

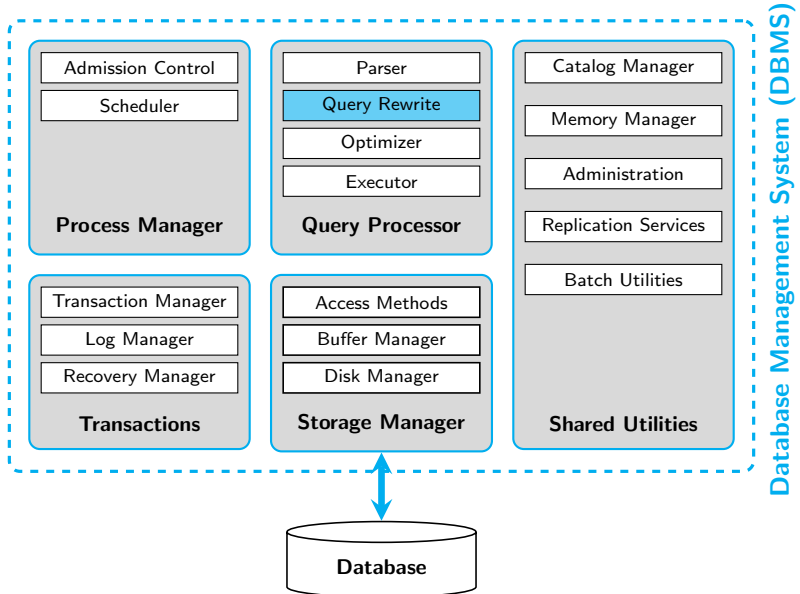
Re-escritura de consultas

Clase 15

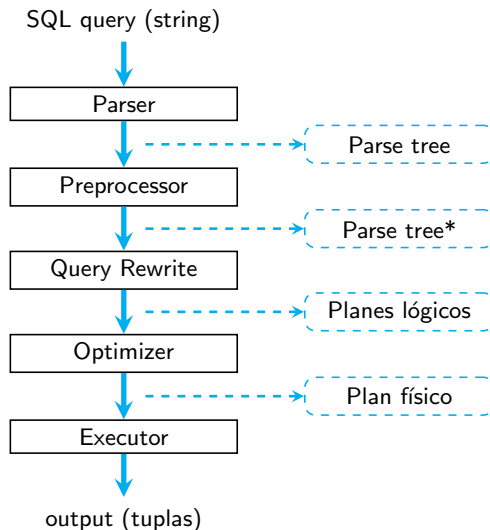
IIC 3413

Prof. Cristian Riveros

Re-escritura de consultas



Zoom al optimizador de consultas



Plan lógico

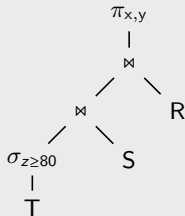
Definición

Un **plan lógico** es un árbol ordenado y etiquetado \mathcal{T} tal que:

- $\text{label}(h)$ es una relación,
- $\text{label}(n)$ es un operador de algebra relacional y
- $|\text{children}(n)| = \text{arity}(\text{label}(n))$

para todo nodo hoja $h \in \text{nodes}(\mathcal{T})$ y nodo interno $n \in \text{nodes}(\mathcal{T})$.

Ejemplo



Re-escritura de consultas

1. Convertir el árbol de parsing en un **plan lógico**.
2. Reescribe consulta aplicando **reglas** de algebra relacional.
3. Crea un set de planes lógicos **prometedores** para ser optimizados.

¿cómo convertimos un árbol de parsing en un plan lógico?

```
SELECT  <SelList>  
FROM    <FromList>  
WHERE   <Condition>
```

1. Combinamos las relaciones en el <FromList> con **productos cruz**:

$$R_1 \times \dots \times R_n$$

2. Aplicamos **selección** con la condición C dada en el <Condition>:

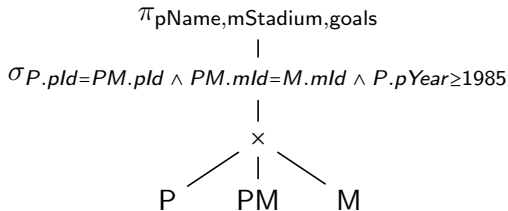
$$\sigma_{\langle \text{Condition} \rangle} (R_1 \times \dots \times R_n)$$

3. Aplicamos **proyección** con los atributos dados en el <SelList>:

$$\pi_{\langle \text{SelList} \rangle} (\sigma_{\langle \text{Condition} \rangle} (R_1 \times \dots \times R_n))$$

Ejemplo de árbol de parsing a plan lógico

```
SELECT  pName, mStadium, goals
FROM    Players AS P, Matches AS M, Players_Matches AS PM
WHERE   P.pld = PM.pld AND PM.mld = M.mld AND
        P.pYear ≥ 1985
```



Uso de vistas (no materializadas)

Por cada **vista** V en un plan lógico P :

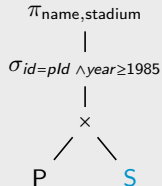
- usamos su definición para construir un plan lógico P' para V .
- colgamos P' en cada nodo que es mencionada en P .

