Optimización de consultas

Sivana Hamer - sivana.hamer@ucr.ac.cr
Escuela de Ciencias de la Computación

El álgebra relacional es un lenguaje formal que define un conjunto de operadores para el modelo relacional por medio de expresiones

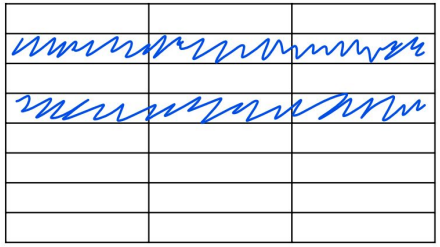
$$\sigma_{Fname='Pedro' AND Byear>1990}(STUDENT)$$

Expresión de álgebra relacional

Operadores de álgebra relacional

La operación SELECT (σ) selecciona todas las tuplas que cumplen una condición (c) de una relación (R).

$$\sigma_c(R)$$



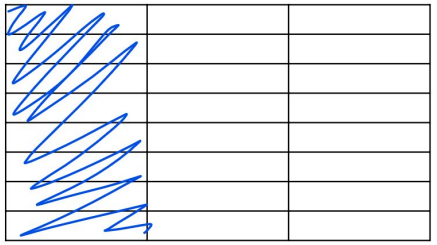
Example: Get all the students who are named Pedro and are born after 1990.

$$\sigma_{Fname='Pedro' AND Byear>1990}(STUDENT)$$

```
SELECT *
FROM STUDENT
WHERE Fname = 'Pedro' AND Byear > 1990
```


La operación PROYECCIÓN (π) selecciona una lista de atributos (L) de una relación (R).

$$\pi_L(R)$$



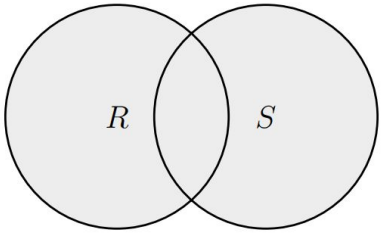
Example: Get the first and last name for all the students.

$$\pi_{Fname,Lname}(STUDENT)$$

```
SELECT DISTINCT Fname, Lname
FROM STUDENT
```


La operación UNION (∪) obtiene todas las tuplas en R o S, eliminando las tuplas duplicadas.

$$R \cup S$$



Example: Get the list of all students or instructors.

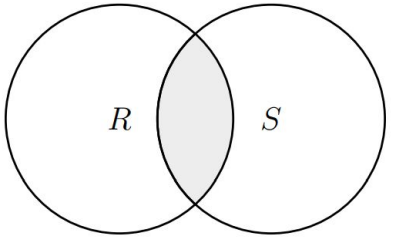
$$STUDENT \cup INSTRUCTOR$$

```
SELECT *
FROM STUDENT
UNION
SELECT *
FROM INSTRUCTOR
```

* Se debe tener los mismos atributos en ambas tablas

La operación INTERSECCIÓN (∩) obtiene todas las tuplas en R y S.

$$R \cap S$$



Example: Get the list of people who are both students and instructors.

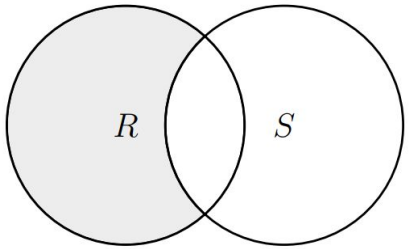
$$STUDENT \cap INSTRUCTOR$$

```
SELECT *
FROM STUDENT
INTERSECTION
SELECT *
FROM INSTRUCTOR
```

* Se debe tener los mismos atributos en ambas tablas

La operación MENOS (-) obtiene todas las tuplas en R pero no en S.

$$R - S$$



Example: Get the list of students who are not instructors.

