Diseño de Bases de Datos

Álgebra/Cálculo Relacional

Lenguajes de consulta (QL)

- Leguajes de consulta
  - permiten la manipulación y consulta de datos de las BD
- Lenguajes relacionales *formales*
  - manipulación de relaciones para obtener una relación resultado, que es el resultado de una consulta
  - fundamentos formales basados en la lógica
  - permiten una muy potente optimización
  - base de los lenguajes *reales* de implementación (SQL)
- QL no es un lenguaje de programación
  - sólo está pensado para soportar el acceso a los datos
  - no para cálculos complejos / computación

QLs relacionales formales

- Álgebra relacional
  - operacional/imperativo —*cómo hago que X ocurra*
  - muy útil para representar *planes de ejecución:* secuencia de operaciones que hacen que se produzca el resultado
  - base de la ejecución interna de una consulta, lo cual es muy útil para un usuario avanzado / administrador (optimización)
- Cálculo relacional
  - declarativo —*cómo reconozco que X ha ocurrido*
  - permite a los usuarios describir lo que buscan: declaración de los resultados en los que se está interesado

Ejemplo: reservas de vehículos

C (clientes)

Cliente

idc	nombre	cat	edad
22	Davila	7	45
29	Bravo	1	32
31	Lorenzo	8	24
32	Alvarez	8	31
58	Rubio	4	67
64	Huerga	2	18
71	Zurro	10	45
74	Huerga	9	35
85	Arnaud	3	25
95	Benitez	3	63

R (reservas)

Reserva

idc	matr	fecha
22	101	2009-11-07
64	101	2010-12-28
22	102	2010-04-21
31	102	2012-01-14
64	102	2010-04-11
22	103	2009-11-07
31	103	2011-07-19
74	103	2009-09-13
22	104	2012-01-01
31	104	2006-12-09

V (vehículos)

Vehiculo

matr	marca	color
101	BMW	azul
102	Lancia	rojo
103	Seat	verde
104	BMW	rojo

C2 (clientes también)

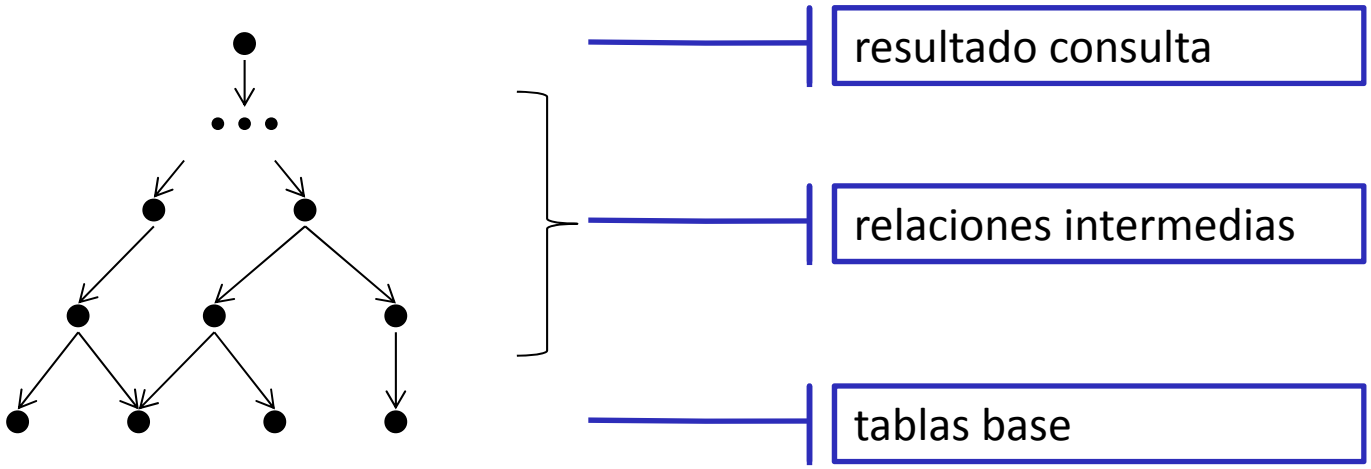
Cliente

idc	nombre	cat	edad
22	Davila	7	45
85	Arnaud	3	25
95	Benitez	3	63

- una consulta se aplica sobre *instancias* de relación, siendo el resultado también una relación —los esquemas son fijos
- referencia a los campos por nombre o por posición
- relaciones intermedias *heredan* los nombres de los campos

