Capítulo 3
Álgebra de Tablas

Operadores del Álgebra de Tablas

$nombre$			
$campo_1 :: \tau_1$	$campo_2 :: \tau_2$	$\ldots$	$campo_N :: \tau_N$
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}

Motivación

- Hay distintos problemas prácticos a resolver.
- **Problema 1**: Darle una semántica formal a SQL.
  - Ustedes vieron una descripción en lenguaje natural del significado de las construcciones de SQL.
- **Problema 2:** Dadas dos consultas en SQL chequear si son equivalentes.
  - Esto conviene porque una consulta puede ser más simple, más clara, más eficiente, etc. que otra.
  - **Equivalentes** quiere decir que para las mismas tablas de la base de datos dan los mismos resultados.
  - Puede incluso considerarse equivalencia mas débil: por ejemplo: dan tablas que solo difieren en orden de las tuplas o dan tablas que solo difieren en cantidad de tuplas repetidas.
  - Y no basta con testear las consultas con algunas tablas para probar esto.

Motivación

- **Problema 3**: Dada una consulta en SQL, obtener una consulta equivalente más eficiente.
  - A esto se le llama **optimización de consultas**.
  - En la práctica no siempre se busca la consulta más eficiente posible porque esto puede ser demasiado costoso.
- **Problema 4:** Dada una consulta en SQL (por ejemplo una óptima) evaluarla.
  - A esto se le llama **procesamiento de consultas**.

Motivación

- Todos los problemas anteriores se pueden atacar si disponemos de un álgebra de tablas.
- Un **álgebra de tablas** es un conjunto de tablas (cada una es una lista de tuplas) con ciertas operaciones.
    - Las operaciones son **definidas** usualmente recursivamente.
    - Las operaciones cumplen ciertas **propiedades** que se pueden probar usando las definiciones de las operaciones, otras propiedades y a veces también se usa el principio de inducción sobre listas.
- Una **consulta en el álgebra de tablas** va a ser una expresión que es una composición de las operaciones y las tablas de la base de datos.

Motivación

- **Problema 1**: Darle una semántica formal a SQL.
- **Solución**: una consulta de SQL tiene como semántica una expresión en el álgebra de tablas.
  - Una consecuencia de esto es que existe un mapeo de SQL a álgebra de tablas.
  - Y también se puede definir un mapeo de álgebra de tablas a SQL.
- **Problema 2:** Dadas dos consultas en SQL chequear si son equivalentes.
- **Solución**: mapear las consultas de SQL a consultas en álgebra de tablas y probar que esas expresiones del álgebra de tablas son equivalentes usando las propiedades del álgebra de tablas, las definiciones de los operadores, e incluso se puede usar inducción sobre listas (porque las tablas son listas).

Motivación

- **Problema 3**: Dada una consulta en SQL, obtener una consulta equivalente más eficiente.
- **Solución**: Se traduce la consulta SQL al álgebra de tablas y se busca otra consulta del álgebra de tablas más eficiente.
  - Esto se puede hacer usando propiedades del álgebra de tablas, heurísticas para la aplicación sistemática de esas propiedades, y programación dinámica (como se suele ver en un capítulo de optimización de consultas).

Motivación

- **Problema 4:** Dada una consulta en SQL (por ejemplo una óptima) evaluarla.
- **Solución**: Se traduce la consulta SQL a una consulta de álgebra de tablas; se evalúa la expresión del álgebra de tablas.
  - Para esto se pueden ejecutar implementaciones de los distintos operadores del álgebra de tablas a los que se les llama **operadores físicos**.
  - Para una operación del álgebra de tablas hay varios operadores físicos que asumen ciertas cosas sobre sus argumentos (p.ej. existencia de índices, cómo están organizados los archivos para las tablas de la base de datos, capacidades de memoria con que se cuenta, etc.)

