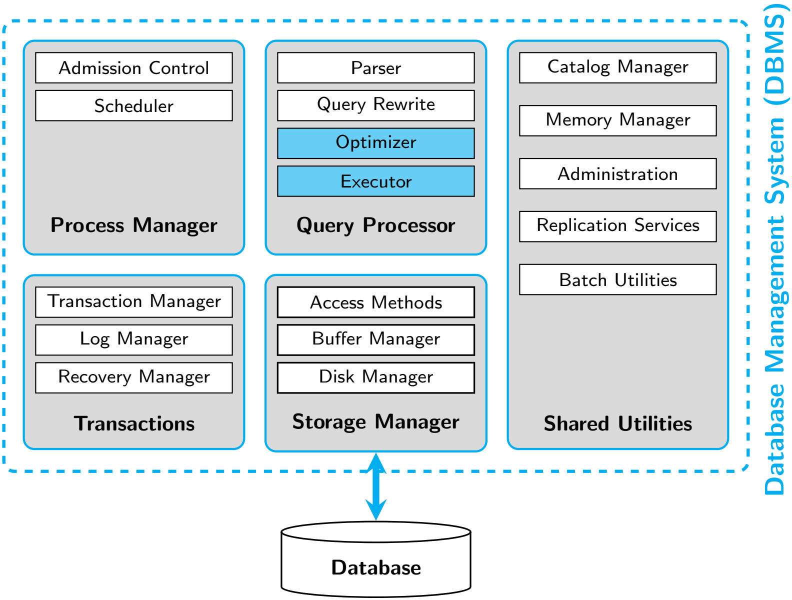
Implementación de operadores relacionales

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 algoritmo de **operadores físicos** se pueden dividir en:

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

Outline

Outline

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

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

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

Parámetros para medir costo

Parámetros de interés:

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

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:

$$\begin{array}{rcl} \textit{attributo}_1 & \text{op} & \textit{attributo}_2 \\ \textit{attributo} & \text{op} & \textit{constante} \end{array}$$

con 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.
$\lvert R \rvert$:	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...

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

Los siguientes parámetros:

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 $*(R)$ o binarios $R * S$, 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:

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

Operadores físicos relacionales

Selección (σ)	Unión ($\cup$)
Proyección (π)	Elim. duplicados (δ)
Join ($\bowtie$)	Sorting (τ)
Intersección ($\cap$)	GroupBy (γ)

Operadores físicos relacionales

Selección (σ)	Unión ($\cup$)
Proyección (π)	Elim. duplicados (δ)
Join ($\bowtie$)	Sorting (τ)
Intersección ($\cap$)	GroupBy (γ)

Definición de Selección (σ)

Selección: $\sigma_p(R)$.

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

$$\begin{array}{rcl} attributo_1 & \text{op} & attributo_2 \\ attributo & \text{op} & constante \end{array}$$

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

Varios casos:

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

Selección (σ_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 ≠ NULL do
        if p(t) = true then
            return t
        t := R.next()
    return NULL
```

Selección (σ_p): sin índices

Costo y parámetros de $\sigma_p(R)$:

$$
\begin{aligned}
\text{cost}(\sigma_p(R)) &= \text{cost}(R) \\
\text{pages}(\sigma_p(R)) &= \text{sel}_p(R) \cdot \text{pages}(R) \\
|\sigma_p(R)| &= \text{sel}_p(R) \cdot |R|
\end{aligned}
$$

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

1. Buscamos la primera tupla que satisface $A = v$.
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 ≠ NULL then
        return t
    return NULL
```

