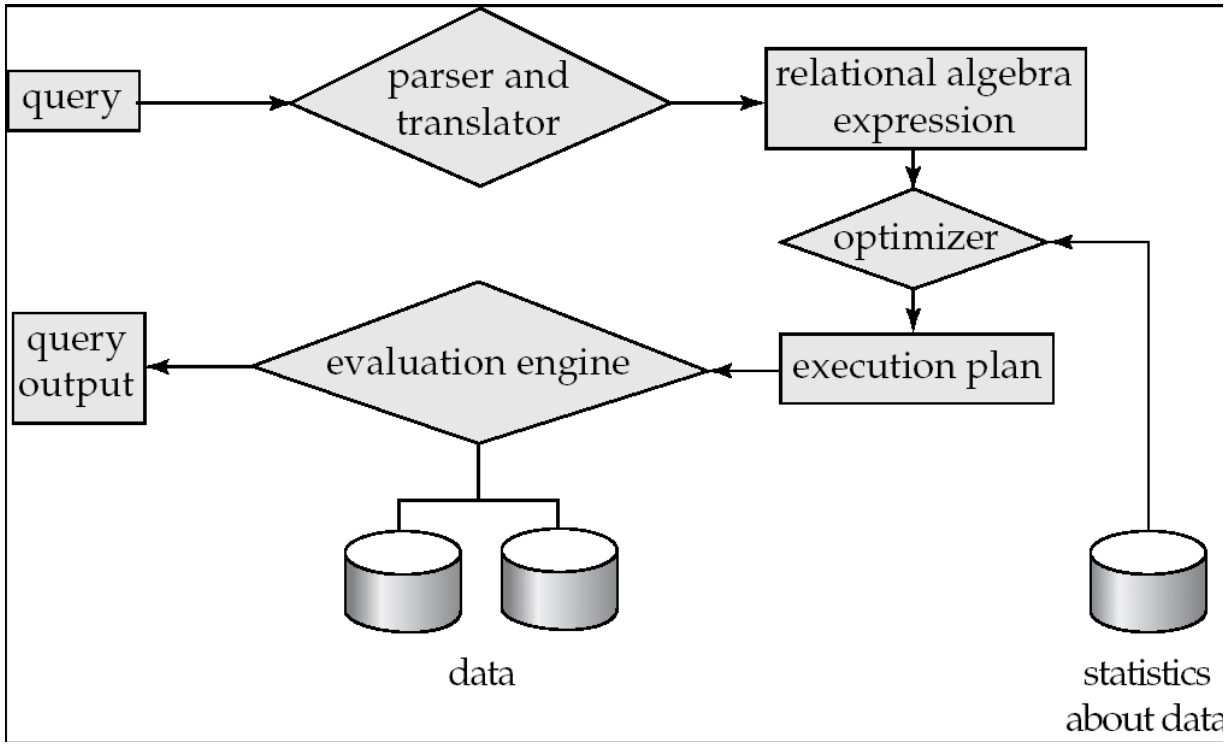
Teórico 6:

Procesamiento de Consultas

Procesamiento de consultas

Los pasos básicos en el procesamiento de consultas son:

1. Parsing y traducción.
2. Optimización.
3. Evaluación.



Parsing y Traducción

- Chequeo sintáctico y la verificación de que las relaciones intervinientes en la consulta sean relaciones de la base de datos, así como campos, etc.
- Traducir el query (SQL) a una expresión del álgebra relacional.
- Si la expresión posee vistas, la traducción reemplaza las vistas por su definición.

Optimización

- Las expresiones del álgebra relacional pueden tener muchas expresiones equivalentes, por ejemplo:

$$\sigma_{precio=100}(\Pi_{precio}(articulo)) \equiv \Pi_{precio}(\sigma_{precio=100}(articulo))$$

- Cada expresión del álgebra relacional puede ser evaluada utilizando diferentes algoritmos.