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}	$\cdots$	v_{1N}
v_{21}	v_{22}	$\cdots$	v_{2N}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
v_{M1}	v_{M2}	$\cdots$	v_{MN}

Tablas y esquemas

- Definir esquemas para: biblioteca, bibliotecario, trabajaEn
- biblioteca(nombreBib:: string, calle::string, número: integer)
- bibliotecario(dni:: integer, antigüedad: integer)
- trabajaEn(dni::integer, nombreBib::string)
- Ejemplos de tuplas de estas tablas son:
- (“FaMAF”, “Medina Allende”, 0) ∈ biblioteca
- (1232, 5) ∈ bibliotecario
- $(1232, \text{“FaMAF”}) \in \text{trabajaEn}$
- $\text{biblioteca} = [(\text{“FaMAF”}, \text{“Medina Allende”}, 0)]$

Motivación para definir álgebra de tablas

- **Ejemplo**: Obtener los bibliotecarios que trabajan y que tienen una antigüedad mayor a 5 años.

  SELECT nombre, DNI, nombreBib

  from bibliotecario, trabajaEN

  where bibliotecario.DNI = trabajaEN.DNI and antigüedad > 5

- Esto se puede **expresar** como composición de tres operadores:
  - **Producto cartesiano**: para combinar tuplas de bibliotecario con tuplas de trabajaEn.
  - **Filtrar**: filtrar según un predicado. En el caso anterior:

    P =def bibliotecario.DNI = trabajaEN.DNI and antigüedad > 5)
  - **Proyectar**: extraer de una tupla valores de ciertos atributos. En el caso anterior: considerar solo: nombre, DNI y nombreBib.
- Proyectar $_{\text{nombre, DNI, nombreBib}}$(Filtrar$_{P}$ (producto cartesiano (bibliotecario , trabajaEn)))

Motivación para definir álgebra de tablas

- **Idea**: una consulta en SQL se expresa como composición de operadores sobre tablas.
  - Las tablas se pueden pensar como listas de tuplas
  - Un operador recibe tablas y devuelve una tabla.
  - Los operadores se pueden pensar como funciones sobre listas de tuplas.
  - Podemos usar todo lo que vimos sobre definición de funciones sobre listas.

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

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

legajo = (\t -> t.legajo)

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

Proyección generalizada

- Genaralizando:

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

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

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

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.Ad)

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 $\sigma_P\ r$

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.

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:

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

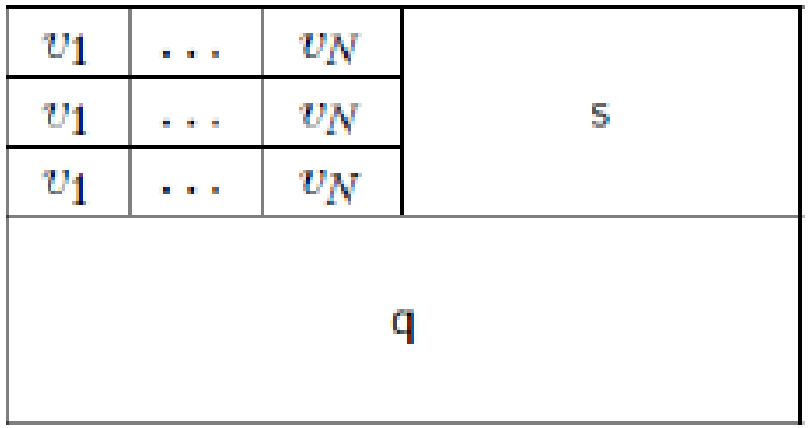
- Si hay conflictos de nombres de atributos se usa el nombre de la tabla para desambiguar.
- Una tabla puede contener columnas sin nombre. Para ese caso se usa: _: T
- Voy a poder aplicar el producto cartesiano para tablas donde no todas las columnas tienen nombre.

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"
...	...	...	...	...	...	...

