniones naturales.
- Una función de asociación *h* se usa para dividir las tuplas de ambas relaciones
- *h* asocia valores de *AtribsReunión* a {0, 1, ..., *n*}, donde *AtribsReunión* indica los atributos comunes de *r* y *s* empleados en la reunión natural.
  - $r_0, r_1, \ldots, r_n$ denota las particiones de *r* tuplas
    - Cada tupla $t_r \in r$ se mete en la partición r_i donde $i = h(t_r$ *[AtribsReunión]).*
  - $r_0, r_1 \ldots, r_n$ denota particiones de *s* tuplas
    - Cada tupla $t_s \in s$ se mete en la partición s_i donde $i = h(t_s$ *[AtribsReunión]).*

- *Nota:* En el libro, r_i se indica como $H_{ri,}$ s_i se indica como H_{si} y *n* se indica como $n_{h..}$

Reunión por asociación (cont.)



Reunión por asociación (cont.)

- *r* tuplas en r_i sólo necesitan ser comparadas con *s* tuplas en s_i
  No es necesario comparar con *s* tuplas en ninguna otra partición, dado que:
  - una tupla *r* y una tupla *s* que cumplen la condición de reunión, tendrán el mismo valor de los atributos de reunión.
  - Si ese valor está asociado a algún valor *i*, la tupla *r* ha de estar en r_i y la tupla *s* en s_i.

Algoritmo de reunión por asociación

La reunión por asociación de *r* y *s* se calcula como sigue:

1. Dividir la relación *s* empleando la función de asociación *h*. Cuando se divide una relación, se reserva un bloque de memoria para la memoria intermedia de salida por cada partición.
2. Análogamente para la división de *r*.
3. Por cada *i:*
   (a) Cargar s_i en memoria y construir un índice asociativo en memoria sobre él, empleando el atributo de reunión. Este índice asociativo emplea una función de asociación diferente de la *h* anterior.
   (b) Leer las tuplas en r_i desde el disco, una a una. Por cada tupla t_r localizar cada tupla correspondiente t_s en s_i empleando el índice asociativo en memoria. Obtener la concatenación de sus atributos.

La relación *s* se denomina **entrada para construir** y *r* **entrada para probar.**

Algoritmo de reunión por asociación (cont.)

- El valor *n* y la función de asociación *h* se seleccionan de tal manera que cada s_i debería encajar en memoria.
  - Típicamente n se elige como $\lceil b_s/M \rceil * f$ donde *f* es un "factor de escape", generalmente alrededor de 1'2
  - Las divisiones de la relación de prueba s_i no necesitan encajar en memoria
- Se requiere **división recursiva** si el número de divisiones *n*, es mayor que el de páginas de memoria *M*..
  - en vez de dividir de *n* formas, emplear *M* – 1 divisiones para s
  - Dividir a su vez las *M* – 1 particiones, empleando una función de asociación diferente
  - Emplear el mismo método de división sobre *r*
  - Raramente requerido: por ejemplo, la división recursiva no es necesaria para las relaciones de 1GB o menos, con tamaño de memoria de 2MB y tamaño de bloque de 4KB.

Gestión de desbordamientos

- La división se dice que está **sesgada** si algunas divisiones tienen significativamente más tuplas que otras
- El **desbordamiento de una tabla de asociación** ocurre en la división s_i, si s_i no encaja en memoria. La razones podrían ser
  - Muchas tuplas en s con igual valor de atributos de reunión
  - Una mala función de asociación
- La **resolución del desbordamiento** se puede hacer en la fase de construcción
  - La división s_i es a su vez dividida empleando funciones de asociación diferentes.
  - La división r_i debe dividirse en forma similar.
- Se **evita el desbordamiento** llevando a cabo las divisiones cuidadosamente, para evitar desbordamientos durante la fase de construcción
  - Por ejemplo, dividir la relación para construir en numerosas divisiones y a continuación combinarlas
- Ambas técnicas fallan con grandes números de duplicados
  - Opción del último recurso: emplear reunión en bucles anidados por bloques sobre divisiones desbordadas

Coste de la reunión por asociación

- Si no se requiere división recursiva: el coste de la reunión por asociación es
  $3(b_r + b_s) + 4 * n_h$ accesos de bloques+
  $2(\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil)$ búsquedas
- Si se requiere división recursiva :
  - el número esperado de ciclos necesarios para dividir s es $\lceil log_{M-1}(b_s) - 1 \rceil$
  - Mejor elegir la relación más pequeña como relación de creación
  - El coste total estimado es:
    $2(b_r + b_s \lceil log_{M-1}(b_s) - 1 \rceil + b_r + b_s$ transferencias de bloques +
    $2(\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil) \lceil log_{M-1}(b_s) - 1 \rceil$ búsquedas
- Si se puede mantener en memoria toda la entrada necesaria no se requiere la partición
  - El coste estimado baja hasta $b_r + b_s$.