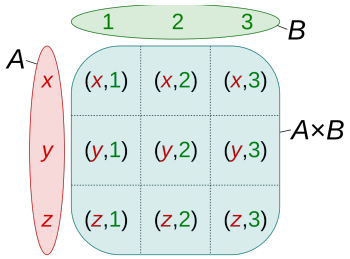
$$STUDENT - INSTRUCTOR$$

```
SELECT *
FROM STUDENT
EXCEPT
SELECT *
FROM INSTRUCTOR
```

* Se debe tener los mismos atributos en ambas tablas

La operación PRODUCTO CARTESIANO (×) combina cada tupla del conjunto R(A1, ..., An) con cada tupla en el conjunto S(B1, ..., Bm)

$$R \times S$$



Example: Get all the combinations of students enrolled to a course.

$$STUDENT \times ENROLLED$$

```
SELECT *
FROM STUDENT, ENROLLED
```

or

```
SELECT *
FROM STUDENT CROSS JOIN ENROLLED
```


La operación JOIN (⋈) combina dos relaciones (R y S) basado en una condición c.

$$R \bowtie_c S$$

R

A	B
W	1
Y	2
Y	1
Z	3

S

B	C
1	A
2	B

R⋈S

A	B	C
W	1	A
Y	2	B
Y	1	A

Example: Get the courses students have enrolled.

$$(\rho_S(STUDENT)) \bowtie_{S.id=E.id} (\rho_E(ENROLLED))$$

```
SELECT *
FROM STUDENT AS S, ENROLLED AS E
WHERE S.id = E.id
```

or

```
SELECT *
FROM STUDENT AS S JOIN ENROLLED AS E ON S.id = E.id
```


Equivalencias en álgebra relacional

1. Cascada de σ.

Una condición de selección con *AND* se puede dividir en una secuencia de operaciones σ.

$$\sigma_{c_1 AND c_2 AND \cdots AND c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\cdots(\sigma_{c_n}(R))\cdots))$$

Example: Get the students with the first name Pedro, born after 1990.

$$\sigma_{Fname='Pedro' AND Byear>1990}(STUDENT) \equiv$$

2. Conmutatividad de σ.

σ puede ser aplicado en cualquier orden.

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$$

Example: Get the students with the first name Pedro, born after 1990.

$$\sigma_{Byear>1990}(\sigma_{Fname='Pedro'}(STUDENT) \equiv$$

3. Cascada de π.

Cuando hay una secuencia de operaciones π, todas excepto la más externa pueden ser ignoradas.

$$\pi_{List_1}(\pi_{List_2}(\cdots(\pi_{List_n}(R))\cdots)) \equiv \pi_{List_1}(R)$$

Example: Get the if of the students.

$$\pi_{id}(\pi_{id,Fname}(\pi_{id,Fname,Lname}(STUDENT))) \equiv$$

4. Conmutatividad de σ con π.

Si la condición de selección *c* involucra sólo los atributos *A1, A2, · · · , AN* en la lista de proyección, las dos operaciones se pueden aplicar en cualquier orden.

$$\pi_{A_1,A_2,\cdots,A_N}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1,A_2,\cdots,A_N}(R))$$

Example: Get the id with the full name of students with the first name Pedro.

$$\pi_{id,Fname,Lname}(\sigma_{Fname='Pedro'}(STUDENT)) \equiv$$

5. Conmutatividad de ⋈ y ×.

⋈ y × pueden ser aplicados en cualquier orden.

$$R \bowtie_c S \equiv S \bowtie_c R$$

$$R \times S \equiv S \times R$$

Example: Get the courses students have enrolled.

$$STUDENT \bowtie_{S.id=E.id} ENROLLED \equiv$$

6. Conmutatividad de σ con ⋈ y ×.

Si todos los atributos de la condición de selección *c* implican sólo los atributos de las relaciones del join, se pueden aplicar en cualquier orden

$$\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$$

Example: Get the students with the first name Pedro and born after 1990, with their course enrollment information.

$$\sigma_{Fname='Pedro' AND byear>1990}(STUDENT \bowtie_{S.id=E.id} ENROLLED) \equiv$$

6.5. Conmutatividad de σ con ⋈ y ×.

Si *c* puede escribirse como *c1* y *c2* donde *c1* tiene solo los atributos de *R* y *c2* tiene solo los atributos de *S*, se pueden aplicar en cualquier orden.

$$\sigma_c(R \bowtie S) \equiv (\sigma_{c_1}(R)) \bowtie (\sigma_{c_2}(S))$$

Example: Get the students with the first name Pedro and with their enrolled courses where they got grades higher than 90.