- Las expresiones que tienen especificado como deben ser evaluadas, se denominan plan de evaluación. Por ejemplo la expresión $\sigma_{precio=100}(articulo)$ puede ser evaluada de la siguiente manera:
  - Si hay definido un índice sobre *precio* en la tabla *articulo*, se puede usar ese índice para recuperar rápidamente la tuplas que cumplen con la condición de precio =100.
  - Si no hay un índice definido se deberá recorrer toda la tabla *articulo* y descartar las tuplas con precio <> 100.

Optimización (Cont..)

- **Optimización de Query**: Entre los planes de evaluación equivalentes, se elige el de menor costo.
  - Los costos son estimados utilizando información estadística del catalogo de la base de datos. Por ejemplo número de tuplas de la relación, tamaño de la tupla, etc.

Evaluación

- El motor de ejecución de la consultas toma un plan de evaluación, lo ejecuta y retorna el resultado.

Medidas de costo de consultas

- El costo de una consulta es básicamente el tiempo que el motor tarda en retornar el resultado, Muchos factores influyen en este tiempo. Por ejemplo: Acceso a disco, potencia de la CPU, Velocidad de la red, Memoria RAM, etc.

- El tiempo de acceso a disco es el costo más importante, y es fácil de estimar. Para calcular se tiene en cuenta lo siguiente:

  Número de búsquedas * costo promedio de búsqueda.

  +

  Número de bloques leídos * costo promedio de lectura de un bloque.

  +

  Números de bloques escritos * costo promedio de escritura de bloque.

Medidas de costo de consultas (Cont.)

- Para simplificar el cálculo sólo se usará el número de bloques transferidos desde discos y el número de búsquedas como medidas de costo.
  - t_T – tiempo de transferencia de un bloque
  - t_b – tiempo de búsqueda
  - El costo para transferir b bloques más s búsquedas: $b * t_T + s * t_b$
- Por simplicidad no se tiene en cuenta el costo de CPU, ni se incluye el costo de escribir en disco el resultado.
- Algunos datos que se requieren pueden que estén en memoria, pero esto depende de la carga del sistema, esto no se tiene en cuenta en el costo. Se asume que todo dato requerido se debe obtener del disco.
- Se asume lo siguiente:
  - t_b = 4 miliseg.
  - t_T = 0.1 miliseg.
  - Tamaño de bloque de 4 Kb
  - Velocidad de transferencia 40 Mb/s

Estimación de costo Operación de Selección

- A continuación se listan algoritmos que buscan y recuperan registros que cumplen la condición de la selección
- Algoritmo **A1** (*búsqueda lineal*). Busca cada bloque del archivo y se comprueban todos los registros para determinar si satisfacen o no la condición. Si la condición es sobre un atributo clave la búsqueda termina cuando se encuentra.
  - Costo estimado = b_r bloques transferidos + 1 búsqueda

    O sea $= b_r \ t_t + t_b$
    - b_r indica el número de bloques que contienen registros de la relación r.
  - Si la selección es sobre un atributo clave, la búsqueda puede parar cuando se cumple la condición.
    - costo = (b_r/2) bloques transferidos + 1 búsquedas
  - La búsqueda lineal puede ser aplicada sin importar la:
    - Condición de selección o
    - El ordenamiento de los registros en el archivo, o
    - Disponibilidad de índices

Operación de selección: Algoritmo A2

- **A2** *(Búsqueda binaria).* Es aplicable sólo si el archivo está ordenado sobre un atributo y la condición es de igualdad sobre ese atributo.
  - Costo estimado:
    - Costo de localizar la primera tupla con una búsqueda binaria sobre los bloques del archivo

$$\lceil \log_2(b_r) \rceil * (t_T + t_b)$$

    - Si el atributo no es clave se debe agregar el costo de transferencia del número de bloques que contienen los registros que satisfacen la condición.

