

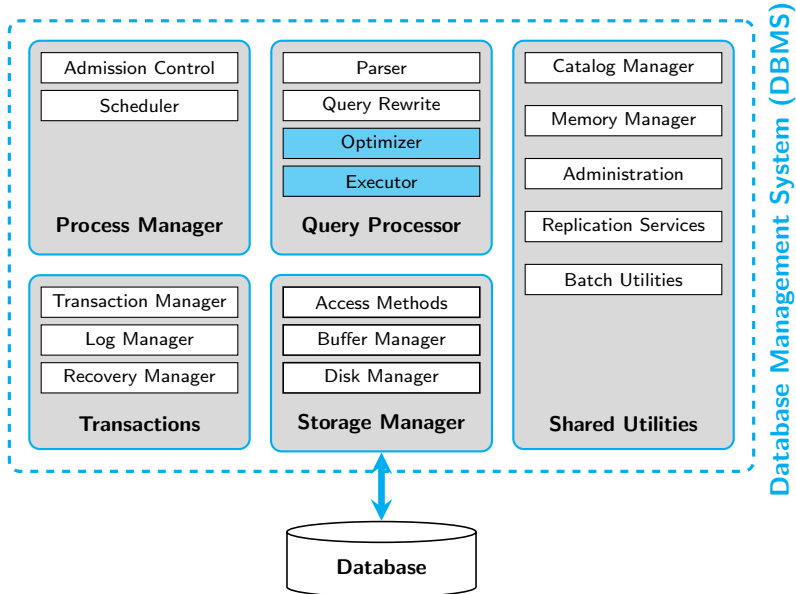
# Implementación de operadores relacionales

Clase 12

IIC 3413

Prof. Cristian Riveros

# Implementación de operadores relacionales



# Implementación de operadores y sus variantes

## Operador físico:

- Cada operador físico implementa un operador relacional (lógico).
- Implementado para desempeñarse bien en una tarea específica.

Cada **variante** aprovecha propiedades físicas de los datos:

- Presencia o ausencia de índices.
- Orden del input.
- Tamaño del input.
- Cantidad de elementos distintos.
- Espacio disponible en memoria.

# Técnicas de operadores físicos

Los algoritmos de **operadores físicos** se pueden dividir en:

1. Iteración.
2. Hashing.
3. Sorting.
4. Índices.
5. Otros.

# Outline

Parámetros de costo

Selección

Proyección

Duplicados

Group by

Unión

# Outline

**Parámetros de costo**

Selección

Proyección

Duplicados

Group by

Unión

# Parámetros para medir costo

## Definición

Durante esta clase denotamos con  $R$  o  $S$  una “consulta” relacional, o sea:

- una relación, o
- el resultado de una consulta.

## Ejemplo

$R \quad := \quad \text{relación Players.}$

$S \quad := \quad \text{consulta } \pi_{Id}(\text{Matches}).$

# Parámetros para medir costo

**Parámetros** de interés:

$\text{cost}(R)$ :	costo (en I/O) para computar $R$ .
$\text{pages}(R)$ :	cantidad de páginas necesarias para almacenar $R$ .
$ R $ :	cantidad de tuplas/records en $R$ .
$\text{rsize}(R)$ :	tamaño de una tupla/record (promedio) en $R$ .
$\text{distinct}(R)$ :	cantidad de elementos distintos en $R$ .
$\text{distinct}_a(R)$ :	cantidad de elementos distintos en el campo $R.a$ .
$ \text{page} $ :	tamaño/espacio de una página <sup>*</sup> .

Estos parámetros debemos estimarlos.



# Parámetros para medir costo

Otro parámetro importante: **selectividad**.

$\text{sel}_p(R)$ : fracción de tuplas/records en  $R$  que satisfacen  $p$ .

donde  $p$  es una combinación booleana ( $\wedge, \vee$ ) de términos:

$$\begin{array}{lcl} \textit{atributo}_1 & \text{op} & \textit{atributo}_2 \\ \textit{atributo} & \text{op} & \textit{constante} \end{array}$$

con  $\text{op} \in \{=, \leq, \geq, <, >\}$ .

Definición

$$0 \leq \text{sel}_p(R) = \frac{|\sigma_p(R)|}{|R|} \leq 1$$

Durante esta clase,  
calcularemos los siguientes parámetros...

$\text{cost}(R)$ : costo (en I/O) para computar  $R$ .  
 $\text{pages}(R)$ : número de páginas necesarias para almacenar  $R$ .  
 $|R|$ : número de tuplas/records en  $R$ .  
 $\text{rsize}(R)$ : tamaño de una tupla/record (promedio) en  $R$ .

