Capítulo 3

Operadores sobre tablas y su implementación

Visión general

- Vamos a pensar una **tabla** como una lista de tuplas.
- Vamos a definir un **álgebra de tablas**.
  - Un **operador del álgebra de tablas** es una función que recibe tablas (que cumplen ciertas condiciones) y devuelve una tabla.
- Para **procesar una consulta**, una sentencia de SQL se puede traducir a una **expresión del álgebra de tablas**.
  - Esta expresión involucra el uso de operadores del álgebra de tabla.
  - La expresión es una composición de operadores del álgebra de tablas.
  - Así una consulta SQL se traduce a operaciones más sencillas a ser evaluadas en un cierto orden.
- Luego de la traducción, se puede evaluar la consulta del álgebra de tablas.

Visión general

- Los operadores del álgebra de tablas se llaman **operadores lógicos.**
  - Estos son definidos usando recursión.
- Un **operador físico** es un algoritmo específico usado para implementar un operador lógico.
  - Los operadores físicos hacen uso de informaciones, como índices, tamaños de búfer en memoria, técnicas avanzadas de algorítmica, etc.
- La evaluación de una expresión del álgebra de tablas va a ser en términos de operadores físicos.

Visión general

- **Para cada operador del álgebra de tablas vamos a estudiar:**
  - Su definición
  - Implementación del operador de usando operadores físicos
    - Aquí se incluye la estimación del costo de ejecutar el operador físico
  - Estimación de tamaño de almacenamiento del resultado de evaluar un operador físico.

Visión general

- Hace falta medir el costo de un operador del álgebra de tablas, y también medir el costo de una consulta del álgebra de tablas.
- Para ellos se contará el número de **transferencia de bloques** de disco.
  - Por simplicidad de las cuentas vamos a asumir que todas las transferencias de bloques tienen el mismo costo.
  - No distinguiremos entre las transferencias de bloques de lectura y las de escritura (a pesar de que se demora más en escribir un bloque que leerlo de disco)

Visión general

- Además, se puede contar la cantidad de **accesos a bloques:**
  - El tiempo de un acceso a bloque es el tiempo que le lleva a la cabeza lectora posicionarse en un bloque deseado.
- El tiempo de un acceso a bloque suele ser de alrededor de 4 mseg.
- El tiempo de transferencia de bloques suele ser de 0,1 mseg, asumiendo un tamaño de bloque de 4 KiB y una tasa de transferencia de 40 MB por segundo.

Visión general

- Los costos de todos los algoritmos físicos dependen del **tamaño del búfer** en memoria principal.
  - En el mejor caso todos los datos pueden ser leídos en búfer y el disco no necesita ser accedido de nuevo.
  - En el peor caso asumimos que el búfer puede sostener solo unos pocos bloques de datos – aproximadamente un bloque por tabla.
- Cuando presentamos estimaciones de costos, asumimos generalmente el peor caso.
- Vamos a ignorar los costos de CPU por simplicidad.

Tablas y esquemas

- Vieron que un esquema relacional es una lista de nombres de atributos y usaron la notación.
- R = (A1,…, An)
- Una forma más refinada de dar un esquema que adoptamos es:
  - R = (A1::T1, …, An:: Tn)
  - Ti es tipo para valores de Ai
  - Dom(R) = T1 x…x Tn
- Vieron que r(A1,…,An) significa que la tabla r tiene esquema (A1,…,An).
- Para dar más detalle ponemos: r(A1::T1, …, An:: Tn)

nombre			
$campo_1 :: \tau_1$	$campo_2 :: \tau_2$	$\ldots$	$campo_N :: \tau_N$
v_{11}	v_{12}	$\ldots$	v_{1N}
v_{21}	v_{22}	$\ldots$	v_{2N}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
v_{M1}	v_{M2}	$\ldots$	v_{MN}

Tablas y esquemas

- **Ejemplo**: Equemas para: biblioteca, bibliotecario, trabajaEn

  biblioteca(nombreBib:: string, calle::string, número: integer)

  bibliotecario(dni:: integer, antigüedad: integer)

  trabajaEn(dni::integer, nombreBib::string)

- **Ejemplo**: tuplas de las tablas anteriores son:

  (“FaMAF”, “Medina Allende”, 0) ∈ biblioteca

  (1232, 5) ∈ bibliotecario

  (1232, “FaMAF”) ∈ trabajaEn

- **Ejemplo**: biblioteca = [(“FaMAF”, “Medina Allende”, 0)]

Proyección

- Relación *r*

A	*B*	*C*
α	10	1
α	20	1
β	30	1
β	40	2

- Selecciona A and C
  - proyección
  - $\Pi_{A, C}$ (r)

A	*C*
α	1
α	1
β	1
β	2

¿Cómo definir recursivamente la operación que acabamos de hacer?

Proyección

- Relación *r*

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- Selecciona A and C
  - proyección
  - $\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

$\Pi_{A,C}[] = []$

$\Pi_{A,C}(t{:}r) = (t.A, t.C) : (\Pi_{A,C}\ r)$

$= ((\backslash t' \rightarrow (t'.A, t'.C))\ t) : (\Pi_{A,C}\ r)$

$\Pi_{A,C} = map\ (\backslash t' \rightarrow (t'.A, t'.C))$

Proyección Generalizada

profe			
legajo	nombres	apellidos	sueldo
p1	"Benjamin"	"Pierce"	3000
p2	"Patricia"	"Selinger"	6000
p3	"Edgar F"	"Codd"	5500
p4	"Barbara"	"Liskov"	5600

Hacer la consulta: obtener los sueldos anuales de los profesores.

sueldos_anuales	
legajo	_
p1	39000
p2	78000
p3	71500
p4	72800

¿Esto es una proyección?