Selección utilizando Índices

Estos algoritmos utilizan índices definidos sobre atributos (por ejemplo B+)

- **A3: Utilizado en condiciones de igualdad sobre índices de clave primaria o única.**
  - Recupera un único registro que satisface la condición.
  - El costo es la altura del árbol (si utiliza árbol B+), más un acceso para recuperar el registro, esto es:
    - Costo = $(h_i + 1) * (t_T + t_b)$, donde h_i es la altura del índice.

- **A4: Utilizado en condiciones de igualdad sobre índices primarios no claves, con lo cual puede recuperar más de un registro.**
  - Los registro que satisfacen la condición están en registros consecutivos
  - *El costo es:*
    - *Costo* = $h_i * (t_T + t_b) + t_b + t_T * b$

      Donde b indica la cantidad de bloques que contiene registros que cumplen la condición.

- **A5: Utilizado en condiciones de igualdad sobre claves de búsqueda en índices secundarios**
  - Si la clave de búsqueda es única, recupera un único registro, donde el costo será:
    - *Costo* = $(h_i + 1) * (t_T + t_b)$
  - Si la clave de búsqueda no es única , puede recuperar más de un registro
    - El peor caso es que cada registro que cumple la condición este en un bloque diferente, con lo cual el costo es:
    - Costo = $(h_i + n) * (t_T + t_b)$ *donde n es la cantidad de registros que cumplen la condición*

Condiciones con Rango

- Implementación de selecciones con condición con rango, selecciones de la forma $\sigma_{A \leq V}(r)$ o $\sigma_{A \geq V}(r)$ utilizando:
  - Algoritmo de búsqueda lineal o binaria,
  - Utilizando los algoritmos A6 y A7 que utilizan índices:
- **A6:** utiliza índices primarios sobre el atributo A.
  - Para $\sigma_{A \geq V}(r)$ utiliza el índice para encontrar el primer registro con $A \geq v$ y luego recuperar secuencialmente el resto de los registros.
  - Para $\sigma_{A \leq V}(r)$ recuperar los registros secuencialmente hasta encontrar la primera tupla que cumpla que $A > v$; no es necesaria la utilización de índices.
- **A7** utiliza índices secundarios.
  - para $\sigma_{A \geq V}(r)$ utiliza índices para encontrar la primera entrada que cumpla $A \geq v$ y luego recorrer el índice para encontrar todos los punteros a los registros.
  - Para $\sigma_{A \leq V}(r)$ escanear todos los nodos hojas del índice para encontrar los punteros a los registros, esto hasta encontrar el primero que cumpla $A > v$
  - Para todo los casos, se recuperan los registros que son apuntados
    - Requiere un operación de I/O por cada registro
    - El escaneo lineal puede ser más conveniente

Selecciones complejas

Conjunción: $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \theta n}(r)$

- **A8** (**Conjunciones** utilizando un índice*).*
  - Seleccionar una combinación θ_i y uno de los algoritmos A1 a A7 que resulten en el menor costo para $\sigma_{\theta i}$ (*r).*
  - Testear en los registros obtenidos en el punto anterior el resto de las condiciones.
- **A9** (conjunción utilizando índice múltiple clave).
  - Se utiliza si hay disponible un índice multiclave apropiado para la condición.
- **A10** (conjunción por intersección de identificadores*).*
  - Requiere de índices con punteros a registros.
  - Usar los índices apropiados para condición y hacer la intersección de todos los punteros obtenidos por cada condición.
  - Buscar los registros en el archivo
  - Si para alguna condición no se pudo utilizar un índice, se debe chequear la condición sobre los registros obtenidos.

Selecciones Complejas (sigue)

Disyunción: $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \theta n}(r)$.

- **A11** (disyunción por unión de identificadores).
  - Es aplicable si todas las condiciones tienen un índice disponible.
    - Para otros casos usar la búsqueda lineal.
  - Usar el índice definido disponible para cada condición, luego hacer la unión de los conjuntos de punteros obtenidos.
  - Buscar los registros en los archivos.