## Ejemplo

Los parámetros de **sorting** de  $R$  (o  $\tau(R)$ ) son:

$$\begin{aligned}\text{cost}(\tau(R)) &= \text{cost}(R) + 2 \cdot \text{pages}(R) \\ \text{pages}(\tau(R)) &= \text{pages}(R) \\ |\tau(R)| &= |R| \\ \text{rsize}(\tau(R)) &= \text{rsize}(R)\end{aligned}$$

Durante esta clase,  
calcularemos los siguientes parámetros...

$\text{cost}(R)$ : costo (en I/O) para computar  $R$ .  
 $\text{pages}(R)$ : número de páginas necesarias para almacenar  $R$ .  
 $|R|$ : número de tuplas/records en  $R$ .  
 $\text{rsize}(R)$ : tamaño de una tupla/record (promedio) en  $R$ .

Los siguientes parámetros:

$\text{distinct}(R)$ : cantidad de elementos distintos en  $R$ .  
 $\text{distinct}_a(R)$ : cantidad de elementos distintos en el campo  $R.a$ .  
 $\text{sel}_p(R)$ : fracción de tuplas/records en  $R$  que satisfacen  $p$ .

los **estimaremos** en otra clase (por ahora los supondremos dados).

# Operadores y tamaño del input

Suposición para todos los operadores físicos siguientes:

*Se asume que para todos los operadores unarios  $\ast(R)$  o binarios  $R \ast S$ , ambas relaciones  $R$  y  $S$  son de tamaño **mayor** a la disponible en el **buffer**.*

Si  $R$  o  $S$  pueden ser almacenadas en el **buffer** (memoria), entonces:

$$\begin{aligned}\text{cost}(\ast(R)) &= \text{cost}(R) \\ \text{cost}(R \ast S) &= \text{cost}(R) + \text{cost}(S)\end{aligned}$$

# Operadores físicos relacionales

Selección ( $\sigma$ )

Unión ( $\cup$ )

Proyección ( $\pi$ )

Elim. duplicados ( $\delta$ )

Join ( $\bowtie$ )

Sorting ( $\tau$ )

Intersección ( $\cap$ )

GroupBy ( $\gamma$ )

# Operadores físicos relacionales

Selección ( $\sigma$ )

Unión ( $\cup$ )

Proyección ( $\pi$ )

Elim. duplicados ( $\delta$ )

Join ( $\bowtie$ )

Sorting ( $\tau$ )

Intersección ( $\cap$ )

GroupBy ( $\gamma$ )

# Definición de Selección ( $\sigma$ )

Selección:  $\sigma_p(R)$ .

- $p$  es una combinación booleana ( $\wedge, \vee$ ) de terminos:

*atributo*<sub>1</sub>   op   *atributo*<sub>2</sub>

*atributo*   op   *constante*

con  $op \in \{=, \leq, \geq, <, >\}$ .

Varios casos:

1. Sin índices.
2. Con índice primario.
3. Con índices secundarios.
4. Distintos filtros/predicados.

## Selección ( $\sigma_p$ ): sin índices

1. Leer todas las tuplas, una a una.
2. Retornar la tupla si satisface el predicado.

### Algoritmo

**input:** Predicado  $p$  y relación (e.g. operador)  $R$ .

```
open()  
└─ R.open()
```

```
close()  
└─ R.close()
```

```
next()
```

```
└─  $t := R.next()$ 
```

```
  while  $t \neq \text{NULL}$  do
```

```
    └─ if  $p(t) = \text{true}$  then
```

```
      └─ return  $t$ 
```

```
    └─  $t := R.next()$ 
```

```
  return NULL
```



## Selección ( $\sigma_p$ ): sin índices

Costo y parámetros de  $\sigma_p(R)$ :

$$\text{cost}(\sigma_p(R)) = \text{cost}(R)$$

$$\text{pages}(\sigma_p(R)) = \text{sel}_p(R) \cdot \text{pages}(R)$$

$$|\sigma_p(R)| = \text{sel}_p(R) \cdot |R|$$

Selección ( $\sigma_p$ ): primary index,  $p := A = v$

1. Buscamos la primera tupla que satisfice  $A = v$ .
2. Retornar las siguientes tuplas haciendo next del índice.

## Algoritmo

**input:** Predicado  $p := A = v$ , relación  $R$  y índice  $I$ .

open()

```
| I.open()  
| I.search( $A = v$ )
```

close()

```
| I.close()
```

next()

```
|  $t := I.next()$   
| if  $t \neq \text{NULL}$  then  
|   return  $t$   
| return NULL
```

