Administración y Programación de Bases de Datos

Unidad 3: Optimización de Consultas

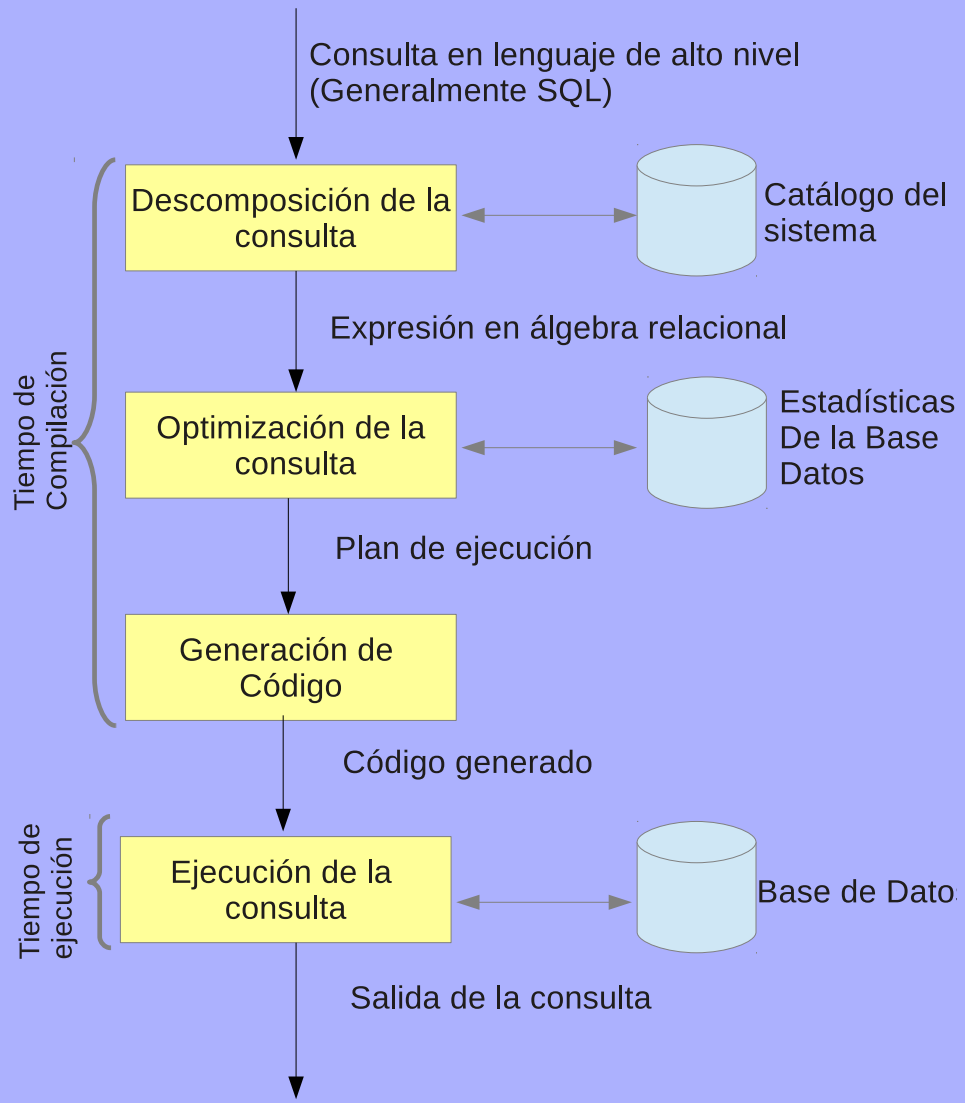
Gilberto Gutiérrez

DCCTI/UBB

Otoño 2016

Contenido

Fases del procesamiento de consultas



Traducción de Consultas SQL al Álgebra Relacional

Example

```
SELECT Apellido1, Nombre
FROM   EMPLEADO
WHERE  Sueldo > (SELECT MAX(Sueldo)
                 FROM EMPLEADOC
                 WHERE Dno=5);
```

Traducción de Consultas SQL al Álgebra Relacional

Bloque interno

```
SELECT MAX(Sueldo)
FROM EMPLEADOC
WHERE Dno=5;
```

$\mathcal{F}MAX\ Sueldo(\sigma_{dno=5}(\text{EMPLEADO}))$

Bloque externo

```
SELECT Apellido1, Nombre
 FROM   EMPLEADO
 WHERE  Sueldo > c
```

$\pi_{Apellido1,Nombre}(\sigma_{Sueldo>c}(\text{EMPLEADO}))$

Merge Sort Externo

1. Fase de Ordenación. Ordena pequeños subarchivos (porciones) del archivo. Sea b el número de bloques del archivo y n_B los bloques de memoria temporal disponible, entonces el número de subarchivos iniciales, n_R

$$n_R = \left\lceil \frac{b}{n_B} \right\rceil$$

2. Fase de Mezclado. Las porciones ordenadas se mezclan en una o más pasadas. Sea d_M el número de porciones que pueden ser mezcladas en una pasada.

$$d_M = \min\{(n_B - 1), n_R\}$$

En $n_B - 1$, se resta 1, pues se usa un bloque para almacenar el resultado de la mezcla.
El número de pasadas es:

$$\lceil \log_{d_M}(n_R) \rceil$$

Algoritmo Merge Sort Externo

1: **MergeSortExterno(Archivo A)**
2: {**Fase de Ordenación**}
3: $i \leftarrow 1$;
4: $j \leftarrow b$; {Tamaño en bloques del archivo A}
5: $k \leftarrow n_B$; {Tamaño en bloques del almacenamiento temporal}
6: $m \leftarrow \left\lceil \frac{j}{k} \right\rceil$; { $m = n_R$}
7: **while** $i \leq m$ **do**
8: Leer los siguientes k bloques desde el archivo A en el buffer o, si quedan menos que k, leer los bloques restantes.
9: Ordenar los registros en el buffer y escribirlos como subarchivo temporal.
10: $i \leftarrow i + 1$
11: **end while**

Algoritmo Merge Sort Externo (cont.)

1: {Fase de mezcla }
2: $i \leftarrow 1; j \leftarrow m;$
3: $p \leftarrow \lceil \log_{k-1} m \rceil$; {$p$ es el número de pasadas en la fase de mezcla}
4: **while** $i \leq p$ **do**
5: $n \leftarrow 1;$
6: $q \leftarrow \left\lceil \frac{j}{(k-1)} \right\rceil$ {q número de subarchivos a escribir en esta pasada}
7: **while** $n \leq q$ **do**
8: Leer los siguientes $k-1$ subarchivos o los subarchivos restantes (de pasadas previas), un bloque a la vez.
9: Mezclar y escribir como nuevo subarchivo un blqoue cada vez.
10: $n \leftarrow n+1$
11: **end while**
12: $j \leftarrow q$
13: $i \leftarrow i+1$
14: **end while**

Algoritmo Merge Sort Externo (cont.)