Ordenamiento

- Para ordenar una relación por algún atributo dado, se puede crear un índice sobre los atributos a ordenar y luego recuperar las tuplas de manera ordenada. Esto puede tener un costo de un acceso a bloque de disco por cada tupla a recuperar.

- Para relaciones que entran en memoria, se pueden utilizar técnicas de ordenamientos como de quicksort.

- Para relaciones que no entran en memoria, se puede utilizar la técnica **external Sort-merge**.

External Sort-Merge

M es el número de bloques de disco que pueden almacenarse en memoria.

1. *"Se ordena la relación de a partes que caben en memoria"*

   I=0;

   repeat

   leer M bloques de la relación y ponerlos en memoria;

   Ordenar los registros que están en memoria;

   Escribir los datos en una secuencias S_i *;*

   I = i +1;

   hasta *procesar toda la relación*

1. *Mezclar las secuencias S obtenidas en el paso anterior.*

External Sort-Merge (Cont.)

1. **Mezclar la secuencias (N-way merge)**. Se asume que *N* < *M,* donde *N* es el número de secuencias obtenidos en el paso anterior *y M* la cantidad de bloques que entran en memoria.

   a. Usar *N* bloques de memoria como buffers de entrada, y 1 bloque para buffer de salida. Leer el primer bloque de cada secuencia y ponerlo en los buffer de entrada.

   b. **repeat**

      1. Seleccionar el primer registro de todos los buffer
      2. Escribir el registro en el buffer de salida. Si el buffer se llena escribirlo a disco.
      3. Borrar el registro del buffer de entrada. **IF** el buffer de entrada está vacío **then**

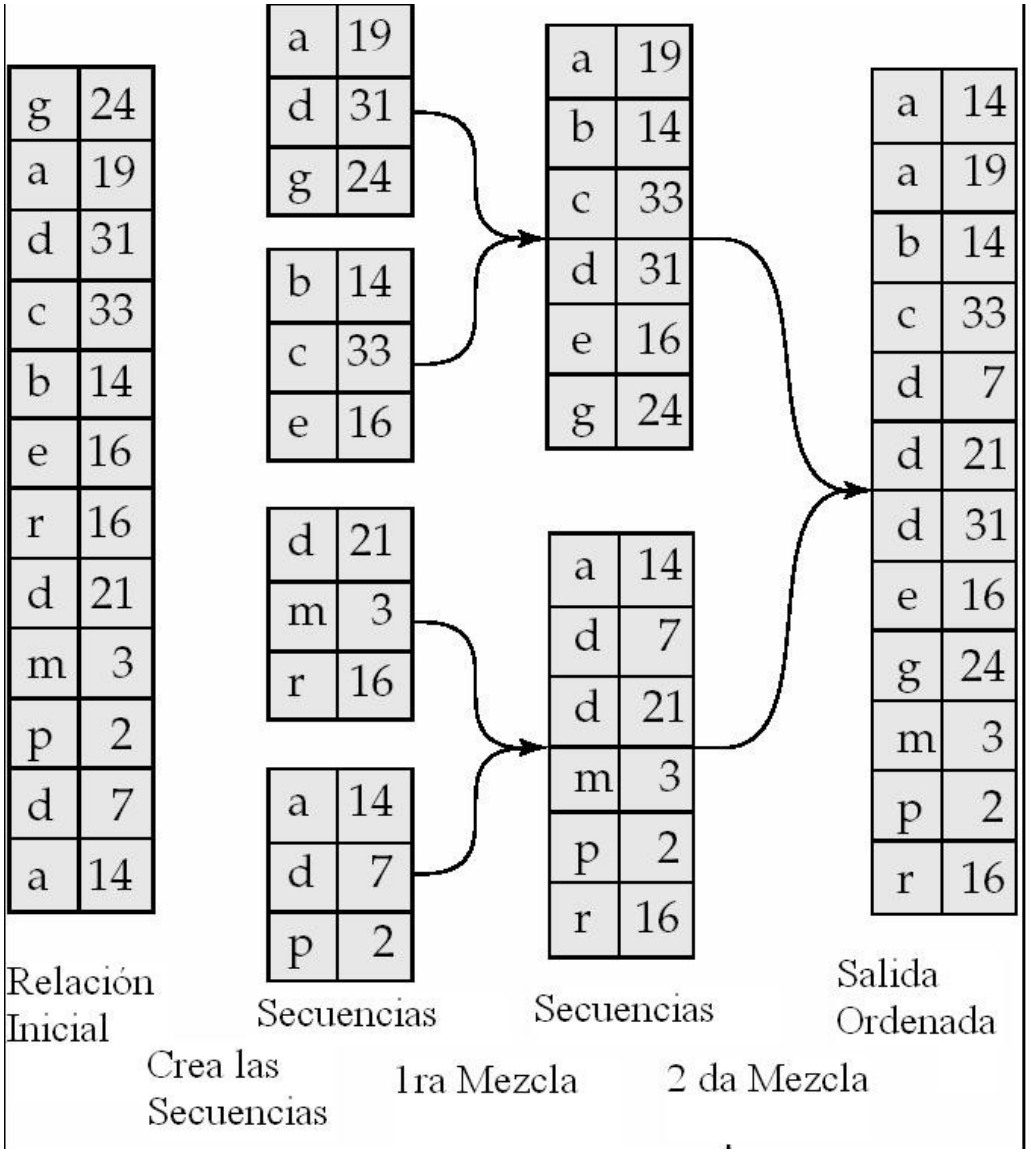
         leer el próximo bloque de la secuencia y ponerlo en el buffer de entrada.