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



Producto cartesiano

[] x s = []

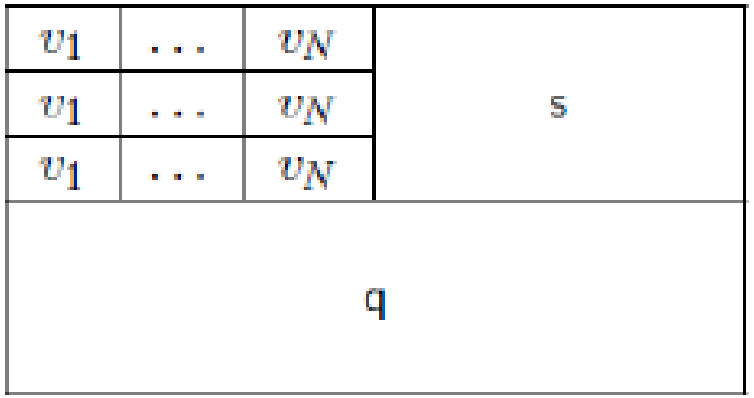
(t:r) x s = anexar s t (rxs)

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

Definimos la concatenación de tuplas como la tupla:

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

anexar s t q = (map (\t' -> t ; t') s) ++ q



Ejemplo

- 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"

- Se quiere calcular el resultado de la consulta: Encontrar los nombres de los profes que dictan 'Análisis de datos'.
- Aplicar operadores de álgebra de tablas uno a uno hasta obtener el resultado deseado.

Ejemplo

- Para obtener el resultado hay que combinar ambas tablas porque se refiere a atributos de ambas tablas. El producto cartesiano profe x curso es:

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

- Nos interesan las filas que cumplen: Q = profe.legajo = curso.legajo.
- El resultado de σ_Q (profe x curso) es el siguiente:

Ejemplo

–						
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"

A esta tabla la llamamos T.

Nos interesan las filas con curso Análisis de Datos.

P = curso.nombre = 'Análisis de Datos'

El resultado de hacer σ_P (T) es:

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

Ejemplo

- Entonces aplicando la proyección según nombres obtenemos:

–
nombres
“Patricia”

- O sea, hicimos la consulta: Πnombres (σ_P (σ_Q (profe x curso)))
- Esta consulta se puede hacer de otra forma usando la propiedad:

$$\sigma_P(\sigma_Q(r)) = \sigma_{P \wedge Q}(r)$$

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

- ¿Cuál es más eficiente?

Relación entre SQL y Álgegra de Tablas

select $A1,\ldots,An$

from $r1,\ldots,rn$

where P

- Es equivalente a:

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

Reunión selectiva

- En el ejemplo anterior unimos las tablas mediante el campo legajo.
- La operación de combinar tablas por atributos en común es muy utilizada.
- Otro ejemplo: Sean las tablas:
  - Persona(DNI, nombre, apellido)
  - Biblioteca(nombreBib, calle, número)
  - Bibliotecario(antigüedad, DNI)
- Considerar la consulta: Encontrar nombre, apellido y DNI de bibliotecarios.
- Esto nos obliga a juntar las tablas persona y bibliotecario por atributo DNI.
- $\Pi_{\text{nombre, apellido, DNI}}\ (\sigma_{\text{persona.DNI = bibliotecario.DNI}}\ \text{(persona x bibliotecario))}$
- Así introducimos un operador llamado reunión selectiva que hace esto:
- $\Pi_{\text{nombre, apellido, DNI}}\ \text{(persona}\ _{\text{DNI}} \bowtie_{\text{DNI}}\ \text{bibliotecario)}$

Reunión selectiva

- Volviendo a la consulta anterior:

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

- Usamos reunión selectiva:

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

- 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

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

Reunión selectiva

- La consulta anterior:

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

- es una reunión natural:

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

Reunión selectiva

- 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

Ejercicio

persona(DNI, nombre,apellido)
libro(título,ISBN,editorial,edición)
libroBib(numInv,ISBN,nombreBib)
prestadoA(numInv,nombreBib,DNI)
socio(posición,DNI)
inscriptoEn(DNI,nombreBib)

- Obtener los ISBN de los libros prestados a *Juan Pérez* en la biblioteca de "FaMAF"

- Tablas para hacer la consulta
- prestadoA, libroBib, persona

$$\Pi_{ISBN} \left(\sigma_{\text{nombre = "Juan" y apellido = "Perez"} \wedge \text{ nombreBib = 'FaMAF'}} \left((\text{libroBib} \bowtie \text{prestadoA}) \bowtie \text{persona} \right) \right)$$