Ejemplo de coste de reunión por asociación

cliente ⋈ *impositor*

- Supóngase que el tamaño de la memoria es de 20 bloques
- $b_{impositor} = 100$ y $b_{cliente} = 400$.
- *impositor* se emplea como entrada para construir. Se divide en cinco partes, cada una de un tamaño de 20 bloques. Estas divisiones se pueden hacer en un ciclo.
- Análogamente, *cliente* se divide en cinco partes de 80 bloques cada una. Esto también se hace en un ciclo.
- Por lo tanto el coste total, ignora el coste de grabar bloques parcialmente llenos :
  - 3(100 + 400) = 1500 transferencias de bloques
    $2(\lceil 100/3 \rceil + \lceil 400/3 \rceil) = 336$ búsquedas

Reunión por asociación híbrida

- Útil cuando el tamaño de la memora es relativamente grande y la entrada para construir es mayor que la memoria.
- **Característica principal de la reunión por asociación híbrida:**

  Mantener en memoria la primera división de la relación para construir.
- Por ejemplo. Con un tamaño de memoria de 25 bloques, *impositor* se puede dividir en cinco partes, cada una con un tamaño de 20 bloques.
  - División de memoria:
    - La primera división ocupa 20 bloques de memoria
    - Se emplea 1 bloque para la entrada y 1 bloque por cada memoria intermedia de las otras 4 divisiones.
- *cliente* está dividido de manera análoga en cinco divisiones de 80 bloques cada una;
  - la primera se emplea inmediatamente para pruebas, en vez de escribirla y leerla de nuevo.
- Coste de 3(80 + 320) + 20 +80 = 1.300 transferencias de bloques por reunión por asociación híbrida, en vez de 1.500 con reunión por asociación sencilla.
- La reunión por asociación híbrida es la más útil si $M >> \sqrt{b_s}$

Reuniones complejas

- Reunión con una condición conjuntiva:

$$r \bowtie_{\theta 1 \wedge \theta 2 \wedge \ldots \wedge \theta n} s$$

  - Se emplean bucles anidados / bucles anidados por bloques, o
  - Calcular el resultado de una de las reuniones más sencillas $r \bowtie_{\theta i} s$
    - el resultado final se compone de esas tuplas, en el resultado intermedio, que cumplen las condiciones restantes

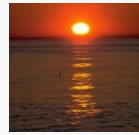
$$\theta_1 \wedge \ldots \wedge \theta_{i-1} \wedge \theta_{i+1} \wedge \ldots \wedge \theta_n$$

- Reunión con una condición disyuntiva

$$r \bowtie_{\theta 1 \vee \theta 2 \vee \ldots \vee \theta n} s$$

  - Se emplean bucles anidados / bucles anidados por bloques, o
  - Calcular como la unión de los registros en reuniones individuales $r \bowtie_{\theta i} s$:

$$(r \bowtie_{\theta 1} s) \cup (r \bowtie_{\theta 2} s) \cup \ldots \cup (r \bowtie_{\theta n} s)$$



Otras operaciones

- La **eliminación de duplicados** se puede implementar por medio de asociación u ordenación.
  - En la ordenación los duplicados aparecerán juntos y se podrán borrar todos menos uno, de los con juntos duplicados.
  - *Optimización:* los duplicados se pueden borrar durante la generación de las secuencias, así como en los pasos intermedios de mezcla, en la mezcla-ordenación externa.
  - La asociación es similar – los duplicados se situarán en el mismo cajón.
- La **proyección**
  - se implementa llevando a cabo la proyección sobre cada tupla
  - seguido de la eliminación de duplicados.