   until todos los buffer estén vacíos.

External Sort-Merge (Cont.)

- Si $N \geq M$, se requieren varios pasos de mezcla.
  - En cada paso, grupos contiguos de M - 1 son mezclados.
  - Un paso reduce el número de corridas en por un factor de M -1, y crea corridas por el mismo factor.
  - Se repiten los pasos hasta que todas las corridas han sido mezcladas en una sola.

Ejemplo: de ordenamiento externo



Operación Join

- Hay varios algoritmos para implementar join:
  - Join en ciclo anidado.
  - Join en ciclo anidado por bloques.
  - Join en ciclo anidado indexada.
  - Merge-join.
  - Hash-join.
- La elección está basada en los costos
- Los ejemplos mostrados a continuación trabajan con la siguiente información:
  - Número de registros de cliente: 10,000
  - Número de bloques de cliente: 400
  - Número de registros de factura: 5,000
  - Número de bloques de factura: 100

Join de ciclo anidado

- Para computar el join $r \bowtie_\theta s$

  for each tuple t_r **in** r **do begin**

  for each tuple t_s **in** s **do begin**

  testear el par (t_r,t_s) para chequear que satisfagan la condición θ

  si la condición se satisface, agregar $t_r \bullet t_s$ al resultado.

  end

  end

- r se denomina relación externa y s la relación interna del join.
- No utiliza índices y puede ser aplicada a cualquier condición.
- Muy caro debido a que examina cada pareja de tuplas de ambas relaciones.

Estimación de costo con ciclo anidado

- En el peor caso, la memoria puede contener sólo un bloque de cada relación, necesitándose $n_r * b_s + b_r$ transferencias de bloques más $n_r + b_r$ búsquedas

  Donde b_s y b_r indican la cantidad de bloques que contienen tuplas de r y s respectivamente y n_r indica la cantidad de tuplas de la relación r, en la segunda formula es n_r, porque se necesita leer n_r veces la relación interna (s) y b_r búsquedas para encontrar los bloques de r.

- Si la relación con menos espacio utilizado entra en memoria, conviene tomar a está, como relación interna.
  - El costo se reduce a $b_r + b_s$ bloques transferidos más 2 búsquedas

- Si las dos relaciones caben en memoria, el costo es el mismo que el caso anterior.

