Teo: Algebra de Tablas

☑ Archive ☐

Introducción | Uso de Listas
    Notación y conceptos importantes (sobre listas)
Álgebra de Tablas
Operadores
Otros Operadores
NOTAS DE EMA

Introducción | Uso de Listas

Para entender el álgebra de tablas se sugiere trabajar con listas ya que las mismas permiten modelar tablas como en SQL y brindan mayor flexibilidad a la hora de realizar operaciones (agregar, eliminar o modificar elementos) en una tabla. En otras palabras, es más fácil manipular bases de datos si entendemos a la BD como una lista de tuplas a que como es planteado en el modelo relacional.

Entonces, básicamente en las filminas se menciona que las tablas son equivalente a listas de tuplas, haciendo referencia a que si hacemos una consulta sobre una tabla, esa consulta también se puede expresar en **operaciones sobre listas**, permitiéndonos tener un paralelismo entre tabla y lista, facilitándonos entender algunos conceptos.

Notación y conceptos importantes (sobre listas)

- Lista vacía: []
- Agregar elemento: operador “:” (Ej: a : b : c : [] = [a, b, c])
- Conjunto de listas de tipo T: [T] (Ej: [int], [string])
- Existe la recursión sobre listas (visto en Haskell, ej abajo)

- Ejemplo: Evaluar suma[1,2,3]

Suma 1 : 2 : 3 : []

= {suma.3} 1 + suma 2 : 3 : []

= {suma.3} 1 + 2 + suma 3:[]

= {suma.3} 1 + 2 + 3 + suma []

= {suma.2} 1 + 2 + 3 + 0

= {+} 6

1. Suma :: [Int] -> Int
2. Suma [] = 0
3. Suma x: xs = x + suma xs

Álgebra de Tablas

Un álgebra de tablas es un marco formal que proporciona un conjunto de operaciones y reglas para trabajar con conjuntos de tablas (representadas por listas de tuplas). Éstas operaciones son recursivas y sirven para manipular las listas.

El Álgebra de tablas nos ayuda para:

- **Darle una semántica formal a SQL**
  Las consultas en SQL son expresiones en el álgebra de tablas.
- **Dadas dos consultas en SQL, chequear si son equivalentes**
  Se usan propiedades (e inducción) para probar que dos expresiones del alg. de tablas son equivalentes
- **Dada una consulta en SQL, obtener una más eficiente**
  Se traduce la consulta a alg. tablas y se busca otra más eficiente usando propiedades, heurísticas y programación dinámica.
- **Dada una consulta en SQL, evaluarla**
  Se usan **operadores físicos** que asumen ciertas cosas sobre sus argumentos.

Operadores

Son lo que usaremos para poder hacer consultas a nuestras tablas. Utilizaremos un conjunto de las siguientes operaciones para construir las expresiones de consultas.

```
SELECT  nombre, DNI, nombreBib
from bibliotecario, trabajaEN
where bibliotecario.DNI = trabajaEN.DNI and antigüedad > 5
```

1. Proyección: Se usa para seleccionar un subconjunto de columnas de una tabla. Como resultado se tiene una nueva tabla que contiene sólo las columnas especificadas.
   Ejemplo: en la imagen anterior, en SELECT se proyectan sólo las tuplas de las columnas nombre, DNI y nombreBib. A esta operación la denotamos con el símbolo ∏. Podemos definir esta operación recursivamente sobre listas:

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

   Si lo vemos por partes:

   - $\Pi_{A,C}[] = []$ caso para lista vacía
   - $\Pi_{A,C}$(t:r) = (t.A, t.C) : ($\Pi_{A,C}$ r) caso recursivo, devuelve una lista donde se toma el componente A y C de la tupla t y se agrega a la lista resultante llamando recursivamente.
   - Luego se define la versión usando map, que para cada tupla t', se devuelve la tupla con sus componentes A y C.
     $\Pi_{A,C}$ = map (\t' -> (t'.A, t'.C))

   En definitiva, todo se traduce a

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

2. Proyección Generalizada: es como proyección pero cuando se quiere proyectar información o atributos que no están en los componentes. En el ejemplo de aquí abajo, sueldos_anuales no está en la tabla profe.

