Capítulo 3

Operadores sobre tablas y su implementación – parte 1

Visión general

- Este capítulo es de **procesamiento de consultas**.
- Para poder procesar consultas necesitamos poder **acceder a las tablas** de la base de datos que están almacenadas en disco.
  - Para acceder a las tablas de manera eficiente muchas veces se usan índices.
- Para **procesar una consulta**, una sentencia de SQL se puede traducir a una **expresión de un álgebra.**
  - Un álgebra tiene operaciones con uno o con dos argumentos sobre tablas. Los resultados de esas operaciones son tablas.

Visión general

- **¿Por qué traducir de SQL a un álgebra?**
  - Un álgebra es un lenguaje formal y procedimental que describe cómo obtener un resultado a partir de las operaciones básicas sobre relaciones (tablas). Esto permite al sistema entender claramente qué operaciones realizar y en qué orden.
  - Una vez traducida a un álgebra, la consulta puede ser optimizada usando reglas algebraicas que simplifican o reordenan operaciones para mejorar el rendimiento.
  - Esto es más difícil de hacer directamente sobre SQL debido a su sintaxis declarativa y diversa.
  - El álgebra define operaciones básicas muy precisas (selección, proyección, unión, producto, etc.) que son fáciles de implementar y combinar para obtener el resultado deseado.

Visión general

- **Algebra relacional**
  - El **álgebra relacional** es muy usada en los libros de texto de bases de datos.
  - Esta álgebra opera con relaciones que son conjuntos.
  - Esto quiere decir que a diferencia de las tablas de SQL en una relación no hay un orden entre los elementos, ni puede haber tuplas repetidas.
  - Como consecuencia de esto, el algebra relacional no cuenta con algunas operaciones importantes.
    - **Por ejemplo**: ordenar una tabla según un conjunto de atributos, remover duplicados de una tabla, unir dos tablas generando tuplas duplicadas.
  - Para superar esta deficiencia del álgebra relacional vamos a estudiar un **álgebra de tablas**, donde una tabla es **una lista de tuplas**.
  - Esto permite **usar programación funcional** para definir operaciones sobre tablas.
  - Además se pueden **probar propiedades de las operaciones** del álgebra de tablas **por inducción sobre listas**.

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.

Relación entre SQL y Álgegra de Tablas

select *A*1,…,*A*n

from *r*1,…,*r*n

where *P*

- **Es equivalente a:**

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

- **Aclaración**:
  - Hacemos las combinaciones de tuplas de los r_i (operador producto cartesiano).
  - De las combinaciones de tuplas se seleccionan las tuplas que cumplen *P* (operador selección).
  - De las tuplas seleccionadas extraemos los valores de los atributos A_i (operador proyección.)

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.
- **Para discos rígidos:**
  - El tiempo de un acceso a bloque suele ser de alrededor de 4 mseg.
  - El tiempo de transferencia de bloques en suele ser de 0,1 mseg, asumiendo un tamaño de bloque de 4 KiB y una tasa de transferencia de 40 MB por segundo.
- **Para disco de estado sólido (SSD):**
  - El acceso a bloque puede llevar 35 microsegundos, gracias al acceso electrónico de la memoria flash.
  - Para SSD de 2 Gbps y tamaño de bloque de 4096 B una transferencia de bloque llevaría 16 microsegundos.
- **Conclusión**: con SSD no hay tanta diferencia entre los accesos a bloque y las transferencias bloque. En cambio en un disco rígido la diferencia es enorme.

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.
- Las estimaciones de coste que se proporcionan ignoran el coste de escribir el resultado final de una operación en disco.
  - Cuando no decimos nada, asumimos que esta escritura tiene lugar al final de la operacion.