Proyección Generalizada

profe			
legajo	nombres	apellidos	sueldo
p1	"Benjamin"	"Pierce"	3000
p2	"Patricia"	"Selinger"	6000
p3	"Edgar F"	"Codd"	5500
p4	"Barbara"	"Liskov"	5600

Hacer la consulta: obtener los sueldos anuales de los profesores.

sueldos_anuales	
legajo	_
p1	39000
p2	78000
p3	71500
p4	72800

- **No** porque 'sueldos anuales' no es atributo de la tabla.
- Pero se parece mucho a proyección.
- Por eso **generalizaremos** la proyección para capturar situaciones como esta.

Proyección Generalizada

profe			
legajo	nombres	apellidos	sueldo
p1	"Benjamin"	"Pierce"	3000
p2	"Patricia"	"Selinger"	6000
p3	"Edgar F"	"Codd"	5500
p4	"Barbara"	"Liskov"	5600

Hacer la consulta: obtener los sueldos anuales de los profesores.

sueldos_anuales	
legajo	_
p1	39000
p2	78000
p3	71500
p4	72800

$$\text{sueldos\_anuales} = \Pi_{\text{legajo},\ \text{sueldo}*13}(\text{profe})$$

legajo = (\t -> t.legajo)

sueldo * 13 = (\t -> t.sueldo * 13)

Proyección generalizada

- Generalizando:

  $\Pi_{f1,\ldots,fn}\ [] = []$

  $\Pi_{f1,\ldots,fn}\ (t:r) = (f1(t), \ldots, fn(t)) : \Pi_{f1,\ldots,fn}\ (r) = ((\backslash t' \rightarrow (f1(t'),\ldots,fn(t'))\ t) : \Pi_{f1,\ldots,fn}\ (r)$

- O sea que tiene la forma de un map.
- $\Pi_{f1,\ldots,fn} = map\ (\backslash t' \rightarrow (f1(t'),\ldots,fn(t')))$

Operación de proyección

- **Operador físico para proyección**
  - Requiere recorrer todos los registros y realizar una proyección en cada uno.
  - Se recorren todos los bloques de la tabla.
  - Estimación de costo = b_r transferencias de bloques + 1 acceso a bloque
    - b_r denota el número de bloques conteniendo registros de la tabla *r*
  - La proyección generalizada puede ser implementada de la misma manera que la proyección.

Selección

- Sean las tablas:

profe			
legajo	nombres	apellidos	sueldo
p1	"Benjamin"	"Pierce"	3000
p2	"Patricia"	"Selinger"	6000
p3	"Edgar F"	"Codd"	5500
p4	"Barbara"	"Liskov"	5600

curso		
id	legajo	nombre
c1	p2	"Optimización de Consultas"
c2	p3	"Fundamentos de BD"
c3	p2	"Análisis de Datos"
c4	p1	"Fundamentos del Software"
c5	p4	"Programación OO"

- Sea la consulta: obtener los cursos enseñados por Patricia

cursos_patricia		
id	legajo	nombre
c1	p2	"Optimización de Consultas"
c3	p2	"Análisis de Datos"

$$\text{cursos\_patricia} = \sigma_{\text{legajo}=\text{p2}}(\text{curso})$$

Selección

- Vamos a aplicar la selección a predicados.
- Los **predicados** son funciones booleanas.
- En el ejemplo anterior:
- Legajo = p2 =def (\t -> t.legajo == p2)
- Esto se puede generalizar más aun:
- A rel v =def (\t -> t.A rel v) donde A atributo y v valor constante.
- A1 rel A2 = def (\t -> t.A1 rel t.Ad), donde A1 y A2 atributos.
- Donde rel ∈ {<, >, ==, ….}
- Las anteriores son **predicados básicos o atómicos**.

Selección

- Los predicados básicos se pueden combinar usando **conectivos booleanos.**
- El conjunto de conectivos que consideramos es:
- con ∈ {&&, !!, not}
- **Por ejemplo**:
- A > v && A1 == A2   = def (\t -> t.A > v && t.A1 == t.A2)
- **Ahora que sabemos qué son los predicados, ¿cómo se puede definir el operador de selección recursivamente?**

Selección

- El operador de selección se define recursivamente como sigue:
- $\sigma_P\ [] = []$
- $\sigma_P\ (t{:}r) =$ if $p(t)$ then $t : (\sigma_P\ r)$ else $\sigma_P\ r$

Operadores físicos para selección

- **Algoritmo de búsqueda lineal**
  - Escanear cada bloque del archivo y testear todos los registros para ver si satisfacen la condición de selección.
  - Estimación de costo = b_r transferencias de bloques + 1 acceso a bloque
    - b_r denota el número de bloques conteniendo registros de la tabla *r*

Operadores físicos para selección

- **Algoritmo para índice primario usando árbol B+ con igualdad en clave candidata**
  - Recorrer la altura del árbol más una E/S para recoger el registro.
  - Cada una de estas operaciones requiere un acceso a bloque y una transferencia de bloque.
  - El costo es (h_i + 1) transferencias de bloque y (h_i + 1) accesos a bloque;
    - h_i denota la altura del índice.

Operadores físicos para selección

- **Algoritmo para índice primario usando árbol B+ igualdad para no clave candidata**
  - Hay un acceso a bloque para cada nivel del árbol y un acceso a bloque para el primer bloque.
  - Los registros van a estar en bloques consecutivos.
  - Sea b el número de bloques conteniendo registros con la clave de búsqueda especificada, todos los cuales son leídos.
  - Estos bloques son bloques hoja asumiendo que están almacenados secuencialmente y no requieren accesos adicionales a bloque.
  - El costo se compone de:
    - h_i transferencias de bloque y h_i accesos a bloque para recorrer el árbol B+. Donde h_i es la altura del árbol B+.
    - Un acceso a bloque para buscar los registros en el archivo de la tabla.
    - Un total de b transferencias de bloque para acceder a los registros con la clave de búsqueda.

Operadores físicos para selección

- **Algoritmo para índice secundario usando árbol B+, igualdad en clave candidata**
  - Este caso es similar al índice primario y por lo tanto el costo es el mismo.
  - (h_i + 1) transferencias de bloque y (h_i + 1) accesos a bloque.
  - donde h_i denota la altura del índice.
- **Algoritmo para índice secundario usando árbol B+, igualdad en no clave**
  - Aquí el costo de recorrido del índice es el mismo. Sin embargo, cada registro puede estar en un bloque diferente, requiriendo un acceso a bloque por registro.
  - Sea *n* el número de registros recogidos.
  - El costo es (h_i + *n)* transferencias de bloque y (h_i + *n)* accesos a bloque.
  - Esto puede ser muy costoso.