Ejemplo

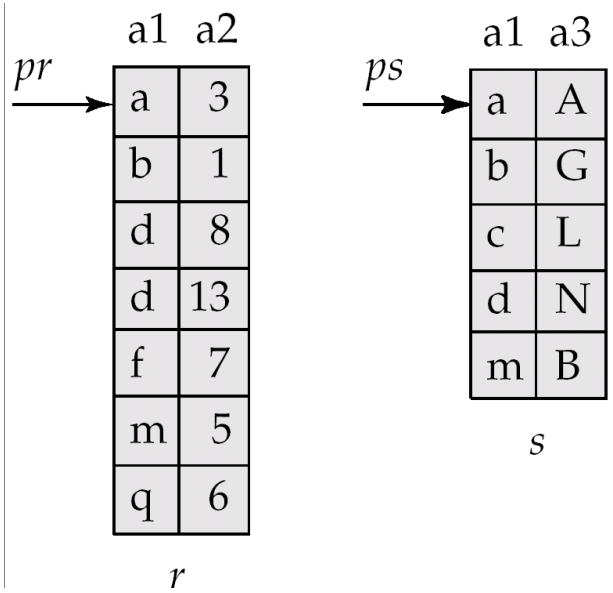
- Por ejemplo, asumiendo el peor caso la estimación de costo para cliente $\bowtie$ factura
  - Con *factura como* relación externa:
    - $5000 * 400 + 100 = 2{,}000{,}100$ bloques a transferir,
    - $5000 + 100 = 5100$ búsquedas
  - Con *cliente* como relación externa
    - $10000 * 100 + 400 = 1{,}000{,}400$ bloques a transferir.
    - $10000 + 400 = 10{,}400$ búsquedas
- Si la relación más pequeña (factura*)* cabe en memoria, el costo estimado será de 500 transferencias de bloques.

Nota: Recordar Número de registros de cliente: 10,000

- Número de bloques de cliente: 400
- Número de registros de factura: 5,000
- Número de bloques de factura: 100

Merge-Join

1. Ordenar ambas relaciones por los atributos del join.
2. Mezclar las relaciones ordenadas para hacer el join:
    1. Es similar al algoritmo de ordenamiento por mezla (sort-merge).
    2. Se recorre las relaciones una sola vez.
    3. Se puede utilizar en join por igualdad.