Construcción de consultas complejas

- Usaremos el operador **let**.
- Sea la consulta: encontrar los nombres de los profesores que enseñan análisis de datos.
- Esta estaba expresada como:

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

- Se puede escribir así:

$$\begin{aligned}
&\textbf{let} \quad \text{profe_curso} = \text{profe} \overset{}{\bowtie}_{\text{legajo}=\text{legajo}} \text{curso} \\
&\textbf{let} \quad \text{pc_analisis} = \sigma_{\text{curso.nombre}=\text{“Análisis de Datos”}}(\text{profe_curso}) \\
&\Pi_{\text{nombres}}(\text{pc_analisis})
\end{aligned}$$

Construcción de consultas complejas

- **let** x = r **in** s = (\x -> s) r
- Ejemplo: let x = 45 in x+1 = (\x -> x+1) 45 = 45 +1 = 46
- **Ejemplo**: para la consulta anterior, segundo let:

$$\text{let} \quad \text{pc\_analisis} = \sigma_{\text{curso.nombre}=\text{"Análisis de Datos"}}(\text{profe\_curso})$$
$$\Pi_{\text{nombres}}(\text{pc\_analisis})$$

- (\pc_análisis -> Π_{nombre} (pc_análisis)) $\sigma_{\text{curso.nombre='Analisis de datos'}}$ (profe_curso)

  = Π_{nombre} ($\sigma_{\text{curso.nombre='Analisis de datos'}}$ (profe_curso))

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**.

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

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

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

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.

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

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

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.

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

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

Renombre

- Sea la tabla:

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

- Sea la consulta obtener el legajo de los profesores de mayor sueldo.
- 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.

Renombre

- 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

Renombre

- Sea la consulta: obtener el legajo de los profesores con sueldo anual > 60000.
- Tenemos de antes:

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

- No podemos usar esta consulta y aplicarle selección.
- Para eso necesitamos darle nombre a la segunda columna de esta tabla.
- Esto se puede hacer con un operador de renombramiento más general que el anterior.

Renombre

- Sea la tabla:

$$\text{Sea } r(n_1 :: \tau_1, \ldots, n_N :: \tau_N).$$
$$\rho_{s(n'_1,\ldots,n'_N)}(r) :: s(n'_1 :: \tau_1, \ldots, n'_N :: \tau_N)$$
$$\rho_{s(n'_1,\ldots,n'_N)}(r) = r$$

- No lo ponemos, pero en el esquema de r algunos campos pueden no tener nombre.
- Entonces obtener el numero de legajo de profes con sueldo anual > 60000:
- $\sigma_{\text{sueldoAnual} > 600000}$ ($\rho_{\text{profe2(nro\_legajo, sueldAnual)}}$ ($\Pi_{\text{nro\_legajo, sueldo * 13}}$ (profe)))

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)

Ordenamiento

- Especificamos **ordenamiento** como 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.

$$O^{>}_{a_1,\ldots,a_N}(r) = \text{reversa}\ (O_{a_1,\ldots,a_N}(r))$$

```
reverse []     = []
reverse (x:xs) = reverse xs ++ [x]
```

- Cuando queramos ordenar toda la tabla, omitiremos la lista de las columnas.

Agregación

- **Funciones de agregación** son las siguientes funciones sobre listas:
  - *count*:: [a] -> integer, para contar la cantidad de elementos en la lista.
  - *sum*: para sumar los valores que deben ser numéricos.
  - *avg*: para promediar los valores que deben ser numéricos
  - *min*: para obtener el mínimo valor en la lista.
  - *max*: para obtener el máximo valor en la lista.