Selecciones involucrando comparaciones

- Podemos implementar selecciones de la forma $\sigma_{A \leq V}(r)$ o $\sigma_{A \geq V}(r)$ usando:
  - un escaneo de archivo lineal,
  - o usando índices
- **Algoritmo para índice primario:**
  - La tabla está ordenada en *A*.
  - para $\sigma_{A \geq V}(r)$ usar el índice para encontrar el primer registro $\geq v$ y escanear la tabla secuencialmente desde allí.
    - El costo se compone de: hi transferencias de bloque, hi accesos a bloque y b transferencias de bloque; b es el número de bloques conteniendo registros con A $\geq v$.
  - para $\sigma_{A \leq V}(r)$ escanear la tabla secuencialmente hasta la primera tupla > *v;* no usar el índice.

Selecciones involucrando comparaciones

- **Algoritmo para índice secundario**
  - para $\sigma_{A \geq V}(r)$ usar el índice para encontrar la primera entrada del índice $\geq v$ y escanear el índice secuencialmente desde allí, para encontrar los punteros a los registros.
  - Costo = (hi + n) transferencias de bloque y (hi + n) accesos a bloque,
    - donde *n* número de registros con A >= v.
  - para $\sigma_{A \leq V}(r)$ escanear hojas del índice encontrando punteros a los registros, hasta la primera entrada > *v.*
  - En ambos casos retornar los registros que son apuntados
    - Requiere una E/S para cada registro.
    - La búsqueda lineal de la tabla puede ser más barata.

Selecciones involucrando comparaciones

- Leer implementación de selecciones complejas:
  - Sección 12.3.3 del Silberschatz.

Tamaño de resultado de operador físico de selección

- **Problema**: Hay que estimar el tamaño de los resultados en cantidad de bloques a escribir a disco para los operadores físicos de selección.
- **Solución**:
  1. Usar una función de probabilidad llamada **factor de selectividad** para calcular la cantidad de registros del resultado del operador físico de selección.
  2. A partir de cantidad de registros, calcular la cantidad de bloques del resultado intermedio.

Tamaño de resultado de operador físico de selección

- Si el operador de selección o reunión usa predicado P, y el input del operador es r, denotamos al factor de selectividad mediante: fs(P, r).
- **Para selección:**

  Cantidad de registros resultado intermedio = |r| * fs(P, r)
- Una forma de calcular el factor de selectividad es asumir uniformidad e independencia:
  - **Uniformidad**: todos los valores de un atributo son igualmente probables
  - **Independencia**: condiciones sobre diferentes atributos son independientes

Tamaño de resultado de operador físico de selección

- **Ejemplo**: atributo sexo de persona: todos los valores de sexo son igualmente probables (la proporción de hombres es igual a la proporción de mujeres).
  - fs(sexo = ‘F’, persona) = 1/2
- **Ejemplo**: Atributos sexo y edad en persona, P = edad = 40 y sexo = ‘F’. La edad es independiente del sexo.
  - fs(edad = 40 and sexo = ‘F’, persona) = fs(edad = 40, persona) * fs(sexo = ‘F’, persona)
- Pero no es obligatorio usar uniformidad o independencia para calcular el factor de selectividad.

Cálculo de número de bloques

- **Problema**: ¿Cómo calcular el número de bloques si tengo en el resultado N registros de tamaño R cada uno y B es el tamaño del bloque?
- **Solución**:

  NumBloques = ⌈(N x R) / B⌉ .

Factor de selectividad para selección

- Definimos fs(P, r) para distintos tipos de propiedades P y tabla r.
- Desde ahora *r* tabla, *A*, *A'* atributos de *r*, *c* y *c'* constantes.
- **Regla 1:** Asumiendo uniformidad:
  - fs(A = c, r) = 1/V(A, r), donde V(A, r) número de distintos valores que aparecen en *r* para *A*.
- **Regla 2:** Asumiendo uniformidad, A con valor numérico:
  - fs(A $\geq$ c, r) = (max(A, r) - c) / (max(A, r) – min(A, r))
- **Regla 3:** Asumiendo uniformidad, A con valor numérico:
  - fs(A < c, r) = (c – min(A)) / (max(A) – min(A) + 1)

Factor de selectividad para selección

- **Regla 4:** asumiendo uniformidad, A con valor numérico:
  - $fs(c \leq A < c', r) = (c' - c)/(\max(A, r) - \min(A, r))$
- **Regla 5:** asumiendo independencia:
  - $fs(P1 \wedge P2 \wedge \ldots \wedge Pn, r) = fs(P_1, r)\, fs(P_2, r) \ldots fs(P_n, r)$
- **Regla 6:** usando propiedad de probabilidades:
  - $fs(\neg P, r) = 1 - fs(P, r)$
- **Regla 7:** asumiendo independencia
  - $fs(P \vee Q, r) = fs(\neg (\neg P \wedge \neg Q, r)) = 1 - fs(\neg P \wedge \neg Q, r) = 1 - (fs(\neg P, r) * fs(\neg Q, r))$
    $= 1 - ((1 - fs(P, r)) * (1 - fs(Q, r)))$

Producto cartesiano

profe			
legajo	nombres	apellidos	sueldo
p1	"Benjamin"	"Pierce"	3000
p2	"Patricia"	"Selinger"	6000
p3	"Edgar F"	"Codd"	5500
p4	"Barbara"	"Liskov"	5600

curso		
id	legajo	nombre
c1	p2	"Optimización de Consultas"
c2	p3	"Fundamentos de BD"
c3	p2	"Análisis de Datos"
c4	p1	"Fundamentos del Software"
c5	p4	"Programación OO"

–						
p.legajo	nombres	apellidos	sueldo	id	c.legajo	nombre
p1	"Benjamin"	"Pierce"	3000	c1	p2	"Optimización de Consultas"
p1	"Benjamin"	"Pierce"	3000	c2	p3	"Fundamentos de BD"
p1	"Benjamin"	"Pierce"	3000	c3	p2	"Análisis de Datos"
p1	"Benjamin"	"Pierce"	3000	c4	p1	"Fundamentos del Software"
p1	"Benjamin"	"Pierce"	3000	c5	p4	"Programación OO"
p2	"Patricia"	"Selinger"	6000	c1	p2	"Optimización de Consultas"
p2	"Patricia"	"Selinger"	6000	c2	p3	"Fundamentos de BD"
...	...	...	...	...	...	...