Costo:

$$t(b) = 2b + 2b \log_{d_M} b$$

$$t(b) = O(b \log_{d_M} b)$$

Sean n el número de registro en el archivo y f el factor de bloque ($b = \frac{n}{f}$)

$$t(n) = O(n \log_{d_M} n)$$

Algoritmos para SELECT

Algunas consultas SELECT

- OP1: $\sigma_{Dni='12345'}(EMPLEADO)$
- OP2: $\sigma_{NumeroDepto>5}(DEPARTAMENTO)$
- OP3: $\sigma_{Dno=5}(EMPLEADO)$
- OP4: $\sigma_{Dno=5\ AND\ Sueldo>30000\ AND\ Sexo='F'}(EMPLEADO)$
- OP5: $\sigma_{DniEmpleado='1234'\ AND\ NumProy=10}(TRABAJA\_EN)$

Algoritmos para SELECT

Métodos de búsqueda en una selección simple

- **S1 – Búsqueda lineal (fuerza bruta)**
- **S2 – Búsqueda binaria**. Ejemplo OP1 si `EMPLEADO` está ordenado por *Dni*.
- **S3 – Utilización de índice primario.** Ejemplo OP1 si `EMPLEADO` tiene un índice por *Dni*
- **S4 – Utilización de índice primario para encontrar varias tuplas.** Ejemplo OP2.
- **S5 – Utilización de índice de agrupación para encontrar varias tuplas.** Ejemplo OP3 con índice de agrupación por *Dno*.
- **S6 – Utilización de índice secundario (árbol B^+) sobre una comparación de igualdad.**

Algoritmos para SELECT

Métodos de búsqueda en selecciones complejas

- **S7 – Selección conjuntiva utilizando un índice individual**
- **S8 – Selección conjuntiva utilizando índices compuesto**. En OP5, si existe un índice compuesto por (*DniEMpleado*, *Numproy*)
- **S9 – Selección conjuntiva mediante intersección de punteros a tuplas**. índices disponibles para más de uno de los atributo implicados en las consultas.

Selectividad(s) de una consulta

Relación entre el número de tuplas que satisfacen la condición (WHERE) y el total de tuplas de la tabla.
Sea $tt = |R(r)|$ el total de tuplas de la instancia r de la relación R.

- Si el atributo de la selección de R es clave y la condición sobre el atributo es de igualdad, entonces $s = \frac{1}{tt}$
- Condición de igualdad sobre un atributo con i valores distintos, $s = \frac{\frac{tt}{i}}{tt} = \frac{1}{i}$. Se asume distribución uniforme de los distintos valores. Un total de $\frac{tt}{i}$ tuplas cumplirán la condición de igualdad sobre el atributo.
- En general el número estimado de tuplas que satisface una condición de selección es $tt * s$.

Algoritmos para la operación de JOIN

EQUIJOIN (dos vías)

$$R \bowtie_{A=B} S$$

- OP6: $EMPLEADO \bowtie_{Dno=NumeroDpto} DEPARTAMENTO$
- OP7: $DEPARTAMENTO \bowtie_{DniDirector=Dni} EMPLEADO$

EQUIJOIN (dos vías)

- J1 – Join usando ciclos anidados (fuerza bruta).

```
    {Orientado a tuplas}
1:  for cada tupla t de R do
2:      for cada tupla s de S do
3:          if t[A] = s[B] then
4:              Agregar tupla ⟨t, s⟩ al resultado
5:          end if
6:      end for
7:  end for
```

Sean n_r y n_s el número de tuplas de R y S respectivamente y b_r y b_s el número de bloques de R y S respectivamente.

Costo (accesos a bloques) $= n_r * b_s + b_r$.

EQUIJOIN (dos vías)

- J1 – Join usando ciclos anidados (fuerza bruta).

```
    {Orientado a bloques}
 1: for cada bloque B_r de R do
 2:    for cada bloque B_s de S do
 3:       for cada tupla t de B_r do
 4:          for cada tupla s de B_s do
 5:             if t[A] = s[B] then
 6:                Agregar tupla ⟨t, s⟩ al resultado
 7:             end if
 8:          end for
 9:       end for
10:    end for
11: end for
```

Costo (accesos a bloques) $= b_r * b_s + b_r$.

Algoritmos para la operación de JOIN

EQUIJOIN (dos vías)

- J2 – Join con ciclo simple (utilizando índice para obtener las tuplas que cumplen la condición). Si existe un índice para uno de los atributos del Join (ejemplo B de S), obtiene los registros de R, uno a la vez (ciclo simple) y luego utiliza el índice para obtener todas las tuplas s de S que cumplen $s[B] = t[A]$.

Algoritmos para la operación de JOIN

- **J3 – Join usando ordenación-mezcla.** R y S se encuentran ordenados por A y B respectivamente.

p_r = dirección de la primera tuplade r
p_s = dirección de la primera tupla de s { r y s intancias de R y S respectivamente}
while $p_s \neq null$ and $p_r \neq null$ **do**
  t_s = tupla que apunta p_s; $S_s = \{t_s\}$; hacer que p_s apunte a la siguiente tupla de s; $hecho = false$
  while not $hecho$ and $p_s \neq null$ **do**
    t'_s = tupla a la que apunta p_s
    if $t'_s[B] = t_s[B]$ **then**
      $S_s = S_s \cup \{t'_s\}$;
      hacer que p_s apunte a la siguiente tupla de s
    else
      $hecho = true$
    end if
  end while
  t_r = tupla que apunta p_r;
  while $p_r \neq null$ and $t_r[A] < t_s[B]$ **do**
    hacer que p_r apunte a la siguiente tupla de r
    t_r = tupla a la que apunta p_r
  end while
  while $p_r \neq null$ and $t_r[A] = t_s[B]$ **do**
    por cada tupla $t_s \in S_s$ Agregar $t_s \bowtie t_r$ al resultado
    hacer que p_r apunte a la siguiente tupla de r;
    t_r = tupla a la que apunta p_r
  end while

end while

Costo (accesos a bloques) = $b_r + b_s$.

Algoritmos para la operación de JOIN

Ejecute el algoritmo *J3* considerando las siguientes relaciones.