Selección ( $\sigma_p$ ): primary index,  $p := A = v$

Costo y parámetros de  $\sigma_p(R)$  con primary index  $I$ :

$$\begin{aligned}\text{cost}(\sigma_p(R)) &= \underbrace{\text{cost}(I)}_{\approx 3} + \text{sel}_p(R) \cdot \text{pages}(R) \\ \text{pages}(\sigma_p(R)) &= \text{sel}_p(R) \cdot \text{pages}(R) \\ |\sigma_p(R)| &= \text{sel}_p(R) \cdot |R|\end{aligned}$$

- $\text{cost}(I)$  depende del tipo de índice.
- En general, este costo es aproximadamente 3, pero depende del índice.

## Selección ( $\sigma_p$ ): Secondary index, $p := A = v$

1. Buscamos la primera tupla que satisface  $A = v$ .
2. Por cada **data entry**  $k^*$  que satisface  $A = v$ :
  - Buscamos la página en  $k^*.RID$ .
  - Retornamos la tupla con RID igual a  $k^*.RID$ .

### Algoritmo

**input:** Predicado  $p := A = v$ , relación  $R$  y índice  $I$ .

```
open()
┌   I.open()
└   I.search( $A = v$ )
close()
┌   I.close()
```

```
next()
┌    $k^* := I.next()$ 
└   if  $k^* \neq \text{NULL}$  then
    ┌   return  $R.get(k^*.RID)$ 
    └   return NULL
```

Selección ( $\sigma_p$ ): secondary index,  $p := A = v$

Costo y parámetros de  $\sigma_p(R)$  con primary index  $I$ :

$$\begin{aligned}\text{cost}(\sigma_p(R)) &= \underbrace{\text{cost}(I)}_{\approx 3} + \text{sel}_p(R) \cdot |R| \\ \text{pages}(\sigma_p(R)) &= \text{sel}_p(R) \cdot \text{pages}(R) \\ |\sigma_p(R)| &= \text{sel}_p(R) \cdot |R|\end{aligned}$$

- Notar la diferencia con el caso anterior:

$$\text{sel}_p(R) \cdot \text{pages}(R) \ll \text{sel}_p(R) \cdot |R|$$

- Si  $\text{sel}_p(R) \rightarrow 0$ , entonces la diferencia de costo es mínima.

Selección ( $\sigma_p$ ): primary/secondary index,  $p := A = v$

¿qué ocurre si  $A$  es clave primaria?

Costo/parámetros de  $\sigma_p(R)$  con clave primaria  $A$ :

$$\begin{aligned}\text{cost}(\sigma_p(R)) &= \underbrace{\text{cost}(I)}_{\approx 3} \\ \text{pages}(\sigma_p(R)) &= 1 \\ |\sigma_p(R)| &= 1\end{aligned}$$

Esto demuestra la importancia de una buena **normalización**!

Selección ( $\sigma_p$ ): primary/secondary index,  $p := A \leq v$

Mismos costo/parámetros anteriores,  
pero solamente posible si índice soporta **range queries** (B+-tree).

Recordar que para Hash index:

- no soporta range queries.
- costo/parámetros es el mismo que hacer una selección sin índice.

# Selección ( $\sigma_p$ ): caso general

Consideramos tres casos:

## 1. Predicados **conjuntivos**:

$$\overbrace{(A_1 \text{ op}_1 v_1) \text{ AND } \dots \text{ AND } (A_k \text{ op}_k v_k)}^{\text{conjunción}}$$

$\underbrace{\hspace{10em}}_{\text{termino}}$

## 2. Predicados **disjuntivos**:

$$\overbrace{(A_1 \text{ op}_1 v_1) \text{ OR } \dots \text{ OR } (A_k \text{ op}_k v_k)}^{\text{disyunción}}$$

$\underbrace{\hspace{10em}}_{\text{termino}}$

## 3. Predicados booleanos sin restricciones.



# Selección ( $\sigma_p$ ): predicados conjuntivos

Varias posibilidades:

$$p := (A_1 \text{ op}_1 v_1) \text{ AND } \dots \text{ AND } (A_k \text{ op}_k v_k)$$

1. Hacemos un **scan** sobre toda la relación y filtramos tuplas.
2. Usamos un índice **multidimensional** (ej: bitmap index).
3. Escogemos la conjunción  $p_i := (A_i \text{ op}_i v_i)$  que:
  - tenga un **índice** sobre  $A_i$  y
  - con **mayor selectividad** (menor  $\text{sel}_{p_i}(R)$ ).

Las tuplas/records retornadas por el índice para  $p_i$  las filtramos con  $p$ .