Observar el esquema del producto cartesiano en el ejemplo.

¿Qué pueden decir del mismo?

Producto cartesiano

- Sean los esquemas:

$$r :: (n_1 :: \tau_1, \ldots, n_N :: \tau_N) \qquad s :: (n'_1 :: \tau'_1, \ldots, n'_M :: \tau'_z M)$$

- Entonces el **esquema del producto cartesiano** es:

$$r \times s :: (n_1 :: \tau_1, \ldots, n_N :: \tau_N, n'_1 :: \tau'_1, \ldots, n'_M :: \tau'_M)$$

- Si hay **conflictos de nombres de atributos** se usa el nombre de la tabla para desambiguar.
- Una tabla puede contener **columnas sin nombre**. Para ese caso se usa: _: T
- Voy a poder aplicar el producto cartesiano para tablas donde no todas las columnas tienen nombre.

Producto cartesiano

- Para definir producto cartesiano podemos inspirarnos en la forma de construir la tabla:

–						
p.legajo	nombres	apellidos	sueldo	id	c.legajo	nombre
p1	"Benjamin"	"Pierce"	3000	c1	p2	"Optimización de Consultas"
p1	"Benjamin"	"Pierce"	3000	c2	p3	"Fundamentos de BD"
p1	"Benjamin"	"Pierce"	3000	c3	p2	"Análisis de Datos"
p1	"Benjamin"	"Pierce"	3000	c4	p1	"Fundamentos del Software"
p1	"Benjamin"	"Pierce"	3000	c5	p4	"Programación OO"
p2	"Patricia"	"Selinger"	6000	c1	p2	"Optimización de Consultas"
p2	"Patricia"	"Selinger"	6000	c2	p3	"Fundamentos de BD"
...	...	...	...	...	...	...

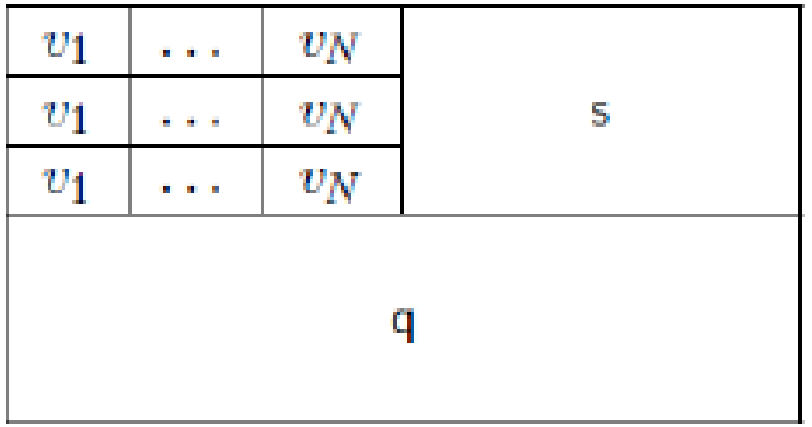
- Junta la primera tupla de profe con cada tupla de curso, junta la segunda tupla de profe con cada tupla de curso.
- **¿Cómo se puede generalizar esta idea para el producto cartesiano en general?**

Producto cartesiano

- Para definir producto cartesiano podemos inspirarnos en la forma de construir la tabla:

–						
p.legajo	nombres	apellidos	sueldo	id	c.legajo	nombre
p1	"Benjamin"	"Pierce"	3000	c1	p2	"Optimización de Consultas"
p1	"Benjamin"	"Pierce"	3000	c2	p3	"Fundamentos de BD"
p1	"Benjamin"	"Pierce"	3000	c3	p2	"Análisis de Datos"
p1	"Benjamin"	"Pierce"	3000	c4	p1	"Fundamentos del Software"
p1	"Benjamin"	"Pierce"	3000	c5	p4	"Programación OO"
p2	"Patricia"	"Selinger"	6000	c1	p2	"Optimización de Consultas"
p2	"Patricia"	"Selinger"	6000	c2	p3	"Fundamentos de BD"
...	...	...	...	...	...	...

- Generalizando:
- Donde t tupla de r y s tabla
- (estoy definiendo r x s)

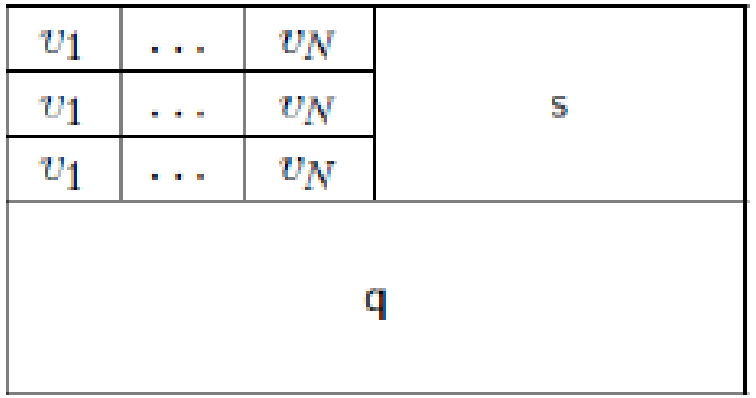


Producto cartesiano

```
[] x s = []

(t:r) x s = (juntar t s) ++ q
```

Recordar que:

```
++ : [a] -> [a] -> [a]

[] ++ l' = l'

(x : l) ++ l' = x : (l ++ l')
```



Ahora definiremos juntar.