r

a1	a2
a	1
a	4
b	2
c	3
d	5
d	4

s

a1	a3
a	8
a	9
a	10
b	11
b	12
c	13
c	14
d	15

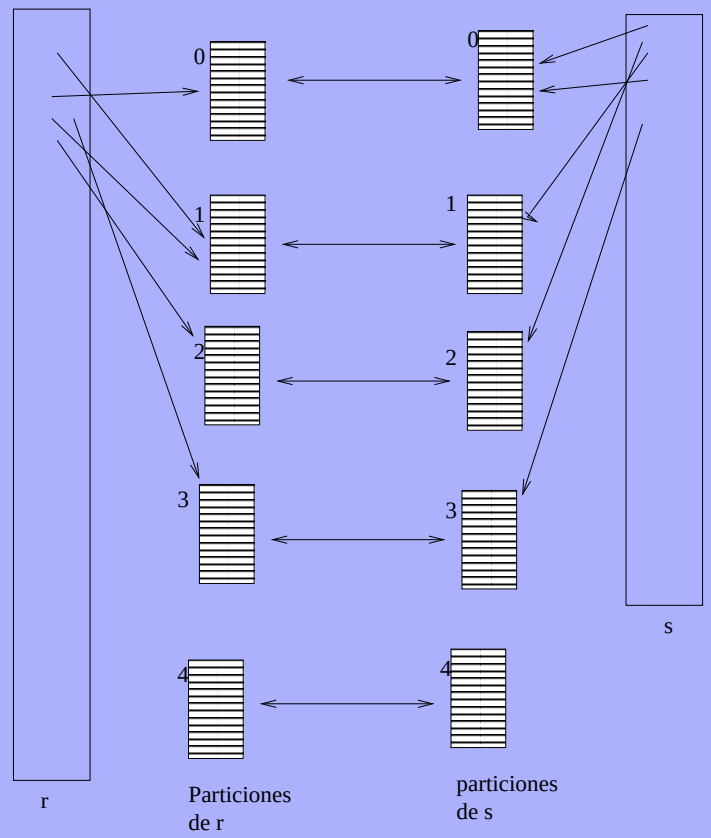
EQUIJOIN (dos vías)

- **J4 – Concatenación por asociación (Hash).**

1: **for** cada tupla t_s de s **do**
2: $i = h(B)$ {$h(.)$ función de hashing que asigna a B los valores $\{0, 1, \ldots, n_h\}$ }
3: $H_{s_i} = H_{s_i} \cup \{t_s\}$
4: **end for**
5: **for** cada tupla t_r de r **do**
6: $i = h(A)$ {$h(.)$ función de hashing que asigna a A los valores $\{0, 1, \ldots, n_h\}$ }
7: $H_{r_i} = H_{r_i} \cup \{t_r\}$
8: **end for**
9: **for** $i = 0$ to n_h **do**
10: leer H_{s_i} y construir un índice asociativo (hashing) en memoria Ih_{s_i}
11: **for** cada tupla t_r de H_{r_i} **do**
12: Explorar Ih_s para localizar las tuplas t_s tales que $t_s[B] = t_r[A]$
13: **for** cada tupla t_s que concuerde en H_{s_i} **do**
14: añadir $t_s \bowtie t_r$ al resultado
15: **end for**
16: **end for**
17: **end for**

EQUIJOIN (dos vías)

- **J4 – Concatenación por asociación (Hash).**



Costo (accesos a bloques) $= 3(b_r + b_s) + b_{res}$.

Algoritmos para la operación de Proyección

$T \leftarrow \pi_{\langle lista\ \ de\ \ atributos \rangle}(R)$

Crear una tupla $t[\langle$ lista de atributos $\rangle]$ en T' por cada tupla t_r de r
if $\langle$ lista de atributos $\rangle$ incluye una clave de R **then**
  $T = T'$
else
  Ordenar las tuplas de T'
  $i = 1$ {Eliminar duplicados}
  $j = 2$
  while $i \leq n_r$ **do**
    Agregar la tupla t'_i de T' a T
    while $(t'_i = t'_j$ con $t'_i, t'_j \in T')$ and $(j \leq n)$ **do**
      $j = j + 1$
    end while
    $i = j$
    $j = i + 1$
  end while
end if

Algoritmos para la operación de UNION

$T \leftarrow R \cup S$

Ordenar las tuplas de r y s por los mismos atributos únicos
$i = 1$
$j = 1$
while $i \leq n_r$ and $j \leq n_s$ **do**
  if $t_{r_i} > t_{s_j}$ **then**
    Agregar la tupla t_{s_j} a T
    $j = j + 1$
  else if $t_{r_i} < t_{s_j}$ **then**
    Agregar la tupla t_{r_i} a T
    $i = i + 1$
  else
    $j = j + 1$
  end if
end while
if $i \leq n_r$ **then**
  Agregar todas las tuplas t_{r_i} restantes a T
end if
if $j \leq n_s$ **then**
  Agregar todas la tuplas t_{s_j} restantes a T
end if

Algoritmos para la operación de interseccion

$T \leftarrow R \cap S$

Ordenar las tuplas de r y s por los mismos atributos únicos
$i = 1$
$j = 1$
while $i \leq n_r$ and $j \leq n_s$ **do**
  if $t_{r_i} > t_{s_j}$ **then**
    $j = j + 1$
  else if $t_{r_i} < t_{s_j}$ **then**
    $i = i + 1$
  else
    Agregar la tupla t_{s_j} a T
    $j = j + 1$
    $i = i + 1$
  end if
end while

Algoritmos para la operación de diferencia

$T \leftarrow R - S$

Ordenar las tuplas de r y s por los mismos atributos únicos
$i = 1$
$j = 1$
while $i \leq n_r$ and $j \leq n_s$ **do**
  if $t_{r_i} > t_{s_j}$ **then**
    $j = j + 1$
  else if $t_{r_i} < t_{s_j}$ **then**
    Agregar la tupla t_{r_i} a T
    $i = i + 1$
  else
    $j = j + 1$
    $i = i + 1$
  end if
end while
if $i \leq n_r$ **then**
  Agregar las tuplas desde t_{r_i} a $t_{r_{n_r}}$ a T
end if

Esquema de una base de datos relacional

EMPLEADO

Nombre	Apellido1	Apellido2	Dni	FechaNac	Direccion	Sexo	Sueldo	SuperDni	Dno

DEPARTAMENTO

NombreDpto	NumeroDpto	DniDirector	FechaIngresoDirector

LOCALIZACIONES_DPTO

NumeroDpto	UbicacionDpto

PROYECTO

NombreProyecto	NumProyecto	UbicacionProyecto	NumDptoProyecto

TRABAJA_EN

DniEmpleado	NumProy	Horas

DEPENDIENTE

DniEmpleado	NombreSubordinado	Sexo	FechaNac	Relacion

Heurística

Aplicar las operaciones de selección (σ) y proyección (π) antes de aplicar las operaciones de JOIN u otras operaciones binarias, ya que el tamaño de la relación (archivo) resultante de una operación binaria como JOIN, es por lo general, una función multiplicativa de los tamaños de las relaciones (archivos) de entrada.