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)

Con map (donde f1 = legajo y f2 = sueldo*13):

$$\Pi_{f1,\ldots,fn} = \text{map } (\backslash t' \text{ -> } (f1(t'),\ldots,fn(t')))$$

Definición

$$\Pi_{f_1,\ldots,f_N}([]) = []$$
$$\Pi_{f_1,\ldots,f_N}(x : xs) = (f_1\ x,\ \ldots,\ f_N\ x) : \Pi_{f_1,\ldots,f_N}(xs)$$

3. Selección: obtiene las tuplas que cumplen las condiciones/predicados que nosotros deseamos.

- 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"

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

El operador de selección se define recursivamente como sigue:

$\sigma_p\ [] = []$

$\sigma_p\ (t:r) =$ if p(t) then $t : (\sigma_p\ r)$ else $\sigma_p\ r$

4. Producto cartesiano: combina cada fila de una tabla con cada fila de otra tabla. Genera todas las posibles combinaciones de filas entre dos 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"

–						
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.

La definición:

[] × s = []

(t:r) × s = anexar s t (r × s)

donde anexar s t q = (map (\t' -> t ; t') s) ++ q. Ejemplo:

si tenemos una tupla `t = (1, "Alice")` y una lista de tuplas `s = [(10, "Bob"), (20, "Charlie")]`, la función `anexar` producirá `[(t, (10, "Bob")), (t, (20, "Charlie"))]`.

Ejemplo de todo

Dadas 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"

Quiero encontrar los nombres de los profes que dictan 'Análisis de datos':

1. Hacemos el *producto cartesiano* de ambas tablas ya que la consulta se refiere a atributos de ambas tablas (legajo profe y legajo curso).

-						
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"
...	...	...	...	...	...	...

2. Hacemos la selección de las filas que nos interesan, en este caso, las filas tal que cumplan el predicado legajo del profesor == legajo del curso.

–						
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”

3. Hacemos selección de nuevo para sacar las filas que tengan nombre de curso == “Análisis de Datos”, es decir

–						
legajo	nombres	apellidos	sueldo	id	legajo	nombre
p2	“Patricia”	“Selinger”	6000	c3	p2	“Análisis de Datos”

4. Aplicando proyección obtenemos

–
nombres
“Patricia”

O sea, hicimos la consulta: $\Pi_{\text{nombres}}\ (\ \sigma_{P}\ (\ \sigma_{Q}\ (\text{profe x curso})\ )\ )$

En definitiva, la consulta tiene la siguiente pinta:

$$\Pi_{\text{nombres}}(\sigma_{\text{profe.legajo}=\text{curso.legajo} \wedge \text{curso.nombre}=\text{“Análisis de Datos”}}(\text{profe} \times \text{curso}))$$

que es equivalente en SQL a:

```
SELECT nombres
FROM profe, curso
WHERE profe.legajo = curso
      AND curso.nombres = “Análisis de Datos"
```

En general:

select *A*1,…,*A*n

from *r*1,…,*r*n

where *P*

Es equivalente a:

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

Otros Operadores

Los primeros 3 operadores son los más básicos.

4. Reunión Selectiva: es un operador que combina tablas por atributos en común. Análogo a Selección cuando los atributos en el predicado coinciden. Con Proyección, muestra las tuplas enteras que compartan los mismos valores en los atributos denotados.

$\Pi_{\text{nombre, apellido, DNI}}(\sigma_{\text{persona.DNI = bibliotecario.DNI}}(\text{persona x bibliotecario}))$

Así introducimos un operador llamado reunión selectiva que hace esto:

$\Pi_{\text{nombre, apellido, DNI}}(\text{persona}_{\text{DNI}} \bowtie_{\text{DNI}} \text{bibliotecario})$

podría decirse que reunión selectiva es como una “abreviación” de selección. De todas maneras, ambas cosas son distintas, en reunión selectiva, la columna DNI no se repite.

¿Cómo se pude definir reunión selectiva 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 $\{a_1, \ldots, a_i\} = \{n_1, \ldots, n_N\} \cap \{m_1, \ldots, m_M\}$.

La consulta anterior:

$$\Pi_{\text{nombres}}(\sigma_{\text{curso.nombre}=\text{“Análisis de Datos”}}(\text{profe} \underset{\text{legajo}}{\bowtie}_{\text{legajo}} \text{curso}))$$

es una reunión natural:

$$\Pi_{\text{nombres}}(\sigma_{\text{curso.nombre}=\text{“Análisis de Datos”}}(\text{profe} \bowtie \text{curso}))$$

Disyunción: Construcción de consultas complejas

Se usa el operador **let** para acortar expresiones a través de la asignación de etiquetas.