Álgebra relacional

- Consulta = árbol relacional
  - tomando como punto de partida las tablas base —que nunca son modificadas,
  - se aplican una secuencia de operadores —unarios o binarios,
  - cada uno de los cuales devuelve una relación intermedia,
  - cuya relación resultante final es el resultado de la consulta



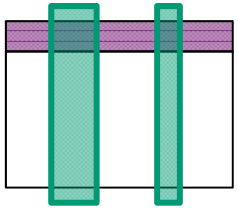
Operadores

» cada operador devuelve una relación, por lo que pueden ser *compuestos*

- Básicos
  - selección: σ, selecciona un subconjunto de filas
  - proyección: π, extrae columnas
  - unión: $\cup$, tuplas en relación1 y relación2
  - diferencia: $-$, tuplas en relación1, pero no en relación2
  - producto cartesiano: $\times$, combina dos relaciones
- Adicionales, se pueden definir en término de los básicos, pero son de gran utilidad y uso (esp. *join*)
  - intersección: $\cap$, tuplas en ambas relación1 y relación2
  - división: $/$, *cociente* de relaciones con *factores* comunes
  - join: $\bowtie$, combinación condicionada

Proyección y selección

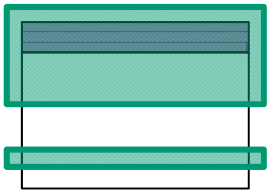
$\pi_{\text{listaAtrib}}(\text{Rel})$ » mantiene los campos que están en la lista de proyección



– elimina los duplicados (conjuntos de tuplas)

– los sistemas reales sólo eliminan duplicados si se pide

$\sigma_{\text{cond}}(\text{Rel})$ » selecciona las tuplas con cumplen la *condición de selección*



– no hay duplicados en el resultado

– el esquema es igual que el de la relación original

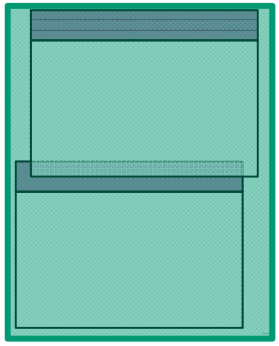
– composición de operadores

$\pi_{nombre,cat}\ (\sigma_{\text{cat}>7}(\text{c2}))$

nombre	cat
Izarra	8
Pozo	10

Unión, intersección y diferencia

R1 ∪ R2

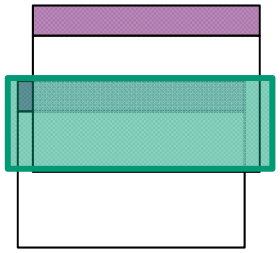


» operaciones clásicas de unión, intersección y diferencia de conjuntos (de tuplas, en este caso)

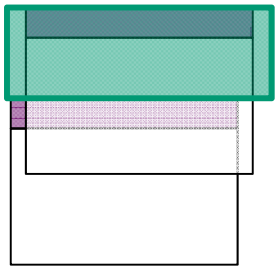
– en estos operadores, el esquema tiene que ser *compatible*, manteniéndose en el resultado

– eliminación de duplicados

R1 ∩ R2



R1 – R2



c1 ∪ c2

idc	nombre	cat	edad
29	Rojo	6	23
33	Izarra	8	42
41	Heredia	2	23
57	Pozo	10	23
23	García	7	31
96	Aragón	8	58

Producto cartesiano

R1 × R2

A
a
b

×

N
1
2
3

=

A	N
a	1
a	2
a	3
b	1
b	2
b	3

» combinación de todos los posibles pares de tuplas de las dos relaciones

– esquema combinado de las dos relaciones, con renombrado de atributos —(atr)— para evitar coincidencias

– producto de tuplas, suma de esquemas

c1 × r1

idc	nombre	cat	edad	(idc)	matr	fecha
23	García	7	31	23	101	10/07/2011
23	García	7	31	57	201	01/10/2011
96	Aragón	8	58	23	101	10/07/2011
96	Aragón	8	58	57	201	01/10/2011
57	Pozo	10	23	23	101	10/07/2011
57	Pozo	10	23	57	201	01/10/2011

¿qué tipo de resultado nos gustaría obtener en realidad?