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

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

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

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

Selección (σ_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* ≠ NULL then
        return R.get(k*.RID)
    return NULL
```

Selección (σ_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:

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

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

Selección (σ_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 (σ_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 (σ_p): caso general

Consideramos tres casos:

1. Predicados **conjuntivos**:

$$\overbrace{\underbrace{(A_1 \ \mathsf{op}_1 \ v_1)}_{\text{termino}} \ \texttt{AND} \ \cdots \ \texttt{AND} \ (A_k \ \mathsf{op}_k \ v_k)}^{\text{conjunción}}$$

2. Predicados **disjuntivos**:

$$\overbrace{\underbrace{(A_1 \ \mathsf{op}_1 \ v_1)}_{\text{termino}} \ \texttt{OR} \ \cdots \ \texttt{OR} \ (A_k \ \mathsf{op}_k \ v_k)}^{\text{disyunción}}$$

3. Predicados booleanos sin restricciones.

Selección (σ_p): predicados conjuntivos

Varias posibilidades:

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

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 (σ_p): predicados disyuntivos

$$(A_1 \ \mathsf{op}_1 \ v_1) \ \ \mathtt{OR} \ \ \cdots \ \ \mathtt{OR} \ \ (A_k \ \mathsf{op}_k \ v_k)$$

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 (σ_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 (σ)	Unión ($\cup$)
Proyección (π)	Elim. duplicados (δ)
Join ($\bowtie$)	Sorting (τ)
Intersección ($\cap$)	GroupBy (γ)

Proyección (π_L)

Proyección π_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 (π_L)

Costo y parámetros de $\pi_L(R)$:

$$
\begin{aligned}
\text{cost}(\pi_L(R)) &= \text{cost}(R) \\
\text{pages}(\pi_L(R)) &= \frac{\text{rsize}(\pi_L(R))}{\text{rsize}(R)} \cdot \text{pages}(R) \\
|\pi_L(R)| &= |R| \\
\text{rsize}(\pi_L(R)) &= \sum_{att \in L} \mathbb{E}(|\pi_{att}(R)|)
\end{aligned}
$$

Esta proyección no considera eliminación de duplicados.

Operadores físicos relacionales

Selección (σ)	Unión ($\cup$)
Proyección (π)	Elim. duplicados (δ)
Join ($\bowtie$)	Sorting (τ)
Intersección ($\cap$)	GroupBy (γ)

Eliminación de duplicados (δ)

Dos posibles implementaciones:

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

Eliminación de duplicados (δ): Sorting

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

Algoritmo

input: Operador R.

```
open()
    R' := merge-sort(R)
    R'.open()
    t := NULL

close()
    R'.close()

next()
    t' := R.next()
    while t' ≠ NULL do
        if t ≠ t' then
            t := t'
            return t
        t' := R.next()
    return NULL
```

Eliminación de duplicados (δ): Sorting

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

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

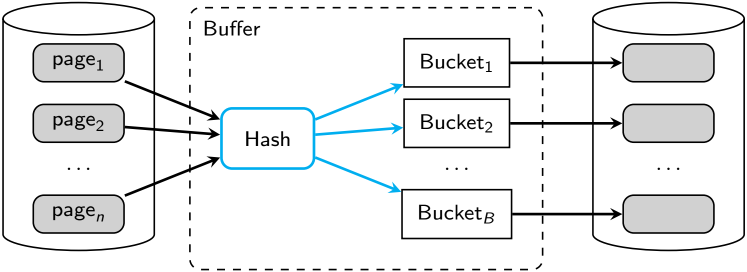
Eliminación de duplicados (δ): Hashing

Consta de dos partes:

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

Eliminación de duplicados (δ): Hashing

1. Fase de partición.



Eliminación de duplicados (δ): 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 (δ): 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 (δ): 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 (δ): 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 (σ)	Unión ($\cup$)
Proyección (π)	Elim. duplicados (δ)
Join ($\bowtie$)	Sorting (τ)
Intersección ($\cap$)	GroupBy (γ)

GroupBy (γ)

Dos posibles implementaciones:

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

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

... ¿ cómo ?

Operadores físicos relacionales

Selección (σ)

Unión ($\cup$)

Proyección (π)

Elim. duplicados (δ)

Join ($\bowtie$)

Sorting (τ)

Intersección ($\cap$)

GroupBy (γ)

Operador de unión (∪)

`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 $R \cup S$:

$$
\begin{aligned}
\text{cost}(R \cup S) &= \text{cost}(R) + \text{cost}(S) \\
\text{pages}(R \cup S) &= \text{pages}(R) + \text{pages}(S) \\
|R \cup S| &= |R| + |S| \\
\text{rsize}(R \cup S) &= \text{rsize}(R)
\end{aligned}
$$

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**.