$$\sigma_{Fname='Pedro' AND grade>90}(STUDENT \bowtie_{S.id=E.id} ENROLLED) \equiv$$

7. Conmutatividad de π con ⋈ y ×.

Para una lista de atributos de la proyección llamada *L*, tal que *L = A1, · · · , An, B1, · · · , Bm* donde *A1, · · · , An* son atributos de *R* y *B1, · · · , Bn* son atributos de *S*.

$$\pi_L(R \bowtie_c S) \equiv (\pi_{A_1,\cdots,A_n}(R)) \bowtie_c (\pi_{B_1,\cdots,B_m}(S))$$

7. Conmutatividad de π con ⋈ y ×.

Si los atributos *An+1, · · · , An+k* de *R* y *Bm+1, · · · , Bm+p* de *S* están involucrados en la condición de unión pero no están en *L*, entonces estos atributos deben agregarse a la condición de unión. no aplica para el operador ×.

$$\pi_L(R \bowtie_c S) \equiv (\pi_{A_1,\cdots,A_n,A_{n+1},\cdots,A_{n+k}}(R)) \bowtie_c (\pi_{B_1,\cdots,B_m,B_{m+1},\cdots,B_{m+p}}(S))$$

7. Conmutatividad de π con ⋈ y ×.

Example: Get the name and course initials of students enrolled in courses.

$$\sigma_{Fname,Lname,Sigla}(STUDENT \bowtie_{S.id=E.id} ENROLLED) \equiv$$
$$(\pi_{Fname,Lname,id}(STUDENT)) \bowtie_{S.id=E.id} (\pi_{id,Sigla}(ENROLLED))$$

8. Conmutatividad de ∩ y ∪.

∩ y ∪ pueden ser aplicados en cualquier orden.

$$R \cap S \equiv S \cap R$$

$$R \cup S \equiv S \cup R$$

Example: Get the list of people who are both students and instructors.

$$STUDENT \cap INSTRUCTOR \equiv$$

9. Asociatividad de ⋈, ×, ∩ y ∪.

Estas cuatro operaciones, representadas por un θ, pueden ser asociadas en cualquier orden.

$$(R\theta S)\theta T \equiv R\theta(S\theta T)$$

Example: Get for all the students their enrolled courses with the course information.

$$(STUDENT \bowtie_{S.id=E.id} ENROLLED) \bowtie_{E.sigla=C.sigla} COURSE \equiv$$

10. Conmutatividad de σ con −, ∩ y ∪.

Las operaciones de conjuntos ∩, ∪ y − (representadas por θ) se pueden aplicar en cualquier orden con σ.

$$\sigma_c(R\theta S) \equiv (\sigma_c(R))\theta(\sigma_c(S))$$

Example: Get the list of people who are both students and instructors named Pedro.

$$\sigma_{Fname='Pedro'}(STUDENT \cap TEACHER) \equiv$$

11. Conmutatividad de π con ∪.

π y ∪ pueden ser aplicados en cualquier orden.

$$\pi_L(R \cup S) \equiv (\pi_L(R)) \cup (\pi_L(S))$$

Example: Get the list of all students or instructors born after 1990.

$$\pi_{Byear>1990}(STUDENT \cup TEACHER) \equiv$$

12. Convertir una secuencia (σ, ×) en ⋈

If a × is followed by a condition *c* of a σ, then it can be changed into a ⋈.

$$\sigma_c(R \times S) \equiv (R \bowtie_c S)$$

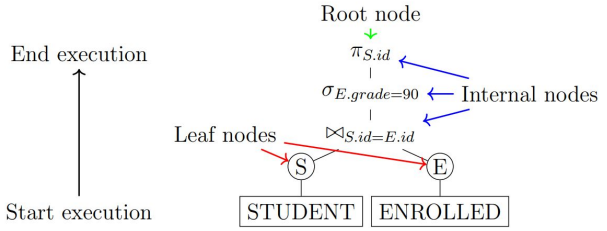
Example: Get the courses students have enrolled.

$$\sigma_{s.id=e.id}(STUDENT \times ENROLLED) \equiv$$

Un árbol del query representa un query de álgebra relacional

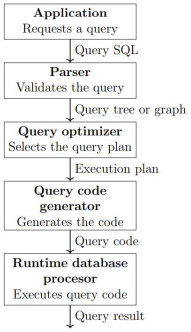
Example: Get the id of students who have gotten a 90 in an enrolled course.