Agregación

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

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

- La consulta del primer let puede generalizarse de la siguiente manera:
  - $\gamma_{f1(a1),\ \ldots,\ fm(am)}\ (r) :: (\_:: \tau'_1, \ldots, \_::\tau'_m)$
  - Donde f_i es función de agregación, o f_i es de la forma: $f \circ v$, para f función de agregación

Agregación

- **Ejemplo**: Obtener la cantidad de departamentos con instructores.
  - $\mathcal{Y}_{(count \circ v)\ (dept\ name)}$ instructor
- **Ejercicio**: Obtener la cantidad de departamentos con instructores y máximo salario de los instructores
  - $\mathcal{Y}_{(count \circ v)\ (dept\ name),\ max(salario)}$ instructor

Agregación

- Sea la tabla r:

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

$\mathcal{G}_{\mathbf{sum(c)}}(\mathbf{r})$

27

Agregación

- Π_{ai} (r) es una lista de tuplas para la columna i de r, cada tupla tiene un valor.
- Una función de agregación se aplica a una lista de elementos y no a una lista de tuplas.
- Queremos aplicar una función de agregación a la lista de elementos de una columna de r.
- La función que obtiene la lista de elementos de la columna i de r es:
  - map (\t-> t.ai)
- $\Upsilon_{f1(a1), \ldots, fm(am)}$ (r) = [(f_1 (map (\t-> t.a_1) r), …, f_m (map (\t-> t.a_m) r)]

Agregación

- **Ejemplo**: Sea la tabla:
  - *instructor*(*ID*, *name*, *dept name*, *salary*)
- Encontrar el salario promedio de cada departamento.

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

- $_{dept\_name}\gamma_{\textbf{avg}(salary)}(instructor)$

Agregación

- **Ejemplo**: Sean las tablas:

  bibliotecario(antigüedad,DNI)

  trabajaEn(DNI,nombreBib)

- Obtener para cada biblioteca el promedio de las antigüedades de los bibliotecarios que trabajan en ella.
- $_{nombreBib}\gamma_{avg(antigüedad)}\ (bibliotecario \bowtie trabajaEn)$

Agregación

- Sea la relación r:

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

- ¿Cuál es el resultado de ${}_{A}\gamma_{\mathbf{max}(C)}(r)$?

A	
α	7
β	10

Agregación

- Para definir el operador de agregación con agrupación en general seguimos los siguientes pasos:
  - calculamos grupo dentro de tabla,
  - al grupo le aplicamos funciones de agregación y
  - al resultado le concatenamos el valor de la tupla usada para caracterizar el grupo.

Agregación

- Calculamos grupo dentro de tabla *r*:
  - Para t ∈p = *v*($\Pi_{g1,\ldots,gN}$ (r)): $\sigma_{g1 = t.g1 \wedge \ldots \wedge gN = t.gN}$ (r)
- Al grupo le aplicamos funciones de agregación:
  - $\mathcal{Y}_{f1(n1), \ldots, fM(n^M)}$ ($\sigma_{g1 = t.g1 \wedge \ldots \wedge gN = t.gN}$ (r)))
- al resultado le concatenamos el valor de la tupla usada para caracterizar el grupo.
  - t; head($\mathcal{Y}_{f1(n1), \ldots, fM(nM)}$ ($\sigma_{g1 = t.g1 \wedge \ldots \wedge gN = t.gN}$ (r))))
  - head(x:xs) = x
  - head([]) = undefined

Agregación

- Juntando todo:

$$g_1,\ldots,g_N\,\gamma_{f_1(n_1),\ldots,f_M(n_M)}(r) =$$

= **let** p = $v(\Pi_{g1,\ldots,gN}(r))$ **in**

map (\t -> (t; head($\gamma_{f1(n1),\ \ldots,\ fM(nM)}$ ($\sigma_{g1 = t.g1 \wedge \ldots \wedge gN = t.gN}$ (r))))) p

Agregación

- **Ejercicio**: Sean las tablas:

  bibliotecario(antigüedad,DNI)

  trabajaEn(DNI,nombreBib)

  - Obtener el máximo de las antigüedades promedio de las bibliotecas.

Ejercicio

- **Probar:**

$$\sigma_P(\sigma_Q(r)) = \sigma_{P \wedge Q}(r)$$

Ejercicio

- Definir **reunión externa a la izquierda** usando operadores del álgebra de tablas.