4. Tratamos de usar **varios índices** a la vez.
  - Si la selectividad de cada predicado es baja.
  - Todos los índices son unclustered/secundarios (¿por qué?).

## Selección ( $\sigma_p$ ): predicados disyuntivos

$$(A_1 \text{ op}_1 v_1) \text{ OR } \dots \text{ OR } (A_k \text{ op}_k v_k)$$

Dos posibilidades:

1. si **al menos un termino NO hace match** ningún índice, entonces hacemos filter/scan de toda la relación. (¿por qué?)
2. si **cada termino hacen match** con al menos un índice, entonces:
  - Evaluar cada termino por separado.
  - Hacer la unión de todos los resultados.
  - Eliminar duplicados.

## Selección ( $\sigma_p$ ): predicados booleanos sin restricciones

Combinación de las anteriores.

¿alguna estrategia?

1. Buscar un termino disyuntivo que nos obligue a realizar un filter/scan.
2. Buscar un termino conjuntivo que tenga mayor selectividad.

# Operadores físicos relacionales

Selección ( $\sigma$ )

Unión ( $\cup$ )

Proyección ( $\pi$ )

Elim. duplicados ( $\delta$ )

Join ( $\bowtie$ )

Sorting ( $\tau$ )

Intersección ( $\cap$ )

GroupBy ( $\gamma$ )

# Proyección ( $\pi_L$ )

Proyección  $\pi_L$  es muy sencillo (especial para **pipeline**):

1. Leemos las tuplas/records de  $R$ , una a una.
2. **Proyectamos** en los atributos  $L$ .

## Algoritmo

**input:** Lista de atributos  $L$  y operador  $R$ .

```
open()
└─ R.open()

close()
└─ R.close()

next()
└─ t := R.next()
   if t ≠ NULL then
       └─ return t.project(L)
   return NULL
```

# Proyección ( $\pi_L$ )

Costo y parámetros de  $\pi_L(R)$ :

$$\text{cost}(\pi_L(R)) = \text{cost}(R)$$

$$\text{pages}(\pi_L(R)) = \frac{\text{rsize}(\pi_L(R))}{\text{rsize}(R)} \cdot \text{pages}(R)$$

$$|\pi_L(R)| = |R|$$

$$\text{rsize}(\pi_L(R)) = \sum_{att \in L} \mathbb{E}(|\pi_{att}(R)|)$$

Esta proyección no considera eliminación de duplicados.

# Operadores físicos relacionales

Selección ( $\sigma$ )

Unión ( $\cup$ )

Proyección ( $\pi$ )

Elim. duplicados ( $\delta$ )

Join ( $\bowtie$ )

Sorting ( $\tau$ )

Intersección ( $\cap$ )

GroupBy ( $\gamma$ )

# Eliminación de duplicados ( $\delta$ )

Dos posibles implementaciones:

1. Basado en **sorting**.
2. Basado en **hashing**.



# Eliminación de duplicados ( $\delta$ ): Sorting

1. Hacemos external merge-sort de las tuplas/records.
2. En el último paso, filtramos según la última tupla leída.

## Algoritmo

**input:** Operador  $R$ .

```
open()
┌    $R' := \text{merge-sort}(R)$ 
├    $R'.\text{open}()$ 
└    $t := \text{NULL}$ 

close()
┌    $R'.\text{close}()$ 
```

```
next()
┌    $t' := R.\text{next}()$ 
├   while  $t' \neq \text{NULL}$  do
├       if  $t \neq t'$  then
├           ┌    $t := t'$ 
├           └   return  $t$ 
├        $t' := R.\text{next}()$ 
└   return  $\text{NULL}$ 
```

# Eliminación de duplicados ( $\delta$ ): Sorting

Costo y parámetros de  $\delta(R)$  con sorting:

$$\text{cost}(\delta(R)) = \text{cost}(R) + 2 \cdot \text{pages}(R)$$

$$\text{pages}(\delta(R)) = \text{distinct}(R) \cdot \frac{\text{rsize}(R)}{|\text{page}|}$$

$$|\delta(R)| = \text{distinct}(R)$$

$$\text{rsize}(\tau(R)) = \text{rsize}(R)$$

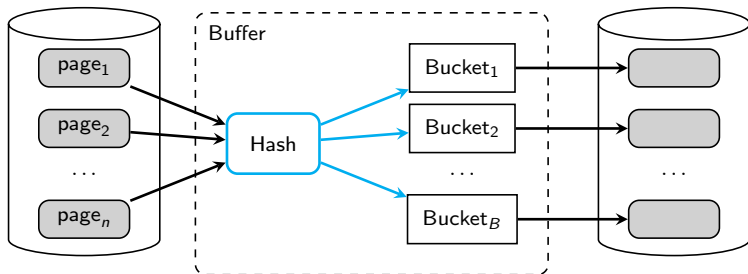
# Eliminación de duplicados ( $\delta$ ): Hashing