Concatenación de tuplas

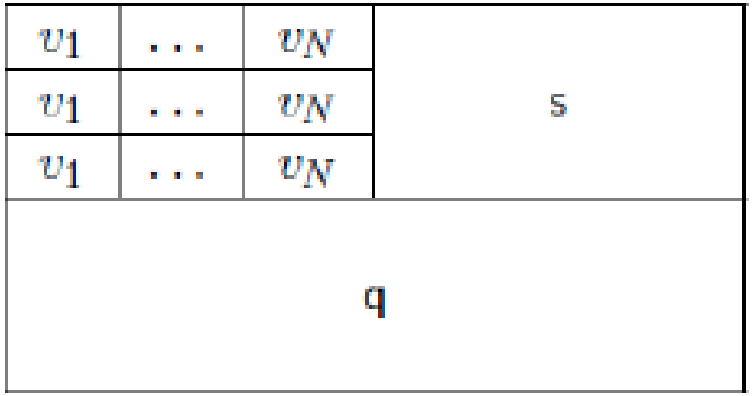
- Definimos la **concatenación de tuplas** como la tupla:

$$(t; t') = (a_1, \ldots, a_N, b_1, \ldots, b_M)$$

- **Usando la concatenación de tuplas, ¿cómo se puede definir juntar?**

Anexar

- juntar t [] = []
- juntar t (u : s) = (t ; u) : (juntar t s)



Relación entre SQL y Álgegra de Tablas

select $A1,\ldots,An$

from $r1,\ldots,rn$

where P

- Es equivalente a:

$$\Pi_{A1,\ldots,An}(\sigma_p(r_1 \times \ldots \times r_n))$$

Ordenamiento

- Especificamos **ordenamiento** usando la idea de insertion sort.
- La función *insert* inserta la tupla *t* en la tabla *r* en la posición que le corresponde: a izquierda las tuplas menores y a derecha las mayores.

$$\mathbf{insert}_{a_1,\ldots,a_N}\ \mathrm{t}\ \mathrm{r} = \sigma_{(a_1,\ldots,a_N)<\mathrm{t}[a_1,\ldots,a_N]}(\mathrm{r}) \mathbin{++} [\mathrm{t}] \mathbin{++} \sigma_{(a_1,\ldots,a_N)\geq\mathrm{t}[a_1,\ldots,a_N]}(\mathrm{r})$$

- El **ordenamiento ascendente** se define:
- $O_{a1,\ldots,aN}([]) = []$
- $O_{a1,\ldots,aN}(x{:}r) = \mathrm{insert}_{a1,\ldots,aN}(x, O_{a1,\ldots,aN}(r))$

Ordenamiento

- El **ordenamiento descendente** se define como la reversa del ordenamiento ascendente.
- $O^{>}_{a1,\ldots,aN}(r) = \text{reverse}\ (O_{a1,\ldots,aN}(r))$
- Reverse [] = []
- Reverse (x : xs) = reverse xs ++ [x]
- Cuando queramos ordenar toda la tabla, omitiremos la lista de las columnas.

Operadores físicos de ordenamiento

- Ya saben ordenar tablas que caben en memoria principal.
- Estudiamos el caso de ordenamiento donde las tablas son mayores que la memoria principal.
  - Ordenar tablas que no caben en memoria se llama **ordenamiento externo**.
  - El algoritmo de ordenamiento externo más usado se llama **ordenamiento-combinación externo (external merge sort)**.

Ordenamiento externo

- Sea *M* la cantidad de bloques en búfer de memoria principal disponibles para ordenación.

1. **Crear corridas ordenadas**

   i = 0

   repeat

      Leer *M* bloques de la tabla en memoria

      ordenar esos bloques en memoria

      Escribir los datos ordenados al archivo de ejecución R_i,

      i = i+1

   until el fin de la tabla

   - Sea N el valor final de i

2. **Combinar las corridas**

Ordenamiento externo

- Luego las corridas son combinadas. Suponemos que $N < M$.

  Leer un bloque de cada uno de los *N* archivos Ri en un bloque del búfer de memoria de corrida Ri.

  repeat

  Seleccionar la primera tupla en el orden de ordenamiento entre todos los bloques del búfer.

  Escribir la tupla al búfer de salida y borrarla del bloque del búfer donde estaba.

  if búfer de salida está lleno **then** escribirlo a disco, borrar contenido de búfer de salida.

  if búfer de corrida Ri está vacío y no fin de archivo Ri **then** leer el próximo bloque de la corrida Ri en el bloque de búfer de memoria correspondiente a esa corrida.

  until todos los bloques de búfer de las corridas estén vacíos.

Ordenamiento externo

- Si $N \geq M$, varios ciclos de combinación son requeridos.
  - Se dividen las corridas en grupos contiguos de M-1 corridas.
  - Cada grupo de M-1 corridas es combinado (usando el algoritmo anterior), dando lugar a una nueva corrida.
  - Hacer los dos pasos anteriores constituye una pasada.
    - La misma reduce el número de corridas por un factor de M-1 y crea corridas más largas por el mismo factor.
  - Si el número de corridas resultantes de esa pasada es mayor a M,
    - se realiza otra pasada de la misma manera, pero con las corridas creadas en la primera pasada.
    - Observar que cada pasada reduce el número de secuencias en M-1.
  - Se repiten las pasadas hasta que número de corridas generadas sea menor que *M*.
  - Luego el último ciclo de combinación genera el resultado ordenado.