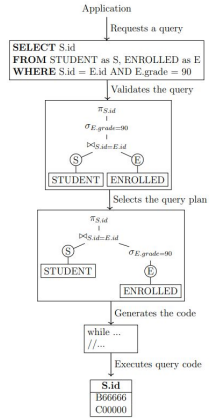
$$\pi_{S.id}(\sigma_{E.grade=90}(STUDENT \bowtie_{S.id=E.id} ENROLLED))$$



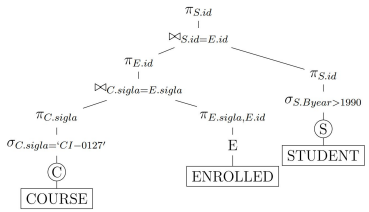
Optimización de consultas

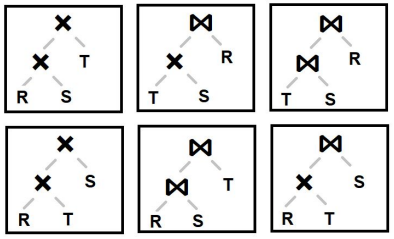
Las consultas del DBMS, son del alto nivel, por lo que se ocupan procesar en un plan de ejecución





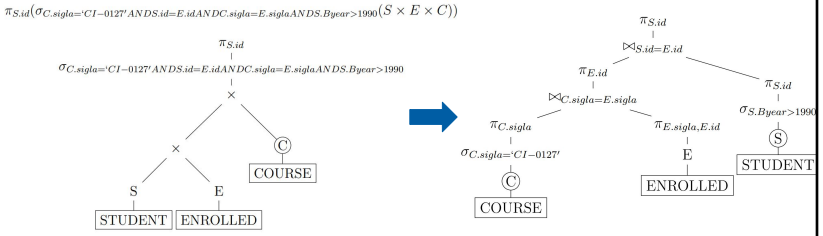
Existen dos niveles de optimización





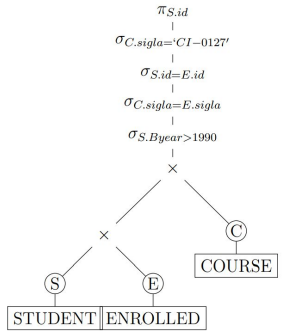
Heurísticas **Basado en costo**

Se utilizan reglas llamadas heurísticas que funcionan en la mayoría de casos, generando un nuevo árbol utilizando reglas de equivalencia de un árbol