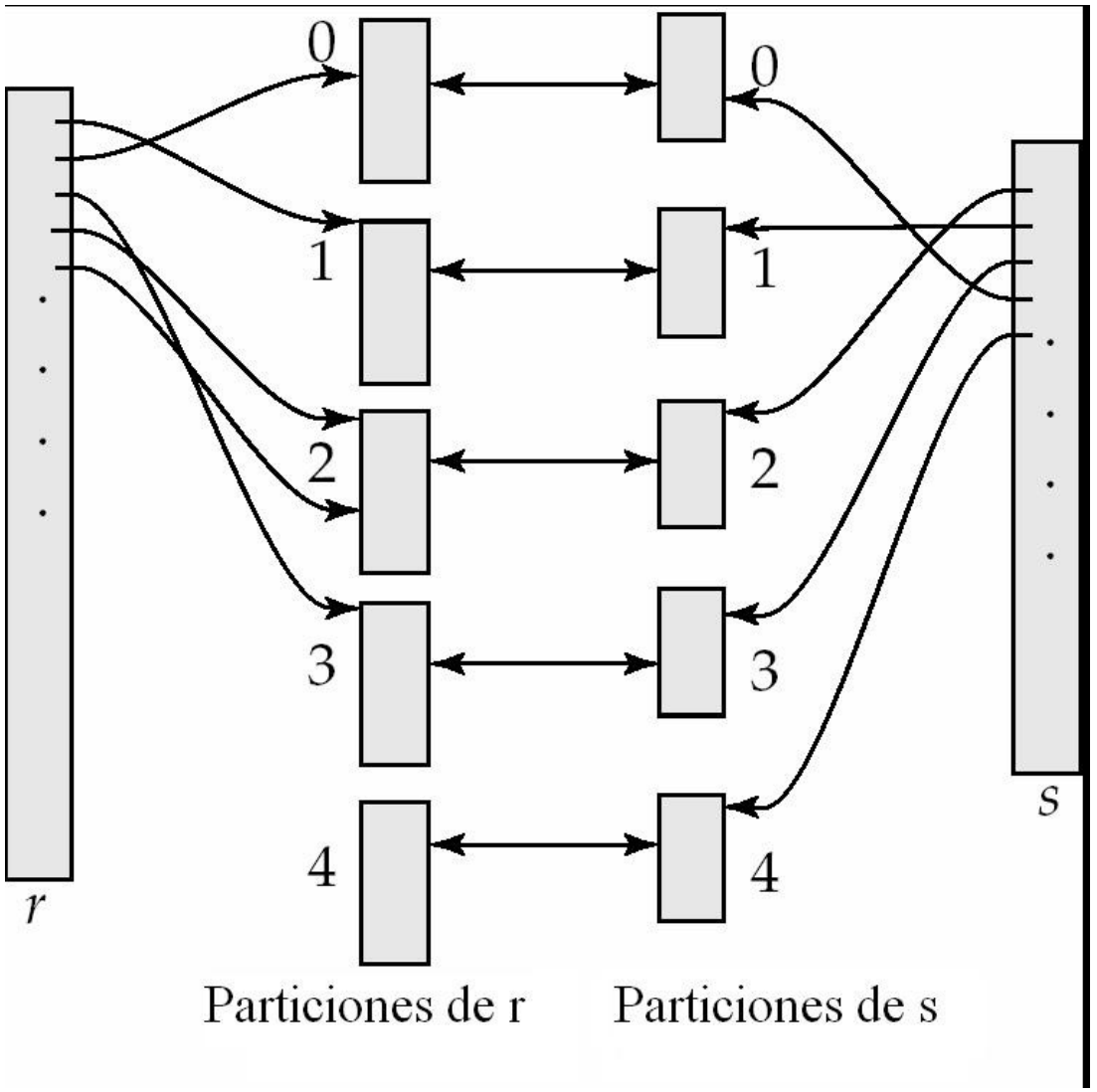
Hash-Join

- Es aplicable para join por igualdad y naturales.
- Una función de hash *h* se utiliza para particionar las tuplas de las dos relaciones
  - La idea es que las particiones entren en Memoria.
- *h* mapea los atributos del join (*JoinAttrs*) a valores en el conjunto {0, 1, ..., *n*}.
  - $r_0, r_1, \ldots, r_n$ denotan particiones de las tuplas de *r.*
    - Cada tupla $t_r \in r$ *es puesta en la* partición r_i donde

      $i = h(t_r[JoinAttrs])$.
  - $s_0, s_1, \ldots, s_n$ denota particiones de las tuplas de *s*
    - Cada tupla $t_s \in s$ es puesta en la partición s_i, donde

      $i = h(t_s[JoinAttrs])$.

Hash-Join (Cont.)



Hash-Join (Cont.)

- La tuplas de *r* en r_i necesitan ser compradas sólo con las tuplas de *s* en s_i .No necesitan ser comparadas con ninguna otras tupla de s.

Operaciones sobre conjuntos

- **Operaciones de conjuntos** ($\cup$, $\cap$ y —): se puede usar una variante de merge-join, o una variante de hash-join.

Evaluación de Expresiones

- Hasta ahora se han visto los algoritmos para las operaciones individuales
- Las alternativas para la evaluación de expresiones compuestas
  - **Materialización**: los resultados generados por una expresión cuyas entradas son relaciones o resultados intermedios son **materializados** (almacenados) en disco. Así hasta que se obtiene el resultado final.
  - **Pipelining**: pasan las tuplas que se van generando en una operación a la operación que toma como parámetro al resultado de esta.