Esta estaba expresada como:

$$\Pi_{\text{nombres}}(\sigma_{\text{curso.nombre}=\text{“Análisis de Datos”}}(\text{profe} \underset{\text{legajo}}{\bowtie}_{\text{legajo}} \text{curso}))$$

Se puede escribir así:

$$\begin{aligned}&\textbf{let}\ \ \text{profe\_curso} = \text{profe} \underset{\text{legajo}}{\bowtie}_{\text{legajo}} \text{curso}\\ &\textbf{let}\ \ \text{pc\_analisis} = \sigma_{\text{curso.nombre}=\text{“Análisis de Datos”}}(\text{profe\_curso})\\ &\Pi_{\text{nombres}}(\text{pc\_analisis})\end{aligned}$$

---

5. Concatenación de tablas: es como la de listas nada mas que ahora el resultado tiene un esquema relacional. Para concatenar dos tablas, los esquemas deben ser compatibles (mismo tipo).

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

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

6. Resta: en una resta entre dos relaciones *r* y *s* se seleccionan las tuplas de r que **no** están en s.

$$r \setminus s :: (n_1 :: \tau_1, \ldots, n_N :: \tau_N)$$
$$r \setminus s = \sigma_{(\backslash \mathrm{t} \to \mathrm{t} \notin \mathrm{s})}(r)$$

7. Intersección: son las tuplas que estan en r y s si r y s son relaciones.

$$r \cap s :: (n_1 :: \tau_1, \ldots, n_N :: \tau_N)$$
$$r \cap s = \sigma_{(t \to t \in s)}(r)$$

8. Renombre: con renombre se pueden dar nombres a tablas para diferenciar los atributos al momento de hacer un producto cartesiano u otras operaciones. Notación: ρ_{p1} con ρ operador y p1 renombre.

- Si usamos *profe x profe*: en la tabla resultante las columnas no van a tener nombre.
- Lo que podemos hacer es darle nombre a la primera tabla profe:
- ρ_{p1} (profe) x profe. Aquí todas las columnas del resultado tienen nombre.

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

las columnas finales tienen nombres: profe.legajo, profe.nombres, profe.apellidos, profe.sueldo pero por cuestiones de espacio no se los puso así.

Ejemplo de uso:

Obtener el legajo de los profesores de mayor sueldo:

- Let proxpro = ρ_{p1} (profe) x profe
- Let sueldo_menor = $\Pi_{p1.legajo}$ ($\sigma_{p1.sueldo < profe.sueldo}$ proxpro)
- Π_{legajo} Profe \ sueldo_menor

9. Remover duplicados: La remoción de duplicados nos permite sacar aquellas tuplas que estén duplicadas en la tabla. Por ejemplo, si queremos obtener los legajos, sin duplicados, de les
profes de los cursos, podemos hacer lo siguiente:

- 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)

10. Agregación: son funciones que se aplican sobre listas. Entre ellas:

- count (cantidad elementos en una lista)

Ejemplo: Obtener la cantidad de departamentos con instructores.

$$\circ\ \mathcal{Y}_{(count \circ v)\ (dept\ name)}\ \text{instructor}$$

- sum (sumar valores que deben ser numéricos)
- avg (promediar valores numéricos)
- min (obtener el mínimo valor en la lista)
- max (obtener el máximo valor en la lista)

Ejemplo de agregación: Obtener los legajos de los profes que tienen el salario más alto.

$$\text{let maximo\_salario} = \gamma_{\max(\text{salario})}(\text{profe})$$
$$\text{let profe\_max} = \text{profe} \bowtie \rho_{(\text{salario})}(\text{maximo\_salario})$$
$$\Pi_{\text{legajo}}(\text{profe\_max})$$

Éstas pueden combinarse entre sí

Ejercicio: Obtener la cantidad de departamentos con instructores y máximo salario de los instructores

$$\circ\ \mathcal{Y}_{(count \circ v)\ (dept\ name),\ max(salario)}\ \text{instructor}$$

NOTAS DE EMA

REPASO DE LISTAS (FUNCIONAL)