Árbol de consulta

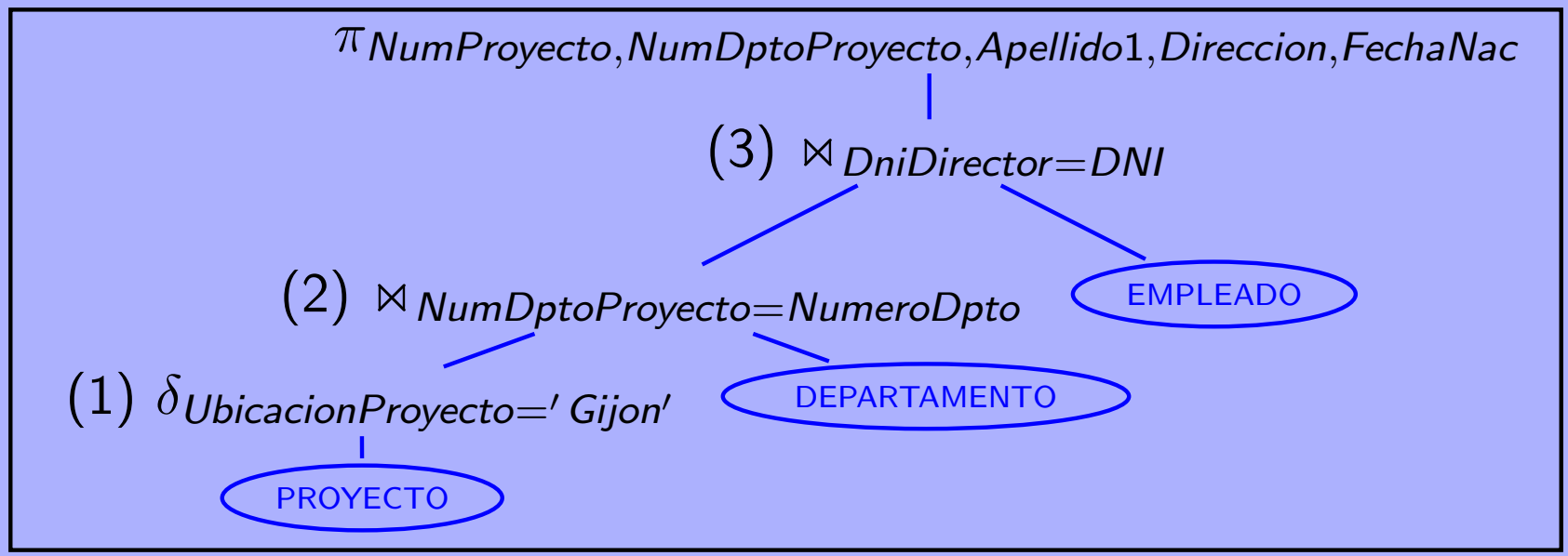
Q: "Para cada proyecto de 'Gijon', obtener el número de proyecto, el número de departamento que lo controla y el apellido del responsable del departamento, su dirección y su fecha de nacimiento"

SELECT P.NumProyecto, P.NumDeptoProyecto, E.Apellido1, E.Direccion, E.FechaNac
FROM PROYECTO AS P; DEPARTAMENTO AS D, EMPLEADO AS E
WHERE P.NumDeptoProyecto = D.NumeroDepto AND D.DniDirector = E.Dni AND p.UbicacionProyecto = 'Gijon'

$$\pi_{NumProyecto,NumDptoProyecto,Apellido1,Direccion,FechaNac}(((\delta_{UbicacionProyecto='Gijon'}(PROYECTO)) \bowtie_{NumDptoProyecto=NumeroDpto} (DEPARTAMENTO)) \bowtie_{DniDirector=DNI} (EMPLEADO))$$

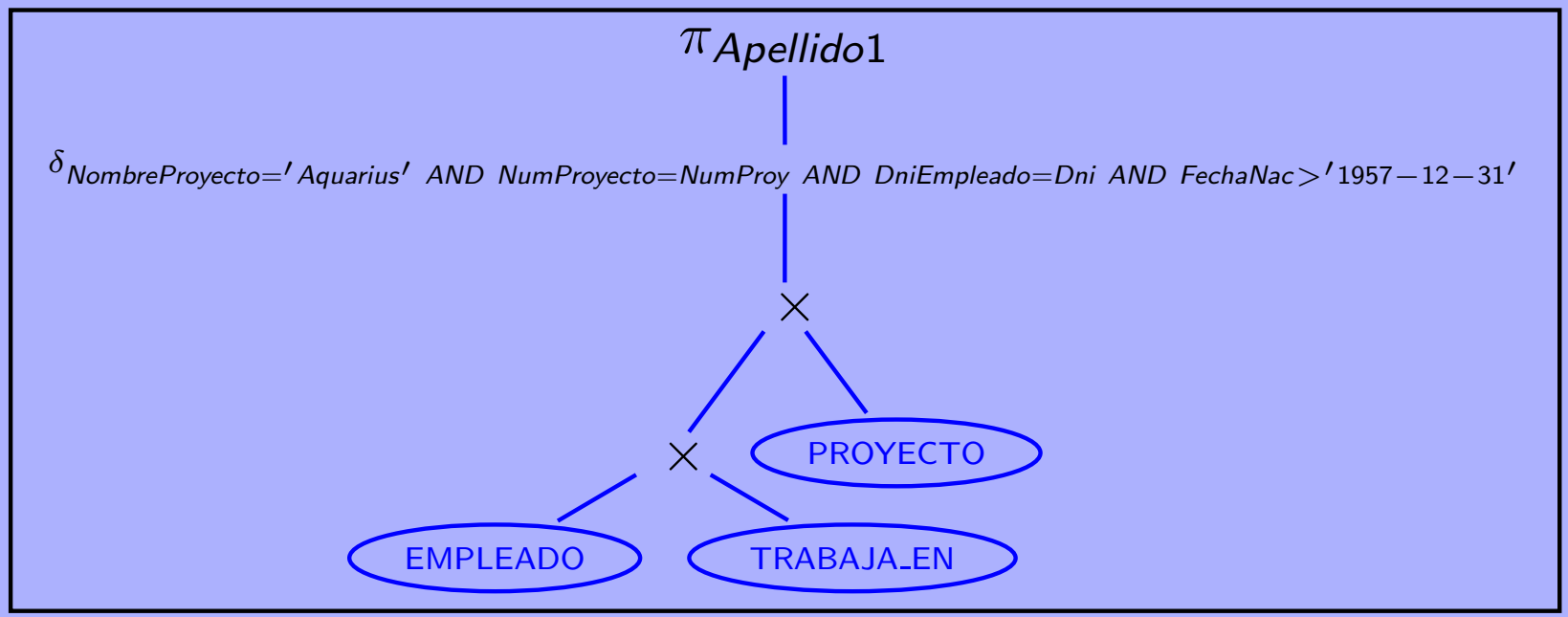
Árbol de consulta (Notación)

$\pi_{NumProyecto,NumDptoProyecto,Apellido1,Direccion,FechaNac}$
$(((\delta_{UbicacionProyecto='Gijon'}(PROYECTO))$
$\bowtie_{NumDptoProyecto=NumeroDpto}(DEPARTAMENTO)) \bowtie_{DniDirector=DNI}$
$(EMPLEADO))$



