PROCESAMIENTO DE CONSULTAS

fs Seleccion

- fs(A = c, r) = 1/V(A, r)
- fs(A ≥ c, r) = (max(A, r) - c) / (max(A, r) – min(A, r))
- fs(A < c, r) = (c – min(A)) / (max(A) – min(A) + 1)
- fs(P1Λ P2 Λ…Λ Pn, r) = fs(P 1 , r) fs(P 2 , r) …. fs(P n , r)

Algoritmos selección (con igualdad):

- **Búsqueda lineal** :
  - Estimación de costo = b_r transferencias de bloques + 1 acceso a bloque
  - b_r denota el número de bloques conteniendo registros de la tabla r

- Algoritmo para índice **primario** usando árbol B+ con **igualdad en clave candidata:**
  - Costo = (hi + 1), **hi = *roof*(log *roof*(n/2) (K))**, donde
    - **n=entradas por bloque**
    - **K = valores de clave de búsqueda en la tabla**

- Algoritmo para índice **primario** usando árbol B+ **igualdad para NO clave candidata**:
  - Costo = hi + b, donde
    - **hi es la altura del arbol B+**
    - **b es el número de bloques conteniendo registros con la clave de búsqueda especificada**

- Algoritmo para índice **secundario** usando árbol B+, **igualdad en clave candidata:**
  - Costo igual a idem para primario con igualdad en clave candidata

- Algoritmo para índice **secundario** usando árbol B+, **igualdad en no clave**
  - Sea n el número de registros recogidos.
  - Costo = (hi + n)

Algoritmos de selección (con comparaciones):

- Algoritmo para **índice primario**:
  - Para **A >= v**:
    - La tabla está ordenada en A.
    - Costo = hi+b, con b número de bloques conteniendo registros con A > v
  - Para **A <= v:**
    - se escanea la tabla secuencialmente hasta encontrar A > v, no se usa el indice

- Algoritmo para **índice secundario**:
  - Para **A >= v**:
    - Costo = (hi + n), n número de registros con A >= v.
  - Para **A <= v:**
    - escanear hojas del índice encontrando punteros a los registros, hasta la primera entrada > v.

PROYECCIÓN

- Algoritmo para proyección:
  - costo = b_r transferencias de bloques + 1 acceso a bloque
  - b_r numero de bloques contenidos en la tabla r

ORDENAMIENTO

- Merge sort externo;
  - Costo: b_r (2* roof(log roof(M –1) (b_r / M) + 1)) **transferencias de bloque**
  - b_r cantidad de bloques en la tabla
  - 2 * roof(b_r / M) + roof(b_r/ b_b) * (2 * roof(log floor(M / b_b) – 1 (b_r/M)) -1) **accesos a bloques**
  - b_b bloques son alojados en cada corrida

EIIMINACION DE DUPLICADOS

- Costo: el peor caso es igual al del ordenamiento

REUNION SELECTIVA

- **fs(r.A = s.B, r, s) = 1 /max(V(A, r), V(B, s))**
- Algoritmo de reunión selectiva de loop anidado:
    - $n_r * b_s + b_r$ transferencias de bloques
    - $n_r + b_r$ accesos a bloques.
    - donde n_r es cantidad de registros en r, b_r es la cantidad de bloques en r, y b_s es la cantidad de bloques en s

- loop anidado por bloques:
    - ordenar tablas en el atributo de la reunion
    - $b_r * b_s + b_r$ transferencias de bloque + $2 * b_r$ accesos a bloque

- merge sort:
    - $b_s + b_r$ transferencias de bloques
    - roof(b_s / b_b) + roof(b_r / b_b)
    - b_b bloques asignados a cada tabla
    - + el costo de ordenar si las tablas no estan ordenadas

AGREGACION

- Mismo costo que la eliminación de duplicados

CONCATENACION

- Peor caso $b_s + b_r$ transferencias de bloques y accesos

INTERSECCION

- Ordenar tablas por su clave primaria, $b_s + b_r$ transferencias de bloques y accesos

RESTA

- Mismo que intersección

CONSULTAS

PROYECCION

$$\Pi_{f_1,\ldots,f_N}(r) = \mathbf{map}\ (\backslash t \rightarrow (f_1 t,\ \ldots,\ f_N t))\ r$$

SELECCION

$$\sigma_p(r) :: R$$
$$\sigma_p(r) = \mathbf{foldr}\ (\backslash t\ r' \rightarrow \mathbf{if}\ p\ t\ \mathbf{then}\ x : r'\ \mathbf{else}\ r')\ []\ r$$

PRODUCTO CARTESIANO

$$r \times s = \mathbf{foldr}\ (\mathrm{anexar}\ s)\ []\ r$$
anexar s x q = map (\t -> (x ; t)) s ++ q

REUNIÓN SELECTIVA

$$r\ {}_{a_1,\ldots,a_i}\bowtie_{b_1,\ldots,b_i} s = \Pi_{a_1,\ldots,a_N,c_1,\ldots,c_{M-i}}(\sigma_{a_1=b_1\wedge\ldots\wedge a_i=b_i}(r \times s))$$

CONCATENACIÓN

$$r{+}{+}s = \mathbf{foldr}\ (:)\ s\ r$$

RESTA

$$r \setminus s = \sigma_{(\backslash t \rightarrow t \notin s)}(r)$$

INTERSECCIÓN

$$r \cap s = \sigma_{(t \rightarrow t \in s)}(r)$$

REMOVER DUPLICADOS

$$v(r) = \mathbf{foldr}\ (\backslash t\ s \rightarrow \mathbf{if}\ t \in s\ \mathbf{then}\ s\ \mathbf{else}\ (t:s))\ []\ r$$

AGREGACIÓN

$$\gamma_{f1(a1),\ \ldots,\ fm(am)}(r) = [\ (f_1\ (\mathrm{map}\ (\backslash t\text{-}>\ t.a_1)\ r),\ \ldots,\ f_m\ (\mathrm{map}\ (\backslash t\text{-}>\ t.a_m)\ r)\ ]$$

AGRUPACION

$$_{g_1,\ldots,g_N}\gamma_{f_1(n_1),\ldots,f_M(n_M)}(r) = \begin{array}{l}\textbf{let } \mathsf{p} = v(\Pi_{g_1,\ldots,g_N}(r)) \\ \textbf{map } (\backslash \mathsf{t} \rightarrow (\mathsf{t}; \gamma_{f_1(n_1),\ldots,f_M(n_M)}(\sigma_{g_1=t.g_1 \wedge \ldots \wedge g_N=t.g_N}(r))[0])) \ \mathsf{p}\end{array}$$

ORDENAMIENTO

```
1 insert_{a_1,...,a_N} t r = σ_{(a_1,...,a_N) < t[a_1,...,a_N]}(r) ++ [t] ++ σ_{(a_1,...,a_N) ≥ t[a_1,...,a_N]}(r)
2
3 O_{a_1,...,a_N}(r) = foldr (insert_{a_1,...,a_N}) [] r
```