Consta de dos partes:

1. Fase de partición.
2. Fase de eliminación de duplicados.

# Eliminación de duplicados ( $\delta$ ): Hashing

## 1. Fase de partición.



# Eliminación de duplicados ( $\delta$ ): Hashing

## 1. Fase de partición.

- Cada página  $p$  del operador  $R$  se lee a memoria.
- Por cada  $t \in p$  se computa  $h(t)$  y se envía al bucket  $\text{Bucket}_{h(t)}$ .
- Si un bucket esta **completo**, se vacía y **materializa** en disco.
- Cada bucket construye una secuencia de **overflow pages** en disco.

# Eliminación de duplicados ( $\delta$ ): Hashing

1. Fase de partición.
2. Fase de eliminación de duplicados.
  - Cada bucket  $b$  se lee a memoria.
  - Se eliminan los duplicados con un algoritmo de memoria interna.
  - Si bucket  $b$  no cabe en memoria, se itera la fase de **partición** con  $b$ .

¿cuánto es la cantidad optima de buckets a utilizar?

# Eliminación de duplicados ( $\delta$ ): Hashing

Costo y parámetros de  $\delta(R)$  con **hashing**:

$$\begin{aligned}\text{cost}(\delta(R)) &= \text{cost}(R) + 2 \cdot \text{pages}(R) \\ \text{pages}(\delta(R)) &= \text{distinct}(R) \cdot \frac{\text{rsize}(R)}{|\text{page}|} \\ |\delta(R)| &= \text{distinct}(R) \\ \text{rsize}(\tau(R)) &= \text{rsize}(R)\end{aligned}$$

... similar a la versión basada en **sorting**.

# Eliminación de duplicados ( $\delta$ ): Sorting vs. Hashing

- ¿qué ventajas tiene la versión basada en **sorting**?
  - Los resultados quedan ordenado.
  - No se ve afectado por datos sesgados.
- ¿qué ventajas tiene la versión basada en **hashing**?
  - Posibilidad de usar más buffer.



# Operadores físicos relacionales

Selección ( $\sigma$ )

Unión ( $\cup$ )

Proyección ( $\pi$ )

Elim. duplicados ( $\delta$ )

Join ( $\bowtie$ )

Sorting ( $\tau$ )

Intersección ( $\cap$ )

GroupBy ( $\gamma$ )

# GroupBy ( $\gamma$ )

Dos posibles implementaciones:

1. Basado en **sorting**.
2. Basado en **hashing**.

Los mismos algoritmos para duplicados pero con **agregación!**

... ¿ cómo ?

# Operadores físicos relacionales

Selección ( $\sigma$ )

Unión ( $\cup$ )

Proyección ( $\pi$ )

Elim. duplicados ( $\delta$ )

Join ( $\bowtie$ )

Sorting ( $\tau$ )

Intersección ( $\cap$ )

GroupBy ( $\gamma$ )

# Operador de unión ( $\cup$ )

SQL cuenta con dos operadores para unión:

- UNION.
- UNION ALL.

¿cuál es la diferencia?

# Unión ALL ( $R \cup S$ ): con duplicados

Dos alternativas:

1. Si  $R$  y  $S$  no están ordenados: leemos y imprimimos.
2. Si  $R$  y  $S$  están ordenados: leemos, ordenamos y imprimimos.
  - Hacemos **merge** de  $R$  y  $S$ .
  - No aumenta el costo I/O del operador.

## Unión ALL ( $R \cup S$ ): con duplicados

Costo y parámetros de  $R \cup S$ :

$$\text{cost}(R \cup S) = \text{cost}(R) + \text{cost}(S)$$

$$\text{pages}(R \cup S) = \text{pages}(R) + \text{pages}(S)$$

$$|R \cup S| = |R| + |S|$$

$$\text{rsize}(R \cup S) = \text{rsize}(R)$$

# Unión ( $R \cup S$ ): sin duplicados

Mismas alternativas que para eliminación de duplicados:

## 1. Basado en **sorting**.

- Ordenamos ambas relaciones con sorting externo.
- Hacemos merge de ambas relaciones.

## 2. Basado en **hashing**.

- Particionamos  $R$  y  $S$  con una función de hash.
- Eliminamos duplicados en cada partición.

Mismo costo que para **eliminación de duplicados**.