Joins: combinaciones cond.

R1 ⋈R2

A	B
a	1
b	2

⋈

B	C
1	x
1	y
3	z

=

A	B	C
a	1	x
a	1	y

- » combinación por igualdad de campos coincidentes —join *natural*
- soporta gran parte de las consultas que involucran a dos o más tablas
- combina datos de entidades que residen en varias tablas (normalización, p.e.)
- operación difícil de implementar eficientemente por un DBMS, lo que genera un problema intrínseco de rendimiento

c1 ⋈ r1

idc	nombre	cat	edad	matr	fecha
23	García	7	31	101	10/07/2011
57	Pozo	10	23	201	01/10/2011

...

- Join-theta, R1 $\bowtie_{cond}$ R2
  - producto cartesiano que se restringe con una condición adicional
- Equijoin, R1 $\bowtie_{=}$ R2
  - la condición sólo involucra igualdades
  - los campos que se igualan sólo aparecen una vez en el esquema resultante
- Natural, R1 $\bowtie$ R2
  - equijoin sobre todos los campos comunes

...

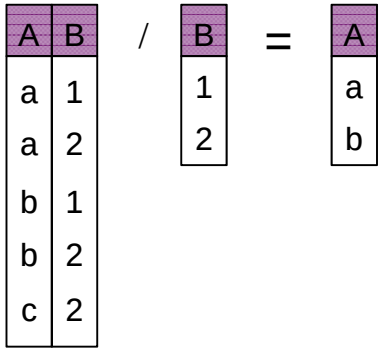
- Semijoin, R1 $\triangleright$ R2
  - tuplas de R1 que participan en un join con R2
  - π_{atrR1} (R1 $\bowtie$ R2)
- Outer (left,right), R1 $\bowtie$ R2
  - (left) las tuplas de R1 que no tienen correspondencia en los campos comunes con R2, también son incluidas en el resultado, completando con nulos

c1 $\bowtie$ r1

idc	nombre	cat	edad	matr	fecha
23	García	7	31	101	10/07/2011
96	Aragón	8	58	*null*	*null*
57	Pozo	10	23	201	01/10/2011

División

R1 / R2



A	B
a	1
a	2
b	1
b	2
c	2

/

B
1
2

=

A
a
b

» (proyección de) tuplas de R1 que figuran en combinación con todas las tuplas de R2

– esquema reducido a los campos que combinan con los de la otra relación
– se puede expresar en términos de los operadores básicos
– útil para consultas del tipo: *clientes que han reservado todos los coches ...*

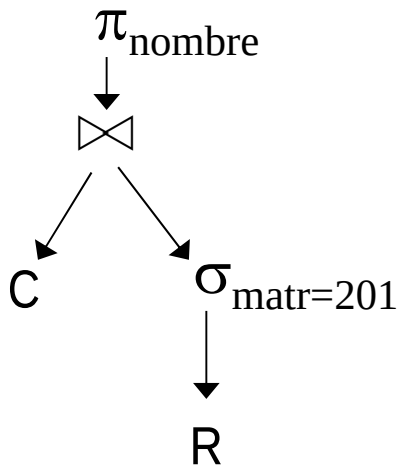
Ejemplos de consultas

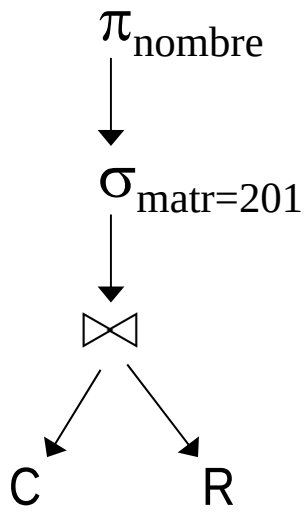
Cliente (idc,nombre,cat,edad)

