Bases de Datos

Clase 12: Evaluación de consultas

Recordatorio: Paginas, Discos y Buffer

Para trabajar con las tuplas de una relación, la base de datos carga la página desde el disco con dicha tupla

Para cargar estas páginas, la base de datos reserva un espacio en RAM llamado **Buffer**





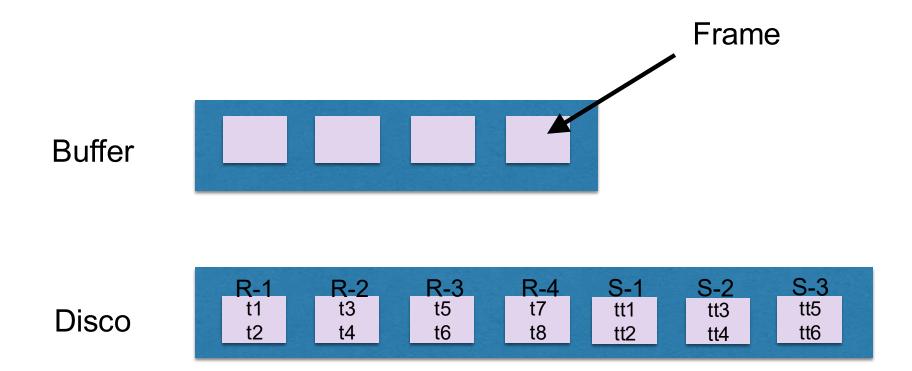


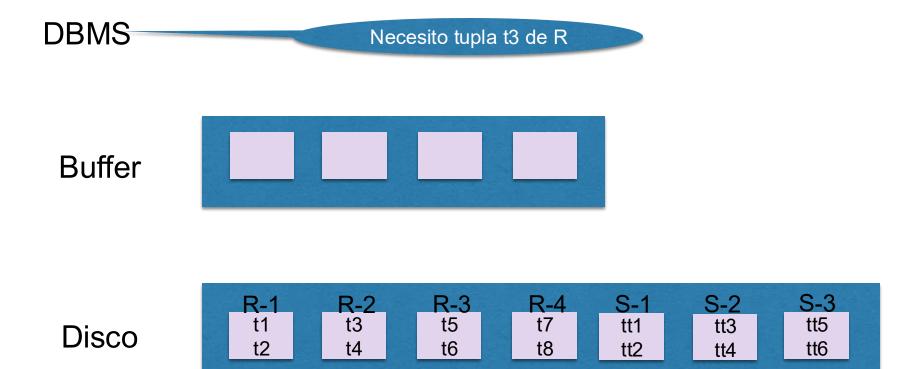
Relación S

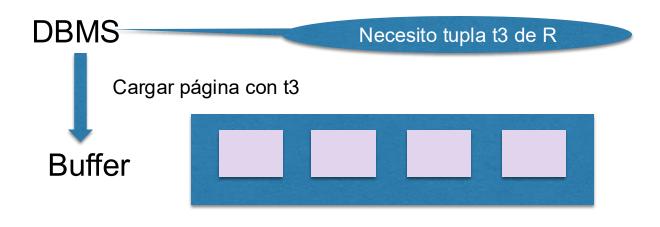


Buffer

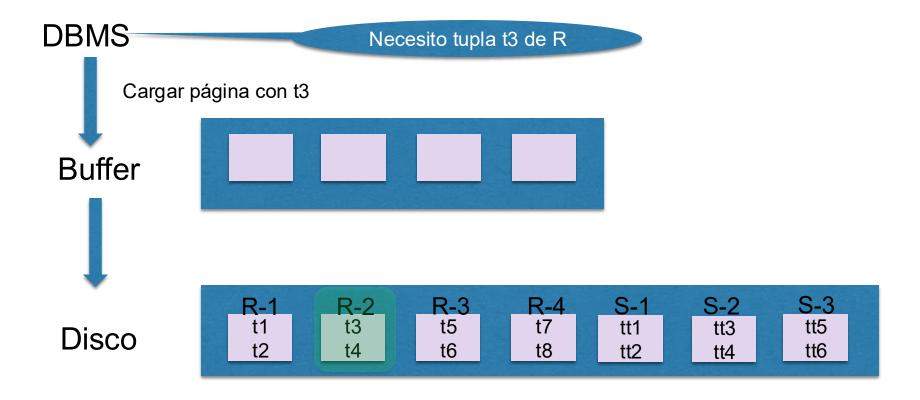
R-1	R-2	R-3	R-4	S-1	S-2	S-3
t1	t3	t5	t7	tt1	tt3	S-3 tt5 tt6
R-1 t1 t2	t4	t6	t8	tt1 tt2	tt4	tt6
Bears there				CONTRACTOR OF	WE STORY	