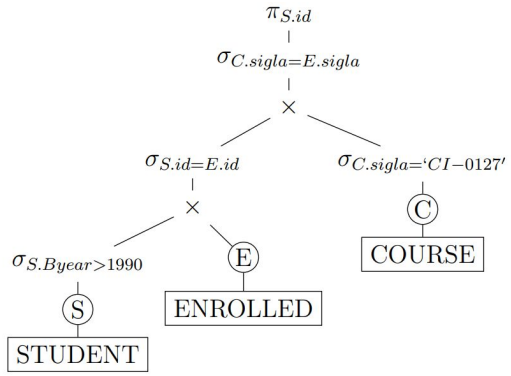


Árbol canónico **Árbol query final**

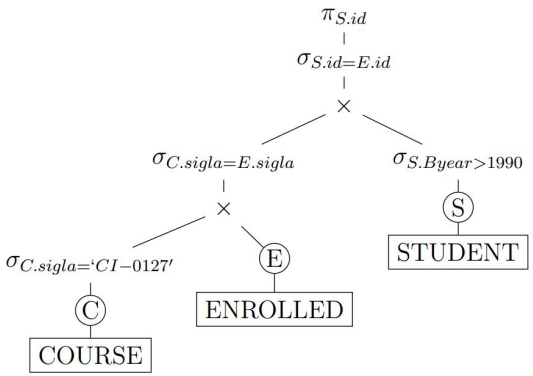
Paso 1. Separar operaciones de σ



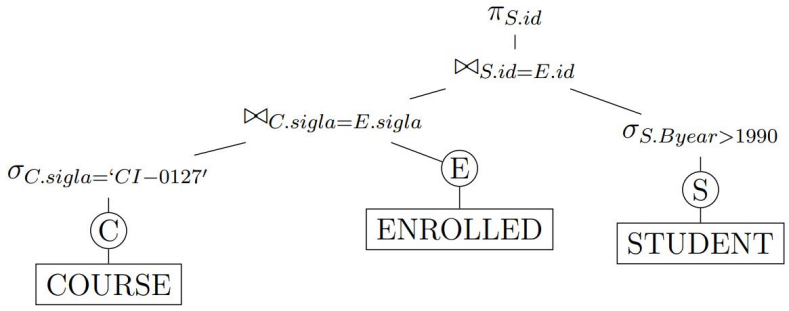
Paso 2. Mover para abajo operaciones de σ



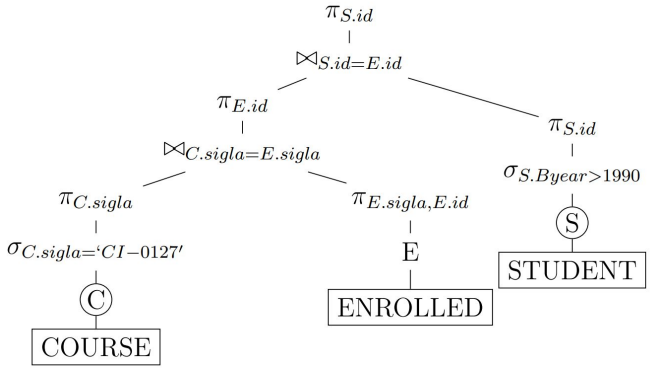
Paso 3. Seleccionar el σ más restrictivo



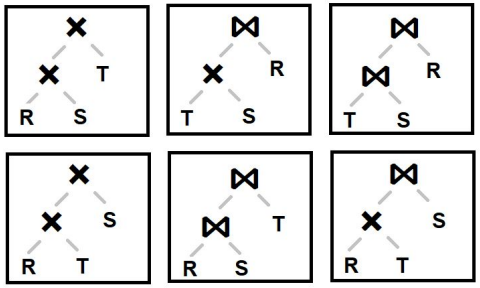
Paso 4. Convertir a ⋈



Paso 5. Mover abajo a π



Para la estimación basado en costos, se estima el costo de cada estrategia y se escoge la que genera el costo menor



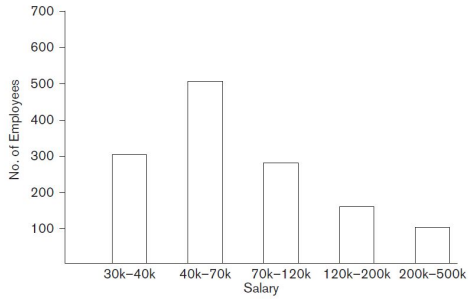
Hay distintas métricas para estimar el costo:

- Costo de acceso a memoria secundaria (I/O cost).
- Costo en almacenamiento de disco.
- Costo computacional (CPU cost).
- Costo de uso de memoria.
- Costo de comunicación (costo de red).

Se utiliza distinta información de la base de datos para estimar funciones de costo

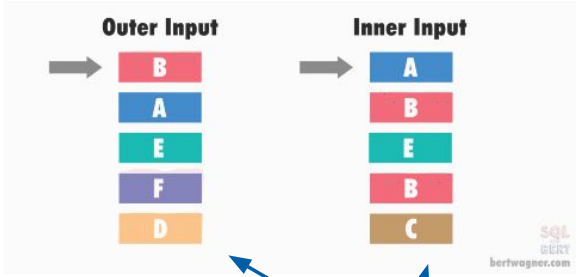
- r_R: Number of tuples for R.
- b_R: Number of blocks occupied in storage for R.
- bfr_R: Number of tuples per block for R.
- $NDV(A, R)$: Number of distinct values of A in R.
- sl_A: Selectivity of attribute A for a condition c. $sl_A = \frac{s_A}{r_R}$
- s_A: Selection cardinality of A. $s_A = sl_A * r_R = \frac{r_R}{NDV(A,R)}$
- x_A: Number of index levels for A.
- b_{I1A}: Number of first level blocks of the index of A.
- n_B: Available buffer space in memory.