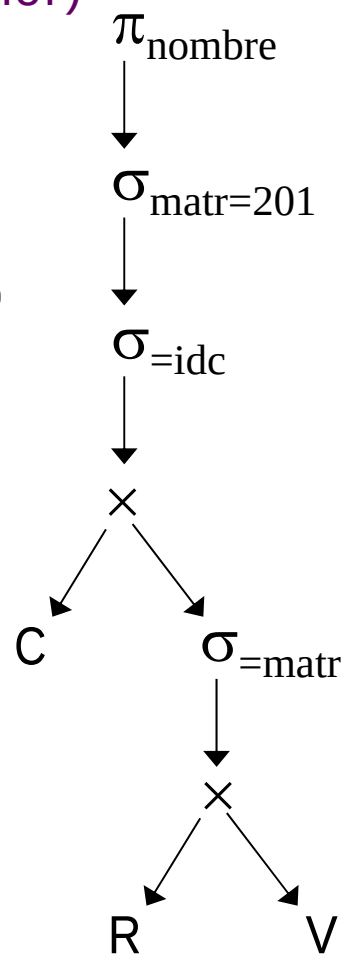
Vehículo (matr,marca,color)

Reserva (idc,matr,fecha)

- Clientes que han reservado el coche 201
  - $\pi_{nombre}\ (\sigma_{matr=201}\ (\ \sigma_{=idc}\ (\ C \times (\ \sigma_{=matr}\ (\ R \times V\ )))))$
  - $\pi_{nombre}\ (\sigma_{matr=201}\ (\ C \bowtie R\ ))$
  - $\pi_{nombre}\ (C \bowtie \sigma_{matr=201}\ (R))$







...

- Clientes que han reservado un coche rojo
  - π_{nombre} (($\sigma_{color='rojo'}$ V $\bowtie$ R) $\bowtie$ C)
  - π_{nombre} (π_{idc} (π_{matr} $\sigma_{color='rojo'}$ V $\bowtie$ R) $\bowtie$ C)

  - el optimizador se encarga de encontrar la mejor forma de resolver la consulta
  - (en principio) no es dependiente de la forma del SQL
  - el color sólo está en la relación Vehículo, por lo que es necesario en join extra

...

- Clientes que han reservado un coche rojo o verde
  - ρ(rvVeh, ($\sigma_{color='rojo' \text{ v } color='verde'}$ V))
  - π_{nombre} ((rvVeh $\bowtie$ R) $\bowtie$ C)

  - operador de renombrado
  - también se podría haber utilizado la unión

...

- Clientes que han reservado un coche rojo y verde
  - ρ(rRes, π_{idc} (($\sigma_{color='rojo'}$ V) $\bowtie$ R))
  - ρ(vRes, π_{idc} (($\sigma_{color='verde'}$ V) $\bowtie$ R))
  - π_{nombre} ((rRes $\cap$ vRes) $\bowtie$ C)

  - el planteamiento anterior (o), ¡no funciona!
  - está basado en la identificación de clientes según sus reservas de vehículos
  - la intersección debe hacerse sólo sobre los *idc;* en caso contrario la intersección sería vacía

...

- Clientes que han reservado todos los vehículos
  - ρ(idcs, ($\pi_{idc,matr}$ R) / (π_{matr} V))
  - π_{nombre} (idcs $\bowtie$ C)
- Clientes que han reservado todos los BMW
  - ρ(idcs, ($\pi_{idc,matr}$ R) / (π_{matr} ($\sigma_{marca='BMW'}$ V)))
  - π_{nombre} (idcs $\bowtie$ C)

Qué no solemos hacer en AR

- Ordenaciones
  - Cliente con mayor categoría
  - Coche con mayor número de reservas
- Agrupaciones
  - Particionado de tuplas y aplicación de operadores sobre esos grupos
    - Edad media de los clientes por categoría