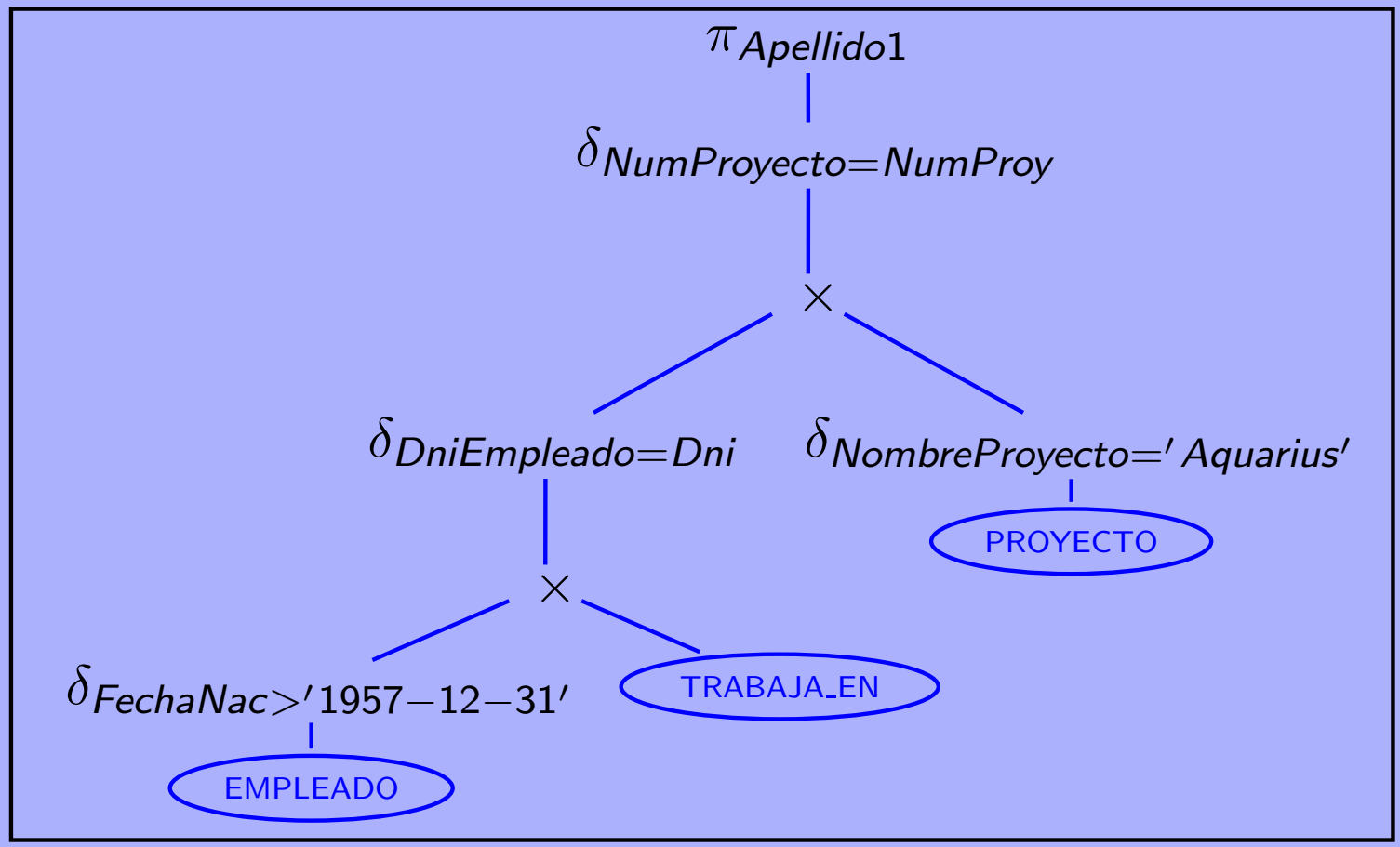
Optimización (Arbol canónico)

SELECT Apellido1
FROM EMPLEADO, TRABAJA_EN, PROYECTO
WHERE NombreProyecto = 'Aquarius' AND
NumProyecto=NumProy AND DniEmpleado = Dni AND FechaNac > '1957-12-31'