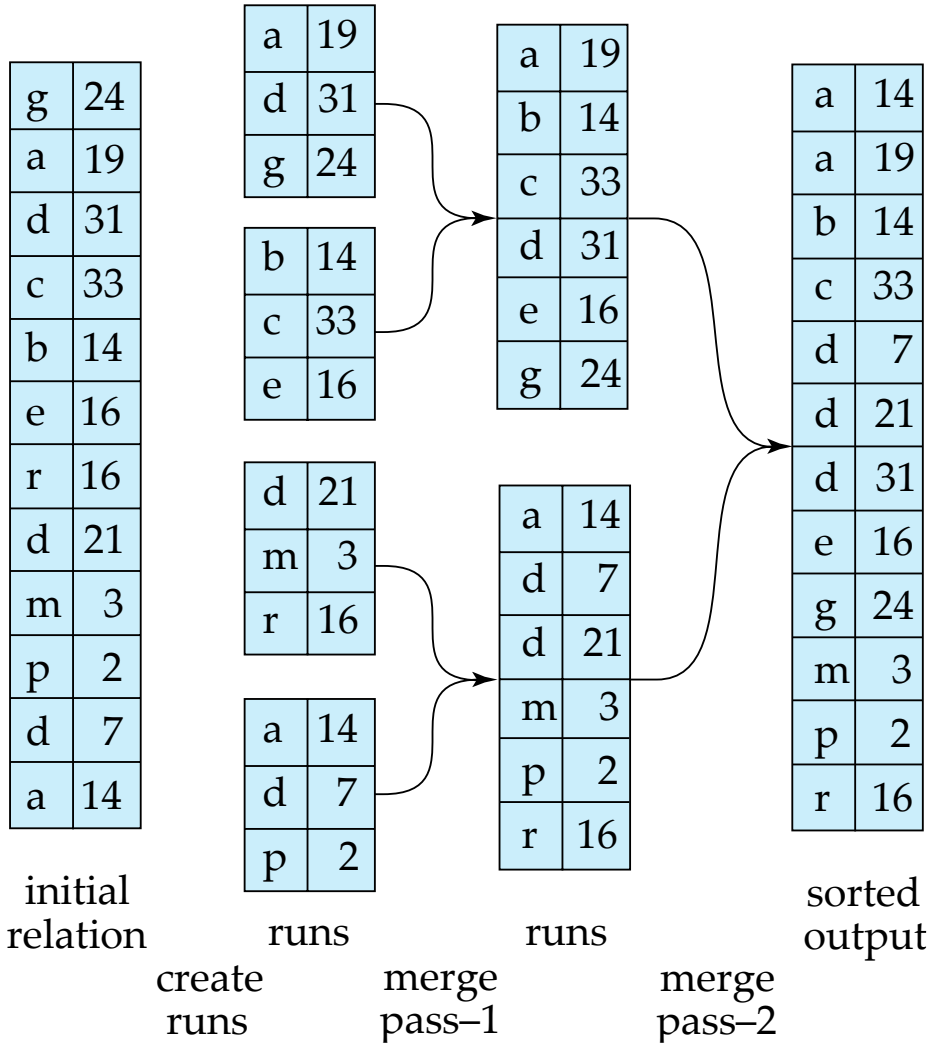
Ordenamiento externo

Ejemplo: Se asume que:

- En memoria entran 3 bloques (M = 3).
- Se asume que bloques son de 1 tupla cada uno.

El algoritmo opera así:

- Primero se crean 4 corridas ordenadas.
- Luego se consideran grupos contiguos de M-1 = 2 corridas.
  - Se combinan grupos contiguos de 2 corridas. Se hacen 2 combinaciones.
  - Se obtienen así 2 corridas.
- Las dos corridas finales son combinadas dando el resultado ordenado final.



Ordenamiento externo

- **Costo del ordenamiento:**
  - Leer justificación en el libro de Silberschatz.
  - Número de transferencias de bloques para ordenamiento externo:

    $b_r (2 \lceil \log_{M-1} (b_r / M) \rceil + 1)$
  - Donde b_r es cantidad de bloques de la tabla
  - Cantidad de accesos a bloque:

    $2 \lceil b_r / M \rceil + \lceil b_r / b_b \rceil (2 \lceil \log_{\lfloor M/bb \rfloor - 1} (b_r / M) \rceil - 1)$,
  - donde b_b bloques son alojados en cada corrida

Reunión selectiva

- Unimos las siguientes tablas mediante el campo *legajo*:

profe			
legajo	nombres	apellidos	sueldo
p1	"Benjamin"	"Pierce"	3000
p2	"Patricia"	"Selinger"	6000
p3	"Edgar F"	"Codd"	5500
p4	"Barbara"	"Liskov"	5600

curso		
id	legajo	nombre
c1	p2	"Optimización de Consultas"
c2	p3	"Fundamentos de BD"
c3	p2	"Análisis de Datos"
c4	p1	"Fundamentos del Software"
c5	p4	"Programación OO"

_						
legajo	nombres	apellidos	sueldo	id	legajo	nombre
p1	"Benjamin"	"Pierce"	3000	c4	p1	"Fundamentos del Software"
p2	"Patricia"	"Selinger"	6000	c1	p2	"Optimización de Consultas"
p2	"Patricia"	"Selinger"	6000	c3	p2	"Análisis de Datos"
p3	"Edgar F"	"Codd"	5500	c2	p3	"Fundamentos de BD"
p4	"Barbara"	"Liskov"	5600	c5	p4	"Programación OO"

- **La operación de combinar tablas por atributos en común es muy utilizada.** $\text{profe} \underset{\text{legajo}}{\bowtie} \underset{\text{legajo}}{\text{curso}}$

Reunión selectiva

- Ahora pasamos a ver el esquema de la reunión selectiva.
- Sean dos tablas de esquema:

$$r(n_1 :: \tau_1, \ldots, n_N :: \tau_N) \text{ y } s(m_1 :: \tau'_1, \ldots, m_M :: \tau'_M)$$

- Entonces:

$$r \underset{a_1,\ldots,a_i}{\bowtie}_{b_1,\ldots,b_i} s :: (n_1 :: \tau_1, \ldots, n_N :: \tau_N, c_1, \ldots, c_{M-i})$$

$$\text{para } j \in [1, M-i],\ c_j \in \{m_i, \ldots, m_M\} \setminus \{b_1, \ldots, b_i\}.$$