Otras operaciones: Agregación

- La **agregación** se puede implementar de forma similar a la eliminación de duplicados.
  - Se puede emplear ordenación o asociación para llevar las tuplas al mismo grupo juntas y después poder aplicar las funciones de agregación sobre cada grupo.
  - *Optimización:* combinar las tuplas en el mismo grupo durante la generación de las secuencias y las mezclas intermedias, mediante el cálculo ce valores agregados parciales
    - Para contar, mínimo, máximo y suma: mantener valores agregados sobre las tuplas encontradas hasta ahora en el grupo.
      - Cuando se combinan agregaciones parciales para contar, sumar las agregaciones
    - Para el promedio conservar la suma y la cuenta, y dividir al final la suma por la cuenta

Otras operaciones: Operaciones de conjuntos

- **Operaciones sobre conjuntos** ($\cup$, $\cap$ y —): pueden ordenar primero y examinar después para obtener el resultado.
- Ej. Operaciones de conjuntos utilizando asociación:
  1. Dividir las dos relaciones utilizando la misma función de asociación
  2. Procesar cada partición *i* de la siguiente forma.
     1. Utilizar una función de asociación diferente, construir un índice asociativo en memoria sobre r_i.
     2. Procesar s_i de la siguiente forma
        - $r \cup s$:
          1. Añadir las tuplas de s_i al índice asociativo si no estaban presentes.
          2. Al final de s_i añadir las tuplas del índice asociativo al resultado.
        - $r \cap s$:
          1. Pasar la tupla s_i al resultado si ya estaba presente en el índice
        - $r - s$:
          1. Para cada tupla en s_i, si está presente en el índice, suprimirla del índice asociativo.
          2. Al final de s_i añadir las tuplas restantes del índice asociativo al resultado.

Otras operaciones: Reunión externa

- La **reunión externa** se puede calcular como
  - Una reunión seguida por una suma de tuplas que no participan rellenas de nulos.
  - modificando los algoritmos de reunión.
- Modificando la reunión por mezcla para calcular $r \mathbin{⟗} s$
  - En $r \mathbin{⟗} s$, las tuplas que no participan son esas en $r - \Pi_R(r \bowtie s)$
  - Modificando la reunión por mezcla para calcular $r \mathbin{⟗} s$. Durante la mezcla, por cada tupla t_r de r que no se asocia a ninguna tupla de s, resulta t_r rellenada con nulos.
  - La reunión externa por la derecha y la reunión externa completa, se pueden calcular de forma similar.
- Modificando la reunión por asociación para calcular $r \mathbin{⟗} s$
  - Si r es una relación de prueba, la salida no asocia tuplas de r rellenadas con nulos
  - Si r es una relación para construir, cuando se prueba se sigue la pista de las tuplas de r que se corresponden con las de s . Al final de s_i se obtienen las tuplas de r no asociadas, rellenas con nulos

Evaluación de expresiones

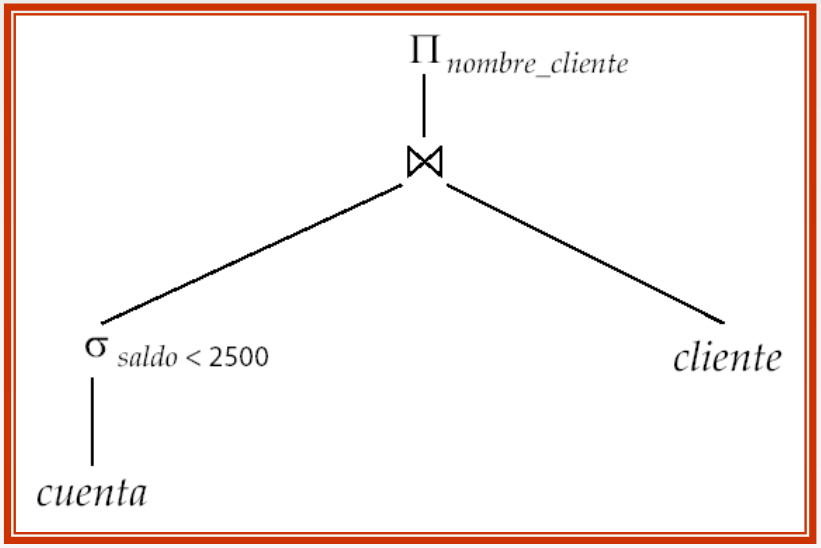
- Hasta ahora: se han visto algoritmos para operaciones individuales
- Alternativas para evaluar un árbol de expresiones completo
  - **Materialización**: generación de resultados de una expresión cuyas entradas son relaciones o ya están calculadas, **materializada** (almacenada) sobre disco. Repetir.
  - **Encauzamiento**: transmitir tuplas para operaciones primarias, incluso cuando se está ejecutando una operación
- Se estudian las alternativas anteriores con más detalle

Materialización

- **Evaluación materializada**: se evalúa una operación cada vez, empezando por el nivel inferior. Se emplean resultados intermedios materializados en relaciones temporales para evaluar operaciones del siguiente nivel.
- Por ejemplo, en la figura siguiente, calcular y almacenar

$$\sigma_{saldo<2500}(cuenta)$$

entonces calcula para almacenar su reunión con *cliente* y finalmente calcular las proyecciones sobre *nombre-cliente.*



Materialización (cont.)