Cálculo relacional

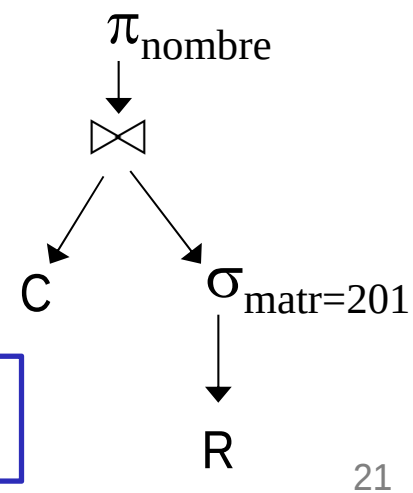
- Consulta = condición (predicado)
  - sobre las tuplas que deben incluirse en la contestación a la consulta —aquellas que evalúan la condición a *true*
  - $\{<x_1,\ldots,x_n> \mid p(<x_1,\ldots,x_n>)\}$
  - las expresiones son fórmulas el algún tipo de *lógica de predicados de primer orden*, con variables a las que puedo asignar tuplas (TRC) o elementos de tuplas (DRC)
  - se utilizan: variables, constantes, comparadores, conectores lógicos y cuantificadores
  - las fórmulas se definen recursivamente: pertenencia de tuplas a relaciones, comparación de valores, conectivas lógicas
  - entorno útil para restricciones / aserciones

DRC

- Variables de campos/dominios
  - las variables son asignadas con elementos del dominio de los campos de las tuplas
  - $\{<i,n,c,e> : Cliente \mid c>7\}$
  - las variables i,n,c,e se *vinculan* a los campos de tuplas Cliente
  - no es una restricción para todas las tuplas cliente (aserciones), sino una condición de consulta
  - $\{<i,n,c,e> : Cliente \mid c>7 \wedge \exists <ir,mr,f> : Reserva \mid ir=i \wedge mr=201\}$

encuentra una reserva que coindice (*join*) con el cliente en consideración

comparar con el álgebra relacional



Lenguajes de restricciones

- ¿Alguna relación con OCL?
  - no serán restricciones del esquema (aserciones), sólo restricciones de las tuplas resultantes de una consulta
  - OCL utiliza la *navegación* en vez de la combinación explícita de relaciones por igualdad de campos
  - no se representa la proyección
  - context Cliente inv:
    self.reserva -> exists (r|r.matr=201)
  - context Cliente inv:
    self.reserva.vehículo -> exists (v|v.color='rojo' $\vee$ v.color='verde')
  - context Cliente inv:
    self.reserva.vehículo -> exists (v1,v2| v1$\neq$v2 $\wedge$
    v1.color='rojo' $\wedge$ v2.color='verde')

...

» Clientes que han reservado todos los BMW

– $\{ \langle i,n,c,e \rangle : \text{Cliente} \mid \forall \langle m,ma,c \rangle : \text{Vehículo} \mid ma \neq \text{'BMW'} \vee \exists \langle ir,mr,f \rangle : \text{Reserva} \mid ir = i \wedge mr = m \}$

sólo se *declara* una condición de pertenencia al resultado;
en contraste con el álgebra, donde se *detalla* la secuencia de operaciones

» Y en OCL?

– context Cliente inv:
self.reserva.vehículo ->
forAll(v | v.marca='BMW')

NO: todos los reservados son BMW !!

– context Cliente inv:
self.reserva.vehículo =
Vehículo.allInstances -> select(v|v.marca='BMW')

la notación punteada evita las igualdades de combinación (join)

Potencia expresiva

- Capacidad de expresar consultas
  - cualquier consulta que se pueda expresar en AR, se puede expresar en CR
  - ¿y viceversa?
    - CR permite fórmulas *inseguras,* con infinitos resultados en dominios distintos al referenciado
      $\{<i,n,c,e> \mid \{\neg(<i,n,c,e> \in \text{Cliente})\}$
    - cualquier consulta segura en CR, se puede expresar en AR
- Relacionalmente completo
  - potencia expresiva del álgebra/cálculo relacional
  - los lenguajes de consulta prácticos (SQL) deben ser relacionalmente completos
  - de hecho, permiten expresiones que no permite el AR