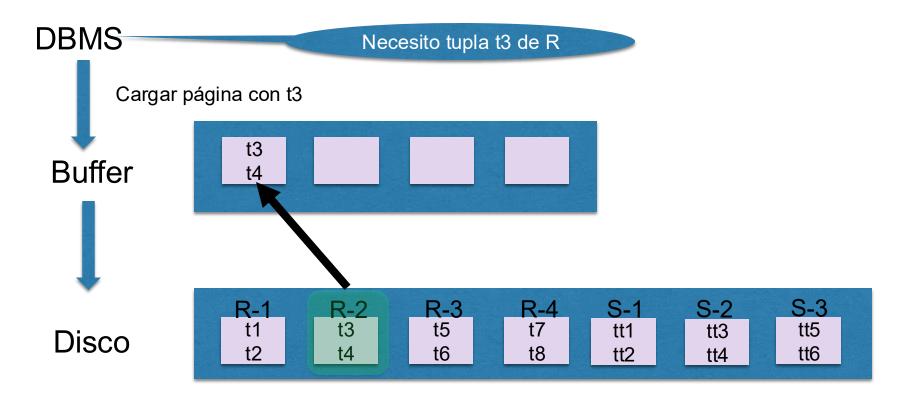






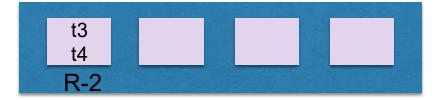




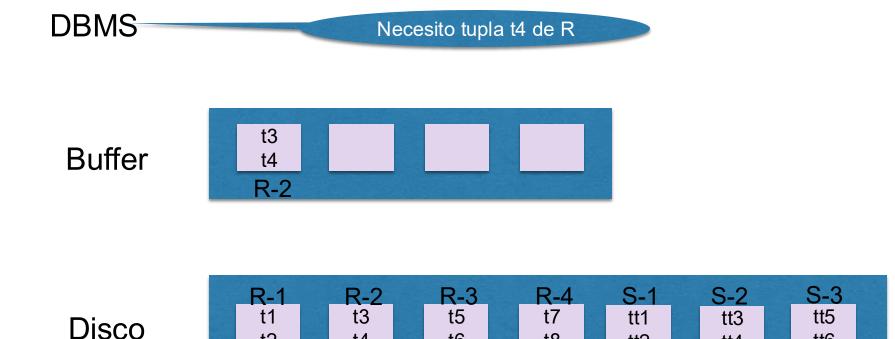


DBMS

Buffer



110	
tt3	tt5
tt4	tt5 tt6
	S-2 tt3 tt4



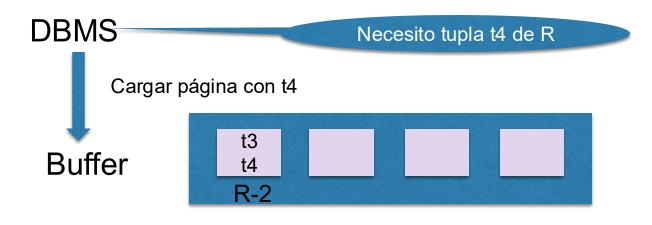
t6

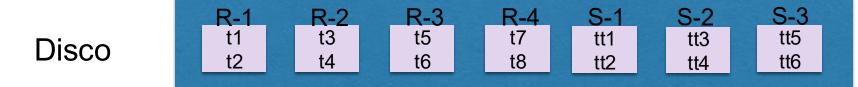
tt2

tt6

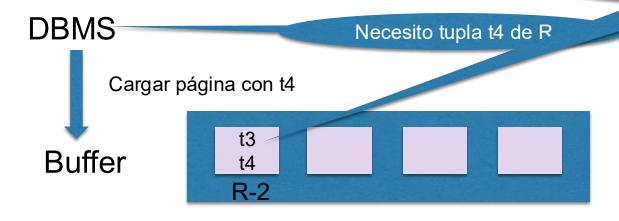
tt4

t4





Ya la tenemos!