Reunión selectiva

- La reunión selectiva se puede definir usando los operadores vistos:

$$r \ {}_{a_1,\ldots,a_i}\bowtie_{b_1,\ldots,b_i} \ s = \Pi_{n_1,\ldots,n_N,c_1,\ldots,c_{M-i}}(\sigma_{a_1=b_1 \wedge \ldots \wedge a_i=b_i}(r \times s))$$

- Un tipo de reunión selectiva que se usa mucho es **reunión natural**: se aplica reunión selectiva a todos los atributos con el mismo nombre en las dos tablas.
- Sean $r(n_1 :: \tau_1, \ldots, n_N :: \tau_N)$ y $s(m_1 :: \tau'_1, \ldots, m_M :: \tau'_M)$, definimos la reunión natural de *r* y *s* como:

$$r \bowtie s = r \ {}_{a_1,\ldots,a_i}\bowtie_{a_1,\ldots,a_i} \ s$$

donde $\{a_1, \ldots, a_i\} = \{n_1, \ldots, n_N\} \cap \{m_1, \ldots, m_M\}$.

Reunión natural

- Relaciones *r*, *s*:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

- Los atributos en común son: B y D
- El esquema de la reunión natural de r y s: (A, B, C, D, E)

- $r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Operadores físicos para reunión selectiva

- **Algoritmo de reunión selectiva de loop anidado**
- **for each** tuple t_r **in** r **do begin**
  for each tuple t_s **in** s **do begin**
  test pair (t_r, t_s) to see if they satisfy the join condition
  if they do, add $t_r \bullet t_s$ to the result.
  end
  end

Operadores físicos para reunión selectiva

- Nomenclatura: *r* se llama **tabla externa** y *s* se llama **tabla interna** de la reunión selectiva.
- El algoritmo anterior no requiere de índices.
- El algoritmo anterior es costoso porque examina todo par de tuplas en las dos tablas.
- En el peor caso, si hay suficiente memoria solo para sostener un bloque para cada tabla, el costo estimado es:
  - $n_r * b_s + b_r$ transferencias de bloques
  - $n_r + b_r$ accesos a bloques.
  - donde n_r es cantidad de registros en *r* , b_r es la cantidad de bloques en *r,* y b_s es la cantidad de bloques en *s.*

Operadores físicos para reunión selectiva

- **Algoritmo de reunión selectiva de loop anidado de bloques**

for each block B_r **of** r **do begin**
for each block B_s **of** s **do begin**
for each tuple t_r **in** B_r **do begin**
for each tuple t_s **in** B_s **do begin**
if (t_r, t_s) satisface condición de reunión selectiva
then add $t_r \bullet t_s$ to the result.
end
end
end
end

Operadores físicos para reunión selectiva

- **Estimación de peor caso:**
  - $b_r * b_s + b_r$ transferencias de bloque + 2 * b_r accesos a bloque
  - Cada bloque en la tabla interna *s* es leído una sola vez por cada bloque en la tabla externa.
- **Mejora del algoritmo anterior:**
  - M tamaño de memoria en bloques.
  - Usar M-2 bloques de disco para tabla externa, usar los restantes 2 bloques para tabla interna y salida.
  - Costo = $\lceil b_r / (M-2) \rceil * b_s + b_r$ transferencias de bloques + $2\lceil b_r / (M-2) \rceil$ accesos a bloque

Operadores físicos para reunión selectiva

- **Algoritmo de reunión selectiva de loop anidado indexado**
  - Asumir que hay un índice en el atributo de la relación interna de la reunión.
  - Para cada tupla t_r en la tabla externa *r* usar el índice para buscar tuplas en *s* que satisfacen la condición de reunión con tupla t_r.
  - **Peor caso**: el búfer tiene espacio para solo un bloque de *r* y un bloque del índice y para cada tupla en *r* hacemos búsqueda en índice de *s*.
    - Para leer *r* en el peor caso hay b_r accesos a bloque y b_r transferencias de bloque.
    - A esto hay que sumarle $n_r * c$. n_r es cantidad de registros en *r.*
    - Aquí *c* es el costo de recorrer el índice y recolectar todas las tuplas de *s* que cazan para una tupla de *r*.
    - El valor *c* puede estimarse como el costo de una selección en *s* usando la condición de la reunión.

Operadores físicos para reunión selectiva

Algoritmo de reunión por mezcla:

- Supongamos que hay un atributo de la reunión.
- Ordenar ambas tablas en el atributo de la reunión.
- Aplicar la reunión selectiva a las tablas ordenadas.
  - Si las tablas consideradas son r y s y se quiere hacer reunión selectiva de r y s:
  - Se trabaja con variables para tupla actual de r y tupla actual de s
  - El algoritmo se basa en aplicar las siguientes dos ideas:
    - Si tupla actual t de r caza con tupla actual t' de s: se agrega t ⋈ t' al resultado.
      - Se repite esto hasta que haya tupla de s que no caza con t o se acabó de recorrer s.
    - Si tupla actual de r no caza con tupla actual de s, se pasa a siguiente tupla de r
      - Se repite esto hasta que se acabe de recorrer r o se encuentra tupla de r que caza con tupla actual de s.

Operadores físicos para reunión selectiva

- **Análisis del algoritmo de reunión por mezcla:**
  - Cada bloque necesita ser leído una sola vez asumiendo que todas las tuplas para un valor dado de los atributos del join entran en memoria.
  - Entonces el costo de reunión por mezcla es:
    - $b_r + b_s$ transferencia de bloques $+\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil$ accesos a bloque
    - Donde b_b bloques de búfer son asignados a cada tabla.
    - A esto se suma el costo de ordenar las tablas si no estaban ordenadas.

Operadores físicos para reunión selectiva

- **Algoritmo de reunión por mezcla híbrido:**
- **Suposición**: la primera tabla está ordenada y la segunda tabla está desordenada, pero tiene un índice secundario árbol B+ en los atributos de la reunión.
- **Algoritmo**:
  1. El algoritmo combina la tabla ordenada con las entradas hoja del índice secundario B+.
     - El archivo resultante contiene tuplas de la relación ordenada y direcciones para las tuplas de la relación no ordenada.
  2. Se ordena el resultado por las direcciones de las tuplas de la relación no ordenada,
  3. Permitiendo el retorno eficiente de las tuplas correspondientes en el orden de almacenamiento físico para terminar la reunión.
- Costo del segundo paso: Costo de ordenar el archivo resultante.
- Costo del tercer paso: costo de acceso a tuplas en el orden de almacenamiento físico.