Uso de vistas (no materializadas)

Ejemplo

```
SELECT name, stadium
FROM Players P, Stadiums S
WHERE id = pld AND
      year ≥ 1985
```

```
CREATE VIEW Stadiums AS
SELECT pld, stadium
FROM Matches
```

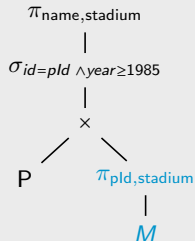


Uso de vistas (no materializadas)

Ejemplo

```
SELECT name, stadium
FROM Players P, Stadiums S
WHERE id = pld AND
      year ≥ 1985
```

```
CREATE VIEW Stadiums AS
SELECT pld, stadium
FROM Matches
```



Uso de consultas anidadas

Consultas anidada: consulta que contienen una subconsulta embebida en:

- FROM
- WHERE
- HAVING

Uso de consultas correlacionadas

Consulta correlacionada: consulta anidada que contienen una subconsulta embebida con una referencia a la consulta externa.

Ejemplo

```
SELECT  pName, pYear
FROM    Players AS P
WHERE   1 ≤ ( SELECT  AVG(mGoals)
              FROM    Matches AS M
              WHERE   M.mld = P.pld )
```

¿cómo desanidamos estas consultas?

(ver pizarra)

Re-escritura de consultas

1. Convertir el árbol de parsing en un **plan lógico**. ✓
2. Reescribe consulta aplicando **reglas** de algebra relacional.
3. Crea un set de planes lógicos **prometedores** para ser optimizados.

Reglas de reescritura de consultas

Definición

Dos consultas Q_1 y Q_2 en algebra relacional son **equivalentes** ($Q_1 = Q_2$) ssi:

para toda instancia D , se tiene que: $Q_1(D) = Q_2(D)$.

Reescritura de consultas

Dado una consulta Q , buscar una consuta Q' tal que:

1. $Q = Q'$ y
2. evaluar Q' con un plan físico sea **más eficiente**.

¿cómo encontramos Q' ?

Reglas de reescritura de consultas

Usando reglas muy sencillas de algebra relacional:

1. Conmutatividad.
2. Asociatividad.
3. Distributividad.
4. Distribución de selección (push-selection).
5. Distribución de proyección (push-projection).
6. Simplificación de productos.
7. Eliminación de duplicados.

Una de las **grandes ventajas** de algebra relacional!

Conmutatividad

Las siguientes consultas son **equivalentes**:

$$R \times S = S \times R$$

$$R \bowtie S = S \bowtie R$$

$$R \cap S = S \cap R$$

$$R \cup S = S \cup R$$

Estas reglas son validas tanto para **set-** como **bag-semantics**.

- ¿es p -join conmutativo?
- ¿cuál es la importancia de estas reglas?

Asociatividad

Las siguientes consultas son **equivalentes**:

$$(R \times S) \times T = R \times (S \times T)$$

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

$$(R \cap S) \cap T = R \cap (S \cap T)$$

$$(R \cup S) \cup T = R \cup (S \cup T)$$

Estas reglas son validas tanto para **set-** como **bag-semantics**.

- ¿es p -join asociativo?
- ¿cuál es la importancia de estas reglas?

Distributividad

Las siguientes consultas son **equivalentes**:

$$R \cap_S (S \cup_S T) = (R \cap_S S) \cup_S (R \cap_S T)$$

¿qué ocurre para **bag-semantics**?

Selección: AND y OR

Las siguientes consultas son **equivalentes**:

$$\sigma_{p_1 \text{ AND } p_2}(R) = \sigma_{p_1}(\sigma_{p_2}(R)) = \sigma_{p_2}(\sigma_{p_1}(R))$$

$$\sigma_{p_1 \text{ OR } p_2}(R) = \sigma_{p_1}(R) \cup_S \sigma_{p_2}(R)$$

¿cuál es la utilidad de esta regla?

Selección: push-selection

Las siguientes consultas son **equivalentes**:

$$\sigma_p(R \times S) = \sigma_p(R) \times S$$

$$\sigma_p(R \bowtie S) = \sigma_p(R) \bowtie S$$

$$\sigma_p(R \bowtie_q S) = \sigma_p(R) \bowtie_q S$$

$$\sigma_p(R \cap S) = \sigma_p(R) \cap S$$

si R tiene todos los atributos mencionados en p .