Dado que la información de los datos puede cambiar frecuentemente, no se utiliza la distribución de los datos



Se puede estimar el costo de cualquier operador, pero vamos a enfocarnos en los costos de JOIN

Algoritmo 1. Nested-loop join



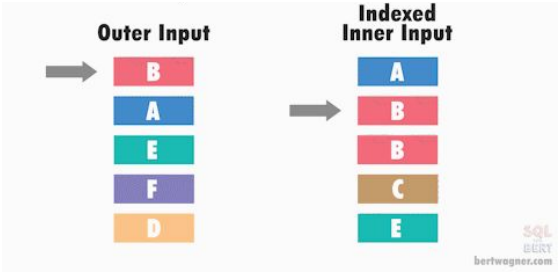
foreach block B_R in R:
  foreach block B_S in s:
    foreach tuple r in B_r:
      foreach tuple s in B_s:
        if $(s == r)$:
          save(s, r)

For the query:

$$STUDENT \bowtie_{S.id=E.id} ENROLLED$$

Suppose $b_S = 13$, $b_E = 2000$, and $n_B = 3$.

Algoritmo 2. Indexed based nested-loop join



El tiempo de usar un indice secundario depende del índice

- **Secondary index:** $b_R + *(r_R * (x_B + 1 + s_B))$
- **Clustering index:** $b_R + *(r_R * (x_B + \frac{s_B}{bfr_B}))$
- **Primary index:** $b_R + *(r_R * (x_B + 1))$
- **Hask key:** $b_R + *(r_R * h))$

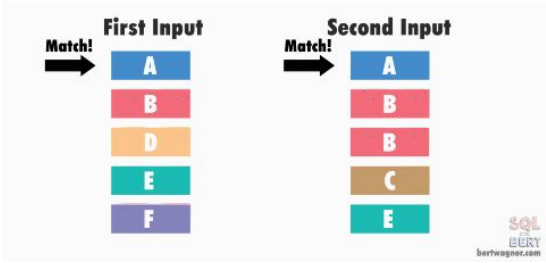
foreach tuple r in R:
    foreach tuple s in **Index** $(r_i = s_j)$:
        if $(s == r)$:
            save (s, r)

For the query:

$$STUDENT \bowtie_{S.id=E.id} ENROLLED$$

Suppose $STUDENT$ has a primary index with $X_{S.id} = 1$. Furthermore, $ENROLLED$ has a secondary index with $X_{E.id} = 2$ and $S_{E.id} = 80$. Also, $b_S = 13$, $b_E = 2000$, $r_E = 10000$, $r_S = 125$, and $n_B = 3$.

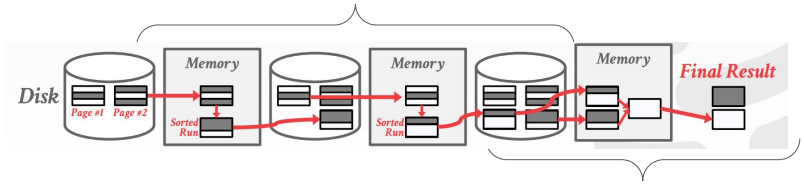
Algoritmo 3. Sort-merge join



¡Requiere que los datos se encuentren ordenados!

Para algunos algoritmos, se utiliza el algoritmo de external merge sort para ordenar la memoría

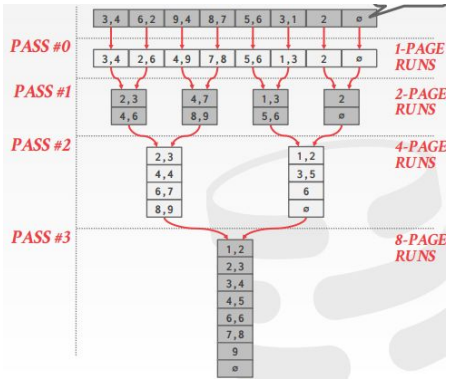
Pass #0: Leer, ordenar páginas y guardar en disco