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr h c []     = c
foldr h c (x:xs) = h x (foldr h c xs)
```
} FOLDR

```
map :: (a -> b) -> [a] -> [b]
map f []     = []
map f (x:xs) = (f x) : (map f xs)
```
} MAP

map f = foldr (\x xs -> f x : xs) []

```
∈ : a -> [a] -> Bool
x ∈ l = foldr (\a b -> a == x || b) False l
```
} PERTENECE

```
++ : [a] -> [a] -> [a]
l1 ++ l2 = foldr (:) l1 l2
```
} CONCATENACIÓN

$\forall l :: [a],\ P(l) \triangleq P([]) \wedge (\forall x :: a,\ l_2 :: [a],\ P(l_2) \Rightarrow P(x : l_2))$ } PRINCIPIO DE INDUCCIÓN SOBRE LISTAS

TABLAS Y SUS ESQUEMAS

nombre

$campo_1 :: t_1$	...	$campo_N :: t_N$
v_{11}	...	v_{1N}
⋮	⋱	⋮
v_{M1}	...	v_{MN}

} ESQUEMA → $R ::= nombre(campo_1 :: t_1, \dots, campo_N :: t_N)$
↳ Decimos que nombre :: R

} TUPLAS
↳ tienen tipo $dom(R) = t_1 \times \dots \times t_N$
↳ para tener el valor de un campo de la i-ésima tupla, tenemos "nombre.campo_i"
↳ Deben estar en el dominio definido (t_N)

CONCATENACIÓN DE TUPLAS

$\forall t = (a_1, \dots, a_N), t_2 = (b_1, \dots, b_M),\ (t, t_2) = (a_1, \dots, a_N, b_1, \dots, b_M)$
↳ el esquema de (t, t_2) es la concatenación de los esquemas de cada una

OPERADORES

PROYECCIÓN GENERALIZADA → Para obtener determinadas columnas de una tabla realizando, opcionalmente, algún cálculo sencillo

- **Def.:**

$$\Pi_{F_1,\dots,F_N}([]) = []$$

$$\Pi_{F_1,\dots,F_N}(x:xs) = (F_1\,x,\dots,F_N\,x) : \Pi_{F_1,\dots,F_N}(xs)$$

$$\Pi_{F_1,\dots,F_N}(r) = map\ (\backslash t \to (F_1\,t,\dots,F_N\,t))\ r$$

(se dice que la función F_i es proyectora si devuelve la col. i.e., sin operación)

- **Uso:** no vamos a escribir las funciones sino que trabajaremos con tipos implícitos:

```
sueldos_anuales = Π legajo, sueldo*13 (profe)
```

(profe: tabla de la que se extrae; legajo: columnas; sueldo*13: operación sencilla aplicada a una columna → su nombre se va a perder si NO es proyector)

sueldos_anuales	
legajo	
P_1	39000
...	...

SELECCIÓN → para filtrar las tuplas de acuerdo a un criterio

- **Def.:**

$$\sigma_p([]) = []$$

$$\sigma_p(t:r) = \text{if } p(t) \text{ then } t:\sigma_p(r) \text{ else } \sigma_p(r)$$

$$\sigma_p(r) = foldr\ (\backslash t\ r' \to \text{if } p\ t \text{ then } x:r' \text{ else } r')\ []\ r$$

- **Uso:** `cursos_potricia = σ legajo=="P" (curso)`

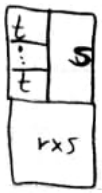
(cursos_potricia: tabla resultado; (curso): tabla que se filtra; legajo: columna; "P": valor; legajo=="P": condición booleana)

PRODUCTO CARTESIANO → Dadas dos tablas, junta cada tupla de la 1era con TODAS las de la 2da (también junta sus esquemas)

- Donde haya conflicto de nombres, se utilizará el nombre de la tabla para desambiguar (profe.legajo)
- **Def.:**

$$[] \times s = []$$

$$(t:r) \times s = (t \mathbin{++} s) : (r \times s)$$



- **Uso:** profe x curso → tablas que se quieran usar para el producto

REUNIÓN SELECTIVA → Producto cartesiano pero se seleccionan solo aquellas combinaciones que tienen mismo valor en la columna seleccionada
- pej., en vez de profe × curso, tenemos profe $\bowtie_{\text{legajo}}$ curso, el cual solo deja las combinaciones donde coinciden los legajos (⚠ importante para las FK)
- solo se deja la col. de la izq. (la que se busca que sea igual)

Def.: $r_{a_1,\dots,a_i} \bowtie_{b_1,\dots,b_i} s := (n_1::t_1,\dots,n_N::t_N, c_1,\dots,c_{M-i})$

$r_{a_1,\dots,a_i} \bowtie_{b_1,\dots,b_i} s = \Pi_{n_1,\dots,n_N,c_1,\dots,c_{M-i}}\left(\sigma_{a_1=b_1\wedge\dots\wedge a_i=b_i}(r\times s)\right)$ } c son las cols de s que no están en b

REUNIÓN NATURAL → Reunión Selectiva con TODAS las columnas que tengan igual nombre en ambas tablas

Def.: $r \bowtie s = r_{a_1,\dots,a_i} \bowtie_{a_1,\dots,a_i} s$ donde $\{a_1,\dots,a_i\} = \{n_1,\dots,n_N\} \cap \{m_1,\dots,m_M\}$

DEFINICIÓN LOCAL (let) → Para no escribir una chorrera de cosas, es como "meterlo en una variable auxiliar"

Def.: $\text{let } x = r \{ s \} = (\lambda x \to s)\, r$

(s) → fórmula a la que se le aplican las let's de arriba

CONCATENACIÓN → concatenación de tablas
- igual que concatenación de listas

Def.: $r +\!+ s = \text{foldr } (:)\ s\ r$
- las tablas deben cumplir que sus esquemas sean "compatibles", i.e., que tenga la misma cant. de campos con los mismos dominios
  - dos tablas son compatibles si sus esquemas lo son
- el resultado queda con el esquema de la izquierda

RESTA → quitar a una tabla las tuplas que existen en otra → ⚠ los esquemas deben ser compatibles y queda el de la izquierda

Def.: $r \setminus s = \sigma_{(t \to t \notin s)}(r)$ → seleccionamos de r todas las tuplas que NO estén en s

INTERSECCIÓN → solo deja tuplas coincidentes entre las tablas

⚠ esperan compatibles y queda el de la izquierda

↳ Def.: $r \cap s = \sigma_{(\lambda t \to t \in s)}(r)$

RENOMBRAMIENTO → solo modifica el esquema de una tabla para darle un nuevo nombre, tanto a la tabla en sí como a las campas

↳ Def.: [redacted] $\rho_{s(n_1', \dots, n_N')}(r) = r$ → con esquema nuevo $s(n_1', \dots, n_N')$ y mismos dominios

↳ Aclaraciones → si solo se quiere renombrar campos, puede omitirse el nombre de la tabla

↳ si se quiere renombrar solo el campo n_i, dejando a las otras igual, usaremos $n_i \leftarrow n_i'$

REMOVER DUPLICADOS → remueve tuplas duplicadas

↳ Def.:
$$v\ [] = []$$
$$v\ (t:r) = \text{if } t \in r \text{ then } v\ r \text{ else } t:(v\ r)$$

$$v(r) = \text{foldr } (\lambda t\ s \to \text{if } t \in s \text{ then } s \text{ else } (t:s))\ []\ r$$

AGREGACIÓN → para esto vamos a considerar las funciones { count, sum, avg, min, max } (count: lista; sum, avg, min, max: numéricos)

↳ $\gamma_{F_1(a_1), \dots, F_m(a_m)}(r) :: (\_ :: t_1', \dots, \_ :: t_m')$ → notar que no tienen nombre las columnas ⚠ devuelve un valor por columna que se pueda

$$\gamma_{F_1(a_1), \dots, F_m(a_m)}(r) = [F_1(\text{map } (\lambda t \to t.a_1)\ r), \dots, F_m(\text{map } (\lambda t \to t.a_m)\ r)]$$

AGRUPACIÓN → para usar agregación para conocer el valor para un determinado grupo de valores

↳ se define como la agregación para precedida la lista de columnas a [redacted] agrupar:

↳ ① se proyectan las columnas de agrupación de r y se le sacan los duplicados

② a cada tupla de la agrupación le anexamos el único resultado de la agregación sobre los registros de r filtrados para ese grupo en particular

Uso

"legajo $\gamma_{\text{count id}}$ (curso)"