R-1	R-2	R-3	R-4	S-1	S-2		S-3
t1	R-2 t3	R-3 t5 t6	t7	tt1	tt3		S-3 tt5 tt6
t2	t4	t6	t8	tt2	tt4		tt6
THE PROPERTY OF	Novince of the last	CONTRACTOR AND ADDRESS OF	THE RESERVE AND ADDRESS OF THE PARTY OF THE	NAME OF TAXABLE PARTY.	NO. 10-10-10-10-10-10-10-10-10-10-10-10-10-1	3	THE RESERVE

Algoritmos en Bases de Datos

Costo de un algoritmo

¿Cuántas veces tengo que leer una página desde el disco, o escribir una página al disco?

Las operaciones en buffer (RAM) son orden(es) de magnitud más rápidas que leer/escribir al disco – costo 0

Algoritmos en una BD

Los algoritmos implementan una interfaz de un iterador lineal:

- open()
- next()
- close()

Algoritmos en una BD

Para una relación R:

- R.open() se posiciona antes de la primera tupla de R
- R.next() devuelve la siguiente tupla o NULL
- R.close() cierra el iterador

DB

Buffer



R-	R-2	R-3	R-4		S-1	S-2		S-3
t1	R-2 t3 t4	t5	t7	ī	tt1	tt3		tt5 tt6
t2	t4	t6	t8		tt2	tt4	超	tt6

DBMS R.open()

Buffer

R-	R-2	R-3	R-4		S-1	S-2		S-3
t1	R-2 t3 t4	t5	t7	ī	tt1	tt3		tt5 tt6
t2	t4	t6	t8		tt2	tt4	超	tt6

DBMS R.open()

Buffer



R-1	R-2	R-3	R-4	S-1	S-2		S-3
t1	t3	t5	t7 t8	tt1	tt3		tt5
t2	t4	t6	t8	tt2	tt4	B	tt6
HERMIT	STATE OF THE PARTY.	OSMIN CENT	THE PERSON NAMED IN	THE PERSON NAMED IN	WIND CONTROL		THE RESIDENCE

DBMS R.next()

Buffer

t1 t2 R-1

R-1	R-2	R-3	R-4	S-1	S-2	S-3
t1	t3	t5	t7 t8	tt1	tt3	tt5 tt6
t2	t4	t6	t8	tt2	tt4	tt6
HERMING	NAME OF TAXABLE PARTY.	STATE OF THE PARTY	CHARLES NO.	NAME OF TAXABLE PARTY.	VOLUME TO SERVICE OF	CALL STREET

DBMS R.next()

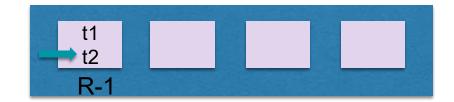
Buffer



R-1	R-2	R-3	R-4	S-1	S-2	S-3
t1	t3 t4	t5	t7 t8	tt1	tt3	tt5 tt6
t2	t4	t6	t8	tt2	tt3 tt4	tt6
HERMINGS				NAME OF TAXABLE PARTY.	NO. FOR THE PARTY OF THE PARTY	

DBMS R.next()

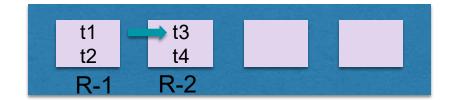
Buffer



R-1	R-2	R-3	R-4	S-1	S-2	S-3
t1	t3	t5	t7 t8	tt1	tt3	tt5 tt6
t2	t4	t6	t8	tt2	tt4	tt6
				CONTRACT OF	OF STREET	

DBMS R.next()

Buffer



R-1	R-2	R-3	R-4		S-1	S-2		S-3
t1	t3	t5	t7 t8		tt1	tt3		tt5 tt6
t2	t4	t6	t8	B.	tt2	tt4	B	tt6
evenued.	EN HORSE	CHARLES AND A SECOND	THE RESIDENCE		WINDS IN	NO PERSONAL PROPERTY.		-