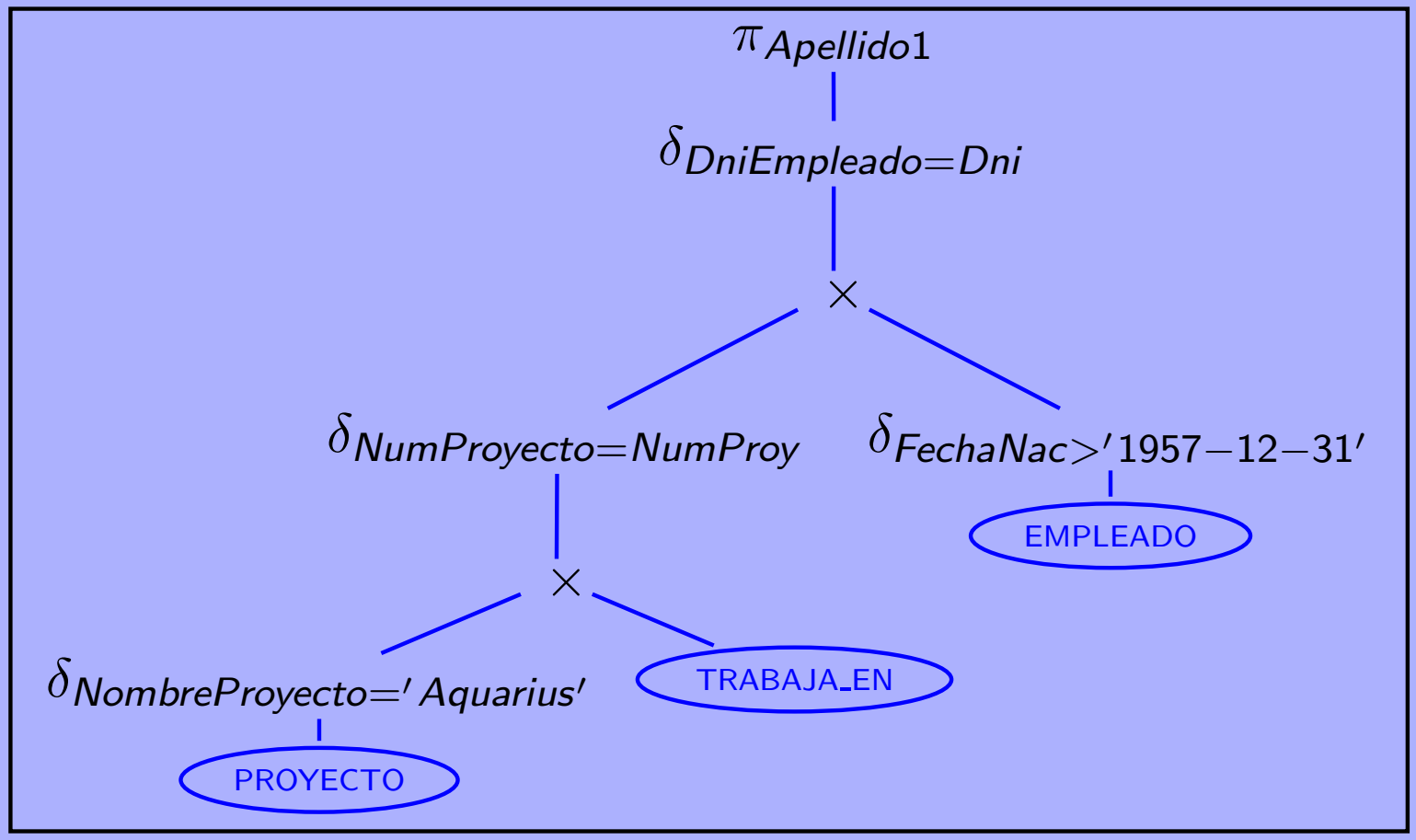
Optimización

Después de aplicar los pasos 1 y 2 al árbol anterior



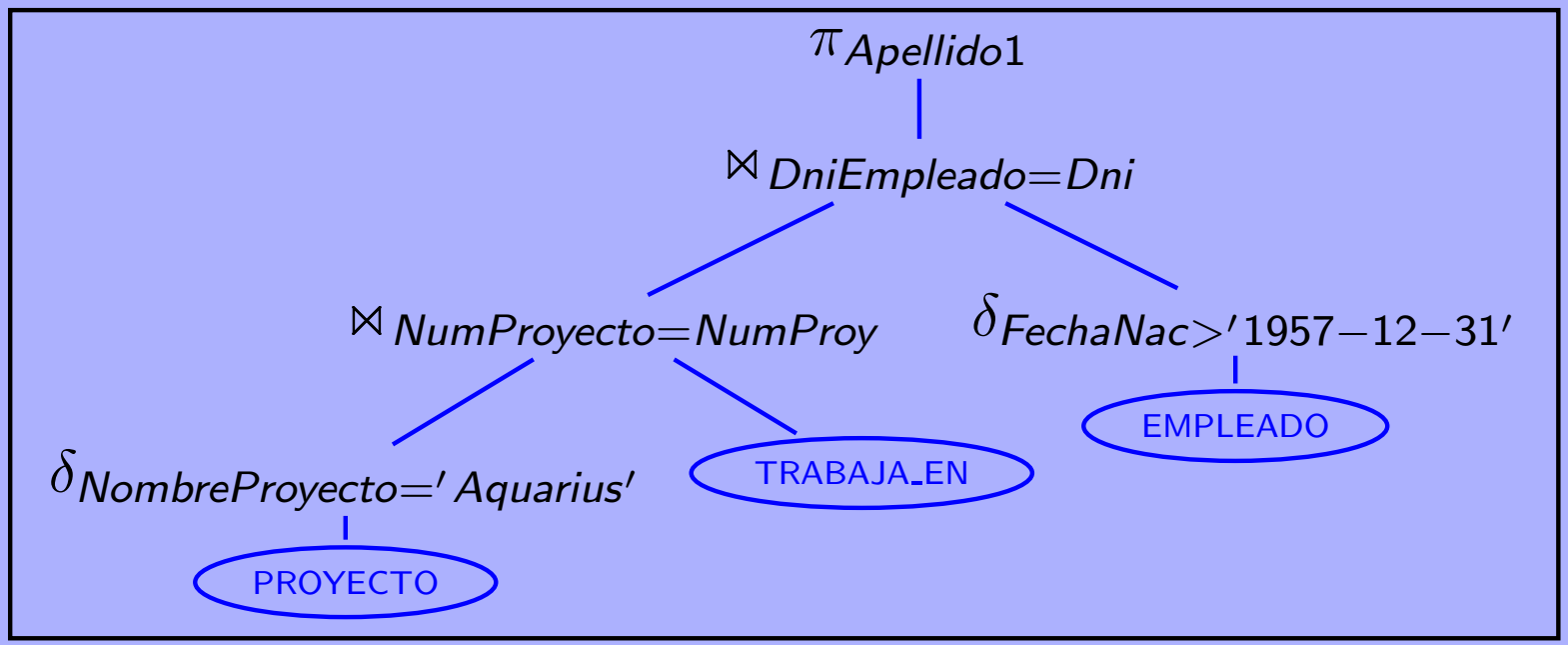
Optimización

Después de aplicar paso 3 al árbol anterior



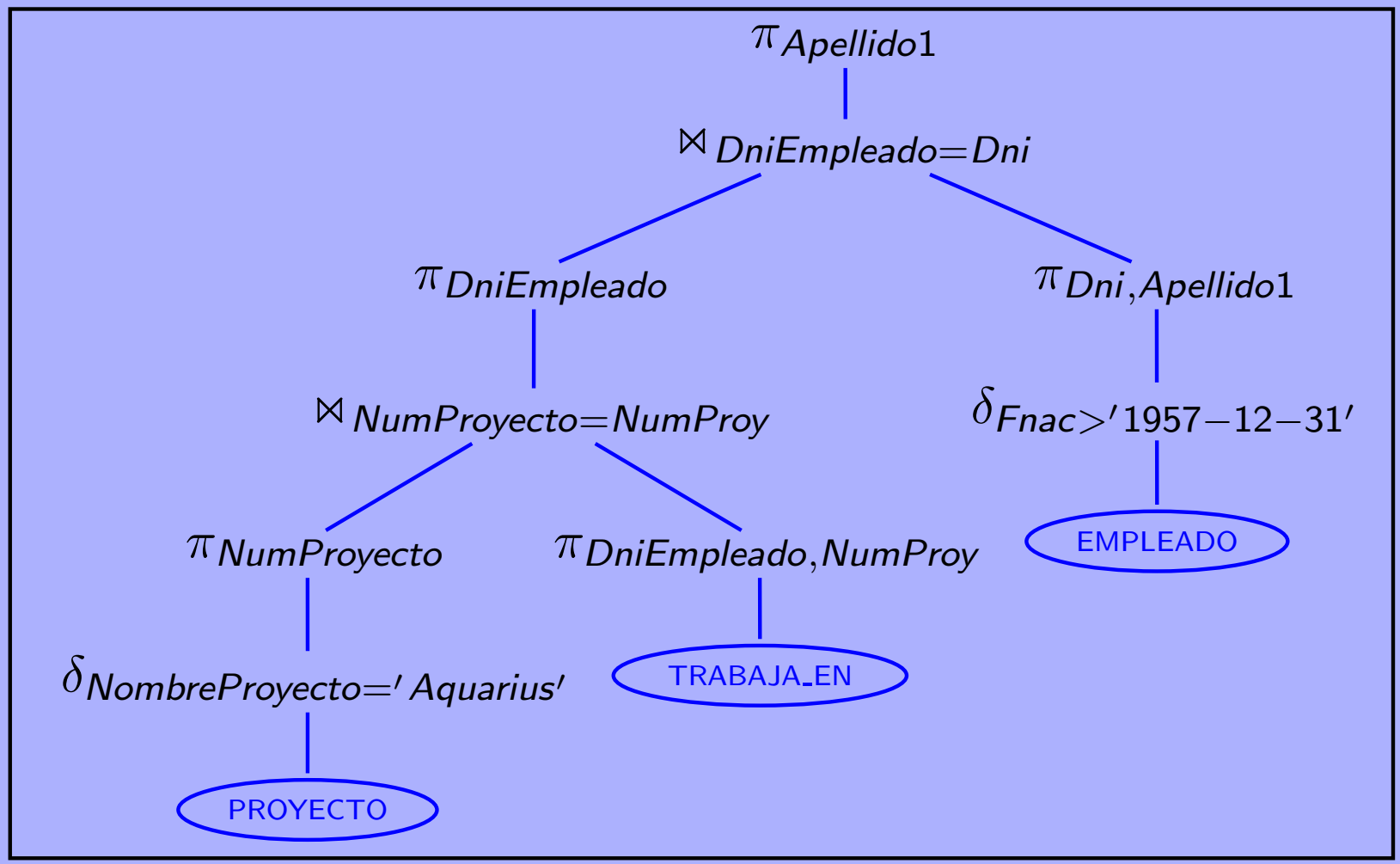
Optimización

Después de aplicar el paso 4 al árbol anterior



Optimización

Después de aplicar el paso 5 al árbol anterior



Heurísticas para la optimización de consultas

Reglas de equivalencia del álgebra relacional

1. Cascada de δ.
   $\delta_{c1\ AND\ c2\ AND...AND\ cn}(R) \equiv \delta_{c1}(\delta_{c2}(\ldots(\delta_{c}n(R))\ldots))$
2. Conmutatividad de δ. $\delta_{c1}(\delta_{c2}(R)) \equiv \delta_{c2}(\delta_{c1}(R))$
3. Cascada de π. $\pi_{Lista1}(\pi_{Lista2}(\ldots(\pi_{Listan}(R))\ldots)) \equiv \pi_{Lista1}(R)$
4. Conmutación de δ por π. Si la condición c incluye sólo a los atributos $A_1, \ldots, A_n$ en la lista de proyección.
   $\pi_{A_1,A_2,...,A_n}(\delta_c(R)) \equiv \delta_c(\pi_{A_1,A_2,...,A_n}(R))$
5. Commutatividad de $\bowtie$ y $\times$.
   $R \bowtie_c S \equiv S \bowtie_c R$
   $R \times S \equiv S \times R$
6. Conmutación de δ con $\bowtie$ (o $\times$).
   $\delta_c(R \bowtie S) \equiv (\delta_c(R)) \bowtie S$. Se asume que c se define solo con atributos de R
   $\delta_c(R \bowtie S) \equiv (\delta_{c1}(R)) \bowtie (\delta_{c2}(S))$ con $c = c1\ AND\ c2$

Reglas de equivalencia del álgebra relacional (cont)

7. Conmutación de π con $\bowtie$ (o $\times$). Sea $L = \{A_1, \ldots, A_2, B_1, \ldots, B_m\}$ donde $A_1, \ldots, A_2 \in R$ y $B_1, \ldots, B_m \in S$. Si la operación de join c incluye los atributos de L.
   $\pi_L(R \bowtie_c S) \equiv (\pi_{A_1,\ldots,A_n}(R)) \bowtie_c (\pi_{B_1,\ldots,B_m}(S))$. Si c contiene atributos que no están en L estos deben ser agregados a la lista de proyección y se necesita hacer una operación de proyeccción adicional.
   $\pi_L(R \bowtie_c S) \equiv \pi_L((\pi_{A_1,\ldots,A_n,A_{n+1},\ldots,A_{n+k}}(R)) \bowtie_c (\pi_{B_1,\ldots,B_m,B_{m+1},\ldots,B_{m+p}}(S)))$.
8. Conmutatividad de operaciones de conjuntos. Las operaciones de $\cap$ y $\cup$ son conmutativos pero la diferencia $(-)$ no lo es.
9. Asociatividad de $\bowtie, \times, \cap$ y $\cup$. Si θ representa a cualquiera de estas operaciones:
   $(R\theta S)\theta T \equiv R\theta(S\theta T)$

Reglas de equivalencia del álgebra relacional (cont)