Tamaño de resultado de operador físico de reunión

- **Problema**: Hay que estimar el tamaño de los resultados en cantidad de bloques a escribir a disco para los operadores físicos de reunión (selectiva y natural).
- **Solución**:
  1. Usar una función de probabilidad llamada **factor de selectividad** para calcular la cantidad de registros del resultado del operador físico de reunión.
  2. A partir de cantidad de registros, calcular la cantidad de bloques del resultado intermedio.
- La cantidad de registros para el resultado de la reunión vien dada por

$$|r| * |s| * fs(P, r, s)$$

Factor de selectividad para reunión selectiva

- Si r tabla y A atributo de r, entonces V(A, r) número de distintos valores que aparecen en *r* para *A*.
- **Regla**: asumiendo uniformidad:
  - fs(r.A = s.B, r, s) = 1 /max(V(A, r), V(B, s))
  - la uniformidad significa: para cada valor de atributo A/B en una tabla hay la misma cantidad de tuplas por el atributo B/A que cazan en la otra tabla.
- **Observaciones**:
  - Si A clave candidata de r: fs(r.A = s.B, r, s) = 1 /max(|r|, V(B, s))
  - Si B clave foránea en s referenciando r: fs(r.A = s.B, r, s) = 1 /|r|
  - Si toda tupla en r produce tuplas en la reunión selectiva:
    - fs(r.A = s.B, r, s) = 1 /V(B,s)
  - Si se puede asumir independencia:
    - fs(r.A = s.B Λ r.A' = s.B', r, s) = fs(r.A = s.B , r,s) * fs(r.A' = s.B', r,s)

Remover duplicados

- La operación de remoción de duplicados se define recursivamente así:

1. V [ ] = [ ]
2. V (t:r) = if t∈r then V (r) else t: V (r)

- El predicado en la segunda ecuación se define del siguiente modo:
- $t \in [\,] = \text{false}$
- $t \in (u : r) = t == u \,||\, t \in r$

Operador físico de remoción de duplicados

- **Implementación de remoción de duplicados**
  - Se ordena la tabla; al hacerlo, todos los duplicados van a ser adyacentes entre sí. Y todos los duplicados menos uno, pueden ser eliminados.
  - **Optimización**: en ordenación externa duplicados pueden ser eliminados durante generación de corridas, así como en pasos de combinación intermedios.
  - El peor caso de costo para eliminación de duplicados es el mismo que el peor caso de costo para ordenar una tabla.

Concatenación de tablas

- ¿Qué operación de tablas corresponde a la unión de conjuntos?
- Una posibilidad es la **concatenación de tablas**.
- Es como la concatenación de listas solo que ahora el resultado tiene un esquema.
- Los esquemas de los operandos deben ser **compatibles (**i.e. igual cantidad de columnas y mismos tipos en las mismas posiciones**)**.

$$\text{Sean } r(n_1 :: \tau_1, \ldots, n_N :: \tau_N) \text{ y } s(m_1 :: \tau_1, \ldots, m_M :: \tau_N)$$

$$r{+}{+}s :: (n_1 :: \tau_1, \ldots, n_N :: \tau_N)$$

- [] ++ s = s
- (t:r)++ s = t: (r++s)

Concatenación

- Concatenación requiere leer ambas tablas: primero la de la izquierda y luego la de la derecha.
  - A medida que se lee una tabla se genera el resultado.
- En el peor caso para computar r++s vamos a necesitar:
  - $b_r + b_s$ transferencias de bloques y accesos a bloques
- Si el resultado es un resultado intermedio: se escriben $b_r + b_s$ bloques en disco.

Resta

Los esquemas de las tablas deben ser compatibles

$$r(n_1 :: \tau_1, \ldots, n_N :: \tau_N) \text{ y } s(m_1 :: \tau_1, \ldots, m_M :: \tau_N)$$

El esquema de la resta es el del primer operando.

$$r \setminus s :: (n_1 :: \tau_1, \ldots, n_N :: \tau_N)$$

$$r \setminus s = \sigma_{(\backslash \mathsf{t} \to \mathsf{t} \notin \mathsf{s})}(r)$$

Resta

- **Algoritmo:**
  - Para calcular la resta de dos tablas, podemos primero **ordenarlas** a ambas por la clave primaria.
  - Luego podemos escanear una vez cada tabla ordenada para producir el resultado.
- **Costo:**
  - Suponiendo las tablas r, s ordenadas de ese modo.
  - Si un solo bloque en búfer se usa por tabla:
    - $b_r + b_s$ transferencias de bloques
    - $b_r + b_s$ accesos a bloque

Intersección

- Los esquemas de las tablas deben ser compatibles

$$r(n_1 :: \tau_1, \ldots, n_N :: \tau_N) \text{ y } s(m_1 :: \tau_1, \ldots, m_M :: \tau_N)$$

- El esquema de la intersección es el del primer operando.

$$r \cap s :: (n_1 :: \tau_1, \ldots, n_N :: \tau_N)$$

$$r \cap s = \sigma_{(\mathsf{t} \rightarrow \mathsf{t} \in \mathsf{s})}(r)$$

intersección

- **Algoritmo:**
  - Para calcular la intersección de dos tablas, podemos primero **ordenarlas** a ambas por la clave primaria.
  - Luego podemos escanear una vez cada tabla ordenada para producir el resultado.
- **Costo:** Suponiendo las tablas r, s ordenadas de ese modo, $r \cap s$ requiere:
  - Si un solo bloque en búfer se usa por tabla:
    - $b_r + b_s$ transferencias de bloques
    - $b_r + b_s$ accesos a bloque
  - El número de accesos a bloque puede ser reducido alojando bloques extra en búfer.