DBMS R.close()

Buffer

t1 t3 t2 t4 R-1 R-2

R-1	R-2	R-3	R-4		S-1	S-2	S-3
t1	t3	t5	R-4 t7 t8		tt1	tt3	tt5
t2	t4	t6	t8	S.	tt2	tt4	tt6
MASSIMILARY					NAME OF THE	OF STREET	

DBMS

Buffer



R.open()

t:= R.next()
while t != null do
output t
t:= R.next()

R.close()



DBMS

Buffer



R.open()

R.close()



DBMS

Buffer



R.open()

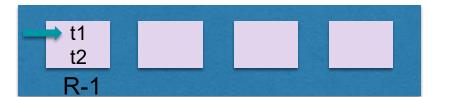
t:= R.next()
while t != null do
output t
t:= R.next()

R.close()



DBMS

Buffer



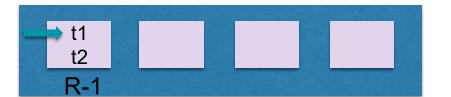
R.open()

R.close()



DBMS

Buffer



R.open()

t:= R.next()

while t != null do

output t

t:= R.next()

R.close()



DBMS

Buffer



R.open()

t:= R.next()
while t != null do
output t
t:= R.next()

R.close()



DBMS

Buffer



R.open()

t:= R.next()
while t != null do
output t
t:= R.next()

R.close()



DBMS

Buffer



R.open()

t:= R.next()

while t != null do

output t

t:= R.next()

R.close()



DBMS

Buffer



R.open()

t:= R.next()
while t != null do
output t
t:= R.next()

R.close()



DBMS

Buffer

R.open()

t:= R.next()
while t != null do
output t
t:= R.next()

R.close()



DBMS

Buffer



R.open()

t:= R.next()
while t != null do
output t
t:= R.next()

R.close()



DBMS

Buffer



R.open()

t:= R.next()
while t != null do
output t
t:= R.next()

R.close()



DBMS

Buffer



R.open()

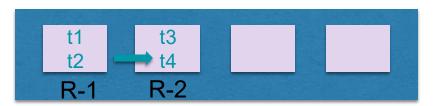
t:= R.next()
while t != null do
output t
t:= R.next()

R.close()



DBMS

Buffer



R.open()

t:= R.next()

while t != null do

output t

t:= R.next()

R.close()



DBMS

Buffer



R.open()

t:= R.next()
while t != null do
output t
t:= R.next()

R.close()



DBMS

Buffer



R.open()

t:= R.next()
while t != null do
output t
t:= R.next()

R.close()



En realidad

Cada operador de algebra relacional implementa interfaz de un iterador lineal:

- open()
- next()
- close()

Algoritmos – Operadores Lineales

El algoritmo de selección cambia dependiendo si es una consulta de igualdad (=) o de rango (<, >)

También depende si el atributo a seleccionar está indexado

Este implementa la interfaz de iterador lineal

Sin indice:

Si queremos hacer una selección sobre una tabla R

```
open()
   R.open()
next() // retorna el siguiente seleccionado
   t:=R.next()
   while t != null do
      if t satisface condición then
         return t
      t:=R.next()
   return null
```

Sin índice

Selección

Si queremos hacer una selección sobre una tabla R

```
open()
   R.open()
                                                 Para recorrer la selección Sel = \sigma_{cond} (R)
next() // retorna el siguiente seleccionado
                                                  Sel.open()
   t:= R.next()
   while t != null do
      if t satisface condición then
                                                 t := Sel.next()
          return t
                                                  while t != null do
      t:=R.next()
                                                            output t
   return null
                                                            t:= Sel.next()
close()
                                                  Sel.close()
   R.close()
```

Sin indice, **necesariamente** tenemos que recorrer todo **R** "FULL SCAN"

Con índice y consulta de igualdad:

Si queremos hacer una selección sobre una tabla **R** a un atributo indexado con un índice **I**

```
open()
   I.open()
   I.search(Atributo = valor)
next()
   t:= I.next()
   while t != null do
      return t
   return null
close()
   I.close()
```

Con índice y consulta de igualdad:

Sólo tenemos que leer las páginas que satisfacen la condición (más I/O si muchas tuplas satisfacen la condición)

Cambia un poco si el índice es Clustered o Unclustered (¿Por qué?)