Selección: push-selection

Análogamente:

$$\sigma_p(R \times S) = R \times \sigma_p(S)$$

$$\sigma_p(R \bowtie S) = R \bowtie \sigma_p(S)$$

$$\sigma_p(R \bowtie_q S) = R \bowtie_q \sigma_p(S)$$

$$\sigma_p(R \cap S) = R \cap \sigma_p(S)$$

si S tiene todos los atributos mencionados en p .

Selección: push-selection

Si R y S tiene todos los atributos mencionados en p , simultáneamente:

$$\sigma_p(R \bowtie S) = \sigma_p(R) \bowtie \sigma_p(S)$$

$$\sigma_p(R \cap S) = \sigma_p(R) \cap \sigma_p(S)$$

Las siguientes consultas son siempre **equivalentes**:

$$\sigma_p(R \cup S) = \sigma_p(R) \cup \sigma_p(S)$$

$$\sigma_p(R - S) = \sigma_p(R) - \sigma_p(S) = \sigma_p(R) - S$$

Ejemplo

Considere las relaciones $R(a, b)$ y $S(b, c)$, transformar:

$$\sigma_{(a=1 \text{ OR } a=3) \text{ AND } b < c}(R \bowtie S)$$

Selección: push-selection

- Una de las reglas mas importante de los optimizadores!
- ¿es siempre útil hacer push-selection?
- ¿puede ser útil hacer “pop-selection”?

Proyección: push-projection

“Haremos push-projection siempre y cuando solo eliminamos atributos que no son usados por los ancestros de ese nodo”

Proyección: push-projection

Las siguientes consultas son **equivalentes**:

$$\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(S))$$

$$\pi_L(R \bowtie_p S) = \pi_L(\pi_M(R) \bowtie_p \pi_N(S))$$

$$\pi_L(R \times S) = \pi_L(\pi_M(R) \times \pi_N(S))$$

$$\pi_L(R \cup_B S) = \pi_L(R) \cup_B \pi_L(S)$$

$$\pi_L(\sigma_p(R)) = \pi_L(\sigma_p(\pi_M(R)))$$

si M y N no “**interfieren**” con el join ni la proyección de L .

Ejemplo

Considere las relaciones $R(a, b, c)$ y $S(b, c, d)$, transformar:

$$\pi_{a,b}(R \bowtie S)$$

Joins y productos

Las siguientes consultas son **equivalentes**:

$$R \bowtie_p S = \sigma_p(R \times S)$$

$$R \bowtie S = \pi_L(\sigma_q(R \times S))$$

donde:

$$q := \bigwedge_{x \in \text{attrib}(R) \cap \text{attrib}(S)} R.x = S.x$$

$$L := \text{attrib}(R) \cup_S \text{attrib}(S).$$

¿cuál es la utilidad de esta regla?

Eliminación de duplicados

Las siguientes consultas son **equivalentes**:

$$\delta(R \times S) = \delta(R) \times \delta(S)$$

$$\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$$

$$\delta(R \bowtie_p S) = \delta(R) \bowtie_p \delta(S)$$

$$\delta(\sigma_p(R)) = \sigma_p(\delta(R))$$

Siempre considerar que $\delta(R) = R$ si R NO tiene duplicados como:

- R tiene una llave primaria.
- R es el resultado de un group-by.
- R es el resultado de operadores con set-semantics.

GroupBy

- Reglas dependen mucho de las funciones de agregación.
- Algunas reglas:

$$\delta(\gamma_L(R)) = \gamma_L(R)$$

$$\gamma_L(\delta(R)) = \gamma_L(R) \quad (*)$$

(*) si L considera funciones de agregación como MAX o MIN.

Re-escritura de consultas

1. Convertir el árbol de parsing en un **plan lógico**. ✓
2. Reescribe consulta aplicando **reglas** de algebra relacional. ✓
3. Crea un set de planes lógicos **prometedores** para ser optimizados.

¿qué reglas usar?

No existe una regla única!

Uso de heurísticas:

- hacer **push** de las **selecciones** lo mas abajo posible.
- hacer **push** de las **proyecciones** lo mas abajo posible.
- proyectar y eliminar atributos que no se usarán.
- convertir los productos en **equi-joins**.
- elegir el **mejor orden*** de joins, uniones o intersecciones.
- remover eliminación de duplicados.

¿qué reglas usar?

Ejemplo



¿cómo decidir cuales planes lógicos son **prometedores**?

En base a (posibles criterios):

- tamaño de las relaciones intermedias.
- uso de índices.
- posibles sorting de los resultados.