Pass #1, 2, …: Merge recursivo de parejas, ordenar y guardar

2-way merge sort

https://15445.courses.cs.cmu.edu/fall2021/slides/09-sorting.pdf



https://15445.courses.cs.cmu.edu/fall2021/slides/09-sorting.pdf

El tiempo de un external merge sort es…

de pasadas

$$\underbrace{2 * b *}_{\text{Por cada bloque de disco o página, se lee y escribe una vez}} (log_{d_M}(n_R) + 1)$$

El costo total de external merge sort es…

Sort (si no se encuentra ordenado) *Merge*

$$2 * b * (log_{d_M}(n_R) + 1) + b_R + b_S$$

$$= Sort + Merge$$

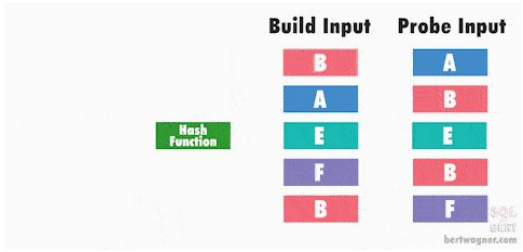

```
//Optional sorts
cursor_R =R.sort(A).cursor()
cursor_S =S.sort(B).cursor()
while (cursor_R AND cursor_S): //We stop if there are no more values
    if (cursor_R > cursor_S):
        cursor_S.next()
    elif (cursor_R < cursor_S):
        cursor_R.next()
    else:
        save(cursor_R, cursor_S)
        cursor_S.next()
```


For the query:

$$STUDENT \bowtie_{S.id=E.id} ENROLLED$$

Suppose $b_S = 13$, $b_E = 2000$, $r_E = 10000$, $r_S = 125$, and $n_B = 3$.

Algoritmo 4. Partition-hash join (hash join)



Cost: $3 * (b_R + b_S)$

Algoritmo 4. Partition-hash join (hash join)

Cost: $3 * (b_R + b_S)$

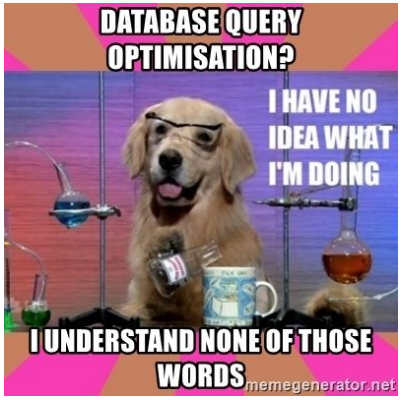
build hash_table HT_R for R
foreach tuple s in S
  if $(h(s)$ in $HT_R)$:
    save (s, r)

For the query:

$$STUDENT \bowtie_{S.id=E.id} ENROLLED$$

Suppose $b_S = 13$, $b_E = 2000$ and $n_B = 3$. Thus:

$$3 * (b_R + b_S) = 3 * (2000 + 13) = 3 * 2013 = 6039$$



Referencias

- R. Elmasri and S. Navathe, Fundamentals of database systems. Pearson, 2016, vol. 7, chapter 6, 18.1 & 19.1
- J. Widom. Relational algebra 1. [Video]. [Online]. Available: https://www.youtube.com/watch?v=tii7xcFilOA&list=PL6hGtHedy2Z4EkgY76QOcueU8lAC4o6c3&index=9
- J. Widom. Relational algebra 2. [Video]. [Online]. Available: https://www.youtube.com/watch?v=GkBf2dZAES0&list=PL6hGtHedy2Z4EkgY76QOcueU8lAC4o6c3&index=10
- https://commons.wikimedia.org/wiki/File:Cartesian_Product_qtl1.svg
- Wagner, Bert (2018). *Visualizing Nested Loops Joins And Understanding Their Implications*. https://bertwagner.com/posts/visualizing-nested-loops-joins-and-understanding-their-implications/

Referencias

- Wagner, Bert (2018). *Visualizing Merge Join Internals And Understanding Their Implications*. https://bertwagner.com/posts/visualizing-merge-join-internals-and-understanding-their-implications/
- Wagner, Bert (2019). *Visualizing Hash Match Join Internals And Understanding Their Implications*. https://bertwagner.com/posts/hash-match-join-internals/