Si el atributo es llave primaria entonces la operación prácticamente tiene I/O ~ 1

¿Cómo podemos hacer este tipo de consultas de forma eficiente?

Usando Proyección

Proyección

Algoritmo muy sencillo

```
open()
   R.open()
next()
   t:=R.next()
   while t != null do
      return project(t, atributos)
   return null
close()
   R.close()
```

Proyección

Necesariamente tenemos que recorrer todo R

Joins

Operación muy costosa

Supondremos solamente restricciones de igualdad (por ejemplo, R.a = S.a)

Queremos hacer un join entre **R** y **S**, cuando se satisface un predicado **p**

```
open()
    R.open()
    S.open()
    r:= R.next()

close()
    R.close()
    S.close()
```

Queremos hacer un join entre **R** y **S**, cuando se satisface un predicado **p**

```
next()
  while r != null do
    s:= S.next()
    if s == null then
        S.close()
        r:= R.next()
        S.open()
    else if (r, s) satisfacen p then
        return (r, s)
  return null
```

DB

Buffer

Join = $R \bowtie_p S$ Join.open() t:= Join.next() while t != null do output t t:= Join.next() Join.close()

R-1	R-2	R-3	R-4	S-1	S-2	S-3
R-1 t1 t2	t3	t5	R-4 t7 t8	tt1	tt3 tt4	tt5 tt6
t2	t4	t6	t8	tt2	tt4	tt6
				de la constitución de la constit		

DB R⋈_pS ...Join.open()

R.open() S.open() r:= R.next()

Buffer



S-2 S-3 R-3 S-1 R-4 tt5 t1 t3 t5 t7 tt1 tt3 Disco t6 t4 tt6 tt2 tt4

DB R⋈_pS ...Join.open()

R.open() S.open() r:= R.next()

Buffer





DB R⋈_pS ...Join.open()

R.open() S.open() r:= R.next()

Buffer





DB R⋈_pS ...Join.open()

R.open() S.open() r:= R.next()

Buffer





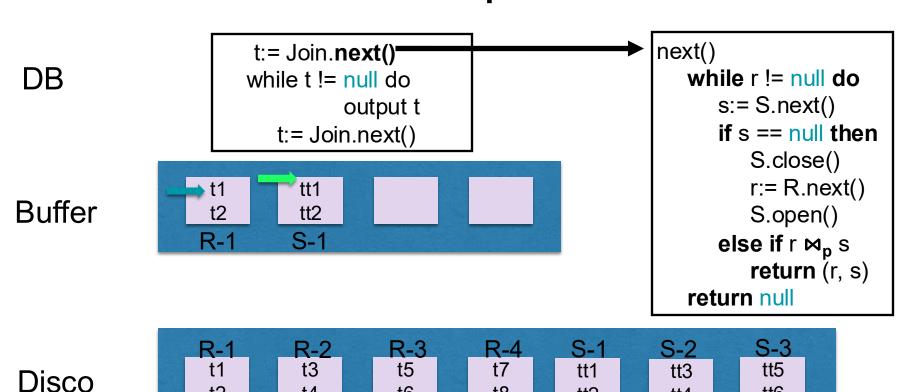
t:= Join.next()
while t != null do
output t
t:= Join.next()

Buffer

t:= Join.next()

t:= Join.next()

_		R-3 R-4		S-2	S-3
t1 t	3	t5 t7 t6 t8	tt1 tt2	tt3 tt4	tt5
t2 t	:4	t6 t8	tt2	tt4	tt6



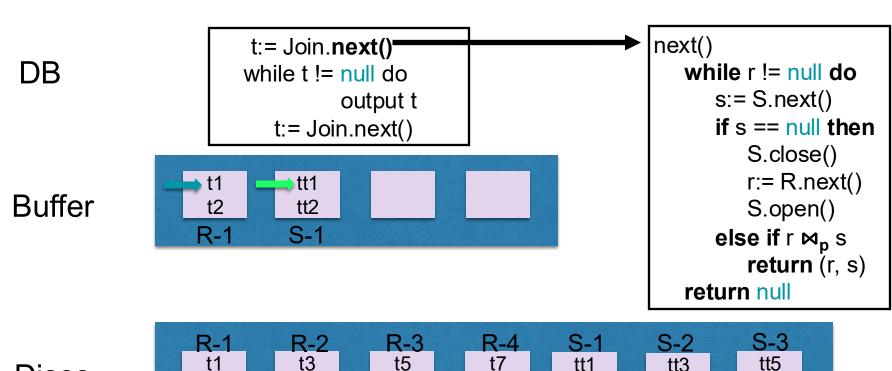
t8

tt2

tt6

tt4

t6



t8

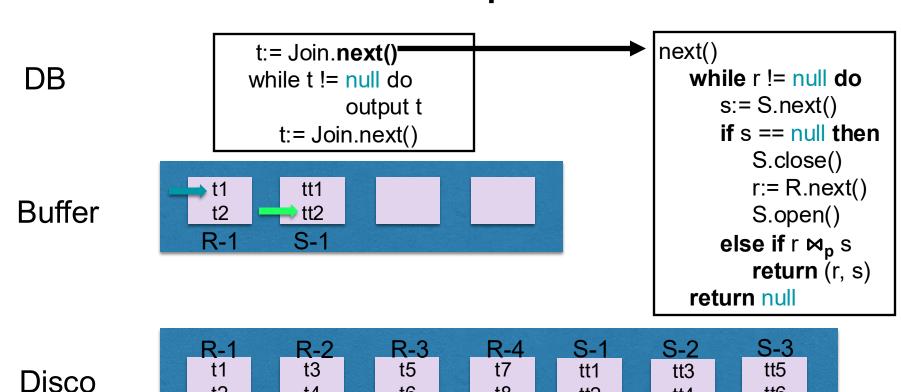
tt2

tt6

tt4

t6

t4



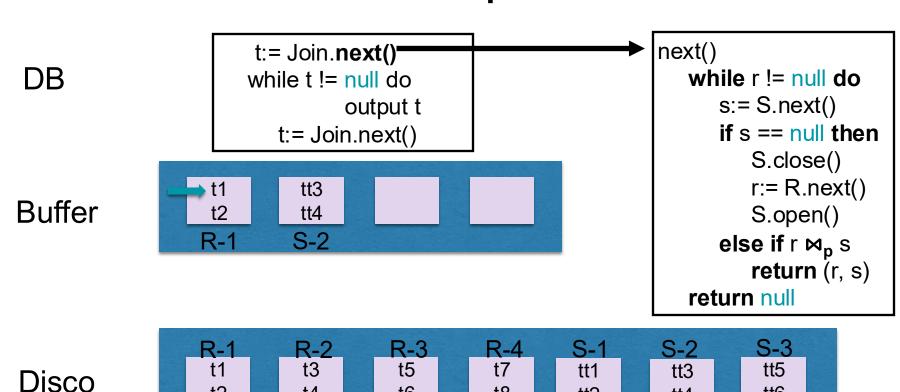
t8

tt2

tt6

tt4

t6



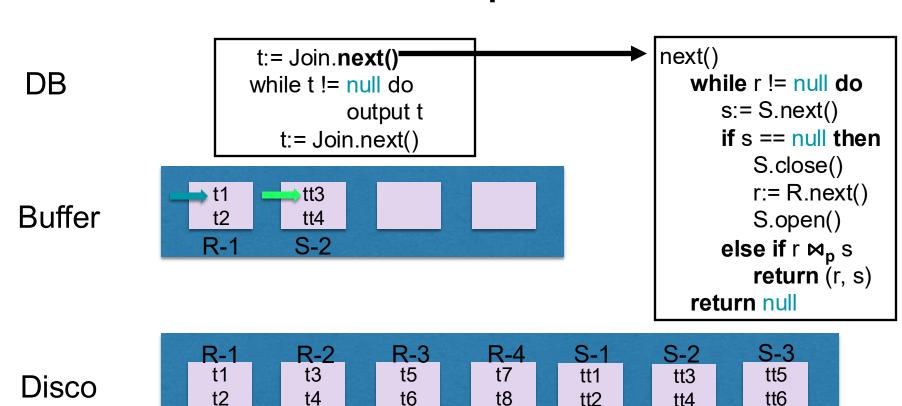
t8

tt2

tt6