10. Conmutación de σ con la operación de conjuntos. La operación σ se puede conmutar con $\cap, \cup$ y $-$. Si θ representa cualquiera de estas tres operaciones tendremos
$\sigma_c(R\ \theta\ S) \equiv (\sigma_c(R))\ \theta\ (\sigma_c(S))$
11. La operación π se puede conmutar con $\cup$.
$\pi_L(R \cup S) \equiv (\pi_L(R)) \cup (\pi_L(S))$
12. Conversión de una secuencia $(\sigma, \times)$ en $\bowtie$. Si la condición c de una operación σ que sigue a una operación $\times$ corresponde a una operación Join convierte la secuencia $(\sigma, \times)$ en una $\bowtie$ de este modo:
$(\sigma_c(R \times S)) \equiv (R \bowtie_c S)$

Algoritmo de optimización algebraica heurística

1. Utilizando regla 1, descomponer cualquier operación SELECT con condiciones conjuntivas en una cascada de operaciones SELECT.
2. Utilizando reglas 2, 4, 6 y 10 sobre la conmutatividad de SELECT con otras operaciones desplazar cada SELECT hacia abajo en el árbol tan lejos como lo permitan los atributos incluidos en la condición de selección.
3. Utilizando reglas 5 y 9 relativas a la conmutatividad y asociatividad de las operaciones binarias reordenar las relaciones de los nodos hojas de utilizando los siguientes criterios:
   - En primer lugar, posicionar las relaciones de los nodos hojas con las operaciones SELECT mas restrictivas de modo que sean ejecutadas en primer lugar en la representación del árbol.
   - En segundo lugar, asegurarse de que la ordenación de lo nodos hojas no produce $\times$. Por ejemplo, si las dos relacionescon SELECT más restriictiva no tienen una condición de join directa entre ellas, puede ser conveniente cambiar el orden de los nodos hoja para evitar productos cartesianos.

Algoritmo de optimización algebraica heurística (cont.)

4. Utilizando regla 12, combinar una operación $\times$ con la siguiente operación SELECT del árbol para formar una operación Join, en el caso de que la condición represente una condición de Join.
5. Utilizando las reglas 3, 4, 7 y 11, descomponer y desplazar las listas de atributos de proyección hacia abjo por el árbol lo más lejos posible mediante la creación de nuevas operaciones de proyección según sea necesario.
6. Identificar los subárboles que representan grupos de operaciones que se pueden ejecutar mediante un único algoritmo.

Conversión de árboles de consultas en planes de ejecución de consultas

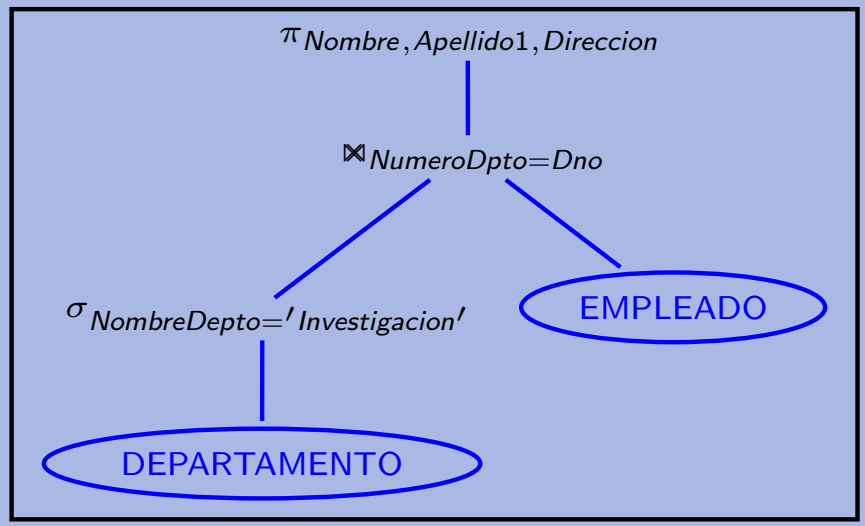
Plan de ejecución de consulta

Sea Q una consulta representada en un árbol de consulta. Un plan para Q incluye información:

- Métodos de acceso disponibles para las relaciones
- Algoritmos a utilizar para calcular los operadores relacionales

Example

$\pi_{Nombre, Apellido1, Direccion}(\sigma_{NombreDepto='Investigacion'}(DEPARTAMENTO) \bowtie_{NumeroDpto=Dno} EMPLEADO)$



Para generar el plan de ejecución el optimizador debería tener en cuenta:

- índice de EMPLEADO y DEPARTAMENTO
- Algoritmo para ejecutar JOIN
- Evaluación materializada[a] o en secuencia[b]

---

[a] el resultado se almacena como relación temporal

[b] a medida que se generan las tuplas en una operación, estas se pasan a otra operación

Estimación de costo de una consulta

Componentes del costo de ejecución de una consulta

1. Costo de acceso al almacenamiento secundario
2. Costo de almacenamiento (archivos/tablas intermedias)
3. Costo computacional (procesamiento en memoria principal)
4. Memoria principal (buffers)
5. Costo de comunicaciones

Estimación de costos de una consulta

Escenarios

- En bases de datos grandes, interesa minimizar el costo de acceso a almacenamiento secundario
- En bases de datos pequeñas (que caben en memoria) Interesa minimizar el costo computacional
- En bases de datos distribuidas también interesa el costo de comunicación

Usaremos funciones de costo que tienen en cuenta solamente el acceso a disco

Estimación de costo de una consulta

Información del catálogo utilizada en funciones de costo

1. Número de tuplas/registros (r) por cada relación/tabla/archivo
2. Tamaño de la tupla (R) de cada relación
3. Número de bloques(b)
4. Factor de bloque (bfr)
5. Método de acceso
   - Primario/secundario
   - Atributos
   - Número de niveles (x)
   - Número de bloques (nodos) internos de un índice (b_{I1})
6. Número de valores distintos (d) de un atributo y su selectividad (sl)
7. Cardinalidad de selección ($s = sl * r$)

Estimación de costo de una consulta

Funciones de costos para σ

1. S1 - Búsqueda lineal. $C_{S1_a} = b$ en el peor caso y $C_{S1_b} = \frac{b}{2}$ en promedio
2. S2 - Búsqueda binaria. $C_{S2} = \log_2 b + \left\lceil \frac{s}{brf} \right\rceil - 1$. Si $s = 1$, $C_{S2} = \log_2 b$
3. S3 - Uso de índice primario para obtener una única tupla. $C_{S3} = x + 1$
4. S4 - Uso de índice de ordenación con predicados $<, \leq, >, \geq$. $C_{S4} = x + \frac{b}{2}$. Cálculo poco preciso.
5. S5 - Uso de índice de agrupación para obtener varios registros $C_{S5} = x + \left\lceil \frac{s}{brf} \right\rceil$, con s la cardinalidad de selección del atributo de indexación.

Funciones de costos para σ

6. S6 - Uso de índice secundario (árbol B^+, condición de igualdad). $C_{6a} = x + s$. Si la condición es $<, \leq, >, \geq$ y se supone que la mitad de las tuplas cumplen la condición, entonces (a grandes rasgos) se accede a la mitad de los bloques internos del índice y a la mitad de los bloques de datos. Es decir, $C_{6b} = x + (\frac{b_{I1}}{2} + \frac{r}{2})$.
7. S7 - Selección conjuntiva. Podemos utilizar S1 o uno de los métodos de S2 a S6 (usamos uno de las condiciones y luego verificamos en el buffer que se cumplan las restantes).
8. S8 - Selección conjuntiva usando un índice compuesto. Igual que S3a, S5, o S6a, dependiendo del tipo de índice.

Estimación de costo de una consulta

Funciones de costos para $\bowtie$

Selectividad de $\bowtie$, $js = \frac{|R \bowtie_c S|}{|R \times S|} = \frac{|R \bowtie_c S|}{|R| * |S|}$

- si $js = 1$, entonces $\bowtie$ es igual $\times$.
- si $js = 0$, entonces ninguna tupla cumple c
- En general $0 \leq js \leq 1$
- Si c es de la forma $R.A = S.B$
  1. Si A es clave de R, entonces $|R \bowtie_c S| \leq |S|$; por tanto, $js \leq \frac{1}{|R|}$
  2. Si B es clave de S, entonces $|R \bowtie_c S| \leq |R|$; por tanto, $js \leq \frac{1}{|S|}$
  3. Si A y B no son claves, entonces $|R \bowtie_c S| = \frac{|R| * |S|}{d_B}$, con d_B los distintos valores de B en S, $js \leq \frac{1}{d_B}$, ó $|R \bowtie_c S| = \frac{|R| * |S|}{d_A}$, con d_A los distintos valores de A en R, $js \leq \frac{1}{d_A}$.

Si disponemos de js podemos estimar el tamaño del resultado de $\bowtie$ como $|R \bowtie_c S| = js * |R| * |S|$

Estimación de costo de una consulta

Funciones de costos para $R \bowtie_c S$

1. J1 - Ciclo anidado. Asumimos R en el ciclo externo.
   $C_{J1} = b_r + (b_r * b_s) + ((js * |R| * |S|)/bfr_{RS})$[a]
2. J2 - Ciclo simple usando usando índice para obtener las tuplas que cumplen la condición c. Asumimos que B tiene un índice de profundidad (niveles) x_B
   1. Índice secundario, s_B cardinalidad de la selección para B
      $C_{J2a} = b_r + (|R| * (x_B + s_B)) + ((js * |R| * |S|)/bfr_{RS})$
   2. Índice agrupado, s_B cardinalidad de la selección para B
      $C_{J2b} = b_r + (|R| * (x_B + (s_B/bfr_B)) + ((js * |R| * |S|)/bfr_{RS})$
   3. Índice primario
      $C_{J2c} = b_r + (|R| * (x_B + 1)) + ((js * |R| * |S|)/bfr_{RS})$
   4. Hashing
      $C_{J2d} = b_r + (|R| * h)) + ((js * |R| * |S|)/bfr_{RS})$, con $h \geq 1$ la cantidad media de accesos a bloques para obtener una tupla

[a] Esta útlima expresión representa el costo (bloques) en escribir la respuesta

Estimación de costo de una consulta

Funciones de costos para $R \bowtie_c S$

3. J3 - Join usando Ordenación-Mezcla.
   $C_{J3} = b_r + b_s + ((js * |R| * |S|)/bfr_{RS})^a$

[a] Se asume que R está ordenado por el atributo A y S por B