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

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

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

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

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

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), …, fn(t)) : $\Pi_{f1,\ldots,fn}$ (r) = ((\t' -> (f1(t'),…,fn(t')) t) : $\Pi_{f1,\ldots,fn}$ (r)

- O sea que tiene la forma de un map.
- $\Pi_{f1,\ldots,fn}$ = map (\t' -> (f1(t'),…,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:
    - Asumimos que la tabla está organizada secuencialmente y todos sus bloques están contiguos en disco.
    - Asumimos que resultado cabe en memoria y se lo guarda al final, para no afectar accesos a bloques de r.
    - Recordar que no se cuenta guardar resultado en disco.
    - 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:
- $\sigma_P \, [] = []$
- $\sigma_P \, (t:r) = \text{ if } p(t) \text{ then } \; t : (\sigma_P \, r) \; \text{ else } \; \sigma_P \, r$

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**
    - Asumimos organización secuencial de r con bloques contiguos.
    - Asumimos resultados se guarda en memoria hasta el final donde se copia a disco. No se cuenta esta etapa.
    - 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.
  - **Estimación de costo:**
    - 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 $\sigma_{A \le V}(r)$ o $\sigma_{A \ge V}(r)$ usando:
  - un escaneo de archivo lineal,
  - o usando índices
- **Algoritmo para índice primario:**
  - La tabla está ordenada en *A*.
  - para $\sigma_{A \ge V}(r)$ usar el índice para encontrar el primer registro $\ge v$ 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 $A \ge v$.
  - para $\sigma_{A \le V}(r)$ 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*. O sino usar búsqueda lineal.
  - 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.

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)

<table>
<tr><td>v_1</td><td>$\dots$</td><td>v_N</td><td rowspan="3">s</td></tr>
<tr><td>v_1</td><td>$\dots$</td><td>v_N</td></tr>
<tr><td>v_1</td><td>$\dots$</td><td>v_N</td></tr>
<tr><td colspan="4">q</td></tr>
</table>

Producto cartesiano

[] x s = []

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

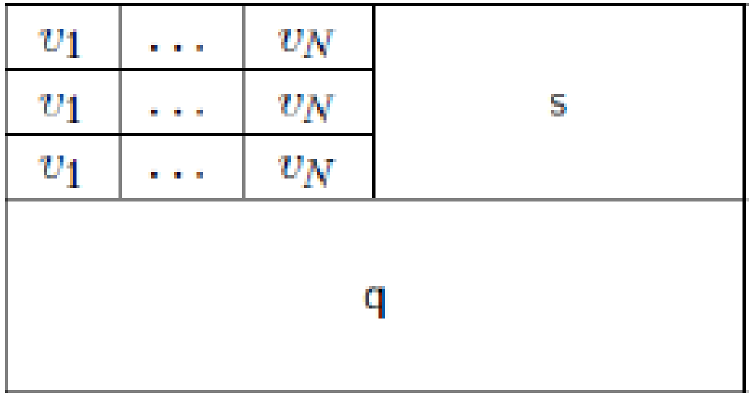
Recordar que:

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

[] ++ l' = l'

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

Ahora definiremos juntar.



Concatenación de tuplas

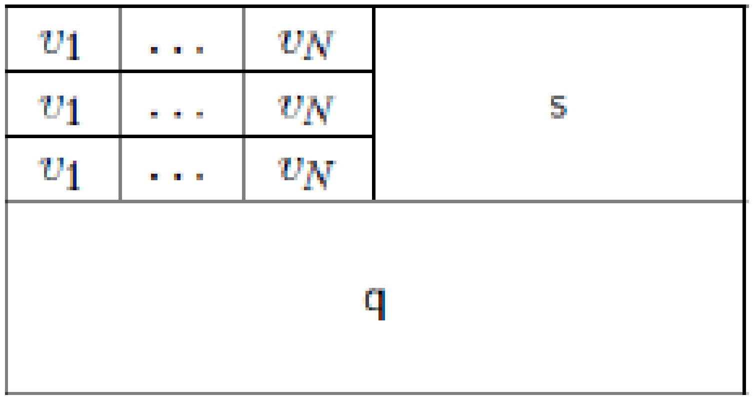
- Definimos la **concatenación de tuplas** como la tupla:

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

- **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 *A*1,…,*A*n

from *r*1,…,*r*n

where *P*

- Es equivalente a:

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