- La evaluación materializada siempre es aplicable
- El coste de grabar los resultados sobre disco y de volver a leerlos puede ser muy alto
  - Nuestras fórmulas de costes para operaciones no tienen en cuenta el coste de grabar resultados sobre disco, así
    - Coste global = Suma de costes de la operaciones individuales + coste de grabar resultados intermedios sobre disco
- Memoria intermedia doble: se emplean dos memorias intermedias de salida por cada operación, cuando una está llena la graba en disco mientras la otra se va llenando
  - Permite solapar escrituras en disco con cálculo y reduce el tiempo de ejecución

Encauzamiento

- **Evaluación del encauzamiento**: se evalúan varias operaciones simultáneamente, pasando los resultados de una operación a la siguiente.
- Por ejemplo, en el árbol de la expresión anterior, no se almacena el resultado de

$$\sigma_{saldo < 2500}(cuenta)$$

  - en su lugar, las tuplas pasan directamente a la reunión. Análogamente, no se almacena el resultado de la reunión, pasan directamente las tuplas a proyección.
- Mucho más barato que la materialización: no es necesario almacenar una relación temporal en el disco.
- El encauzamiento puede no ser siempre posible – por ejemplo, ordenar, reunión por asociación.
- Para que el encauzamiento sea efectivo, emplear algoritmos de evaluación que generan tuplas de salida, incluso cuando las tuplas se reciben de las entradas para la operación.
- Los encauzamientos se pueden ejecutar de dos maneras: demanda impulsada y productor impulsado

Encauzamiento (cont.)

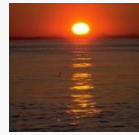
- En un encauzamiento **bajo demanda** o evaluación **perezosa**
  - El sistema solicita repetidas veces la siguiente tupla desde la operación superior
  - Cada operación solicita la tupla siguiente de las operaciones de los hijos según se necesiten, a la hora de producir su siguiente tupla
  - Entre llamadas, la operación ha de mantener el "**estado**", por lo que sabe qué devolver a continuación
- En encauzamiento por los **productores** o encauzamiento **impaciente**
  - Los operadores producen tuplas impacientemente y las pasan a sus padres
    - Memoria intermedia mantenida entre operadores, el hijo pone tuplas en memoria intermedia, el padre elimina tuplas de la memoria intermedia
    - si la memoria intermedia está llena, el hijo espera hasta que haya espacio en la memoria intermedia y, entonces, genera más tuplas
  - Operaciones planificadas del sistema que tienen espacio en la memoria intermedia de salida y pueden procesar más tuplas de entrada
- Nombres alternativos: **pull** y **push** como modelos de encauzamiento

Encauzamiento (cont.)

- Implementación de encauzamiento bajo demanda
  - Cada operación se implementa como un **iterador**, implementando las operaciones siguientes
    - abrir()
      - Por ejemplo, explorador de archivo: inicializar el explorador de archivo
        - » estado: el puntero al principio del archivo
      - Por ejemplo, reunión por mezcla: ordenar relaciones;
        - » estado: el puntero al principio de la relación a ordenar
    - siguiente()
      - Por ejemplo, explorador de archivo: Obtiene la siguiente tupla y avanza y almacena el puntero del archivo
      - Por ejemplo, reunión por mezcla: continua con la mezcla desde el estado anterior hasta que se encuentra la siguiente tupla de salida. Salvar punteros como estado del iterador
    - cerrar()

Algoritmos de evaluación para encauzamiento

- Algunos algoritmos no son capaces de producir resultados, incluso cuando tienen tuplas de entrada
  - Por ejemplo, reunión por mezcla o por asociación
  - Concluyen en resultados intermedios que se graban en disco y después siempre se vuelven a leer
- Son posibles variantes de algoritmos para generar (al menos algún) resultados al instante, en cuanto las tuplas de entrada son leídas
  - Por ejemplo, la reunión por asociación híbrida genera tuplas de salida, incluso como tuplas de relaciones de pruebas se leen en la partición de memoria (división 0)
  - **Técnica de reunión encauzada**: La reunión por asociación híbrida, modificada para las tuplas de la división 0 de la memoria intermedia de ambas relaciones en memora, leyéndolas cuando se hacen disponibles y obteniendo los resultados de cualquier correspondencia entre las tuplas de la división 0
    - Cuando se encuentra una nueva tupla r_0, se asocia con la tuplas existentes s_0, se obtiene la equivalente y se guarda en r_0
    - Análogamente para las tuplas s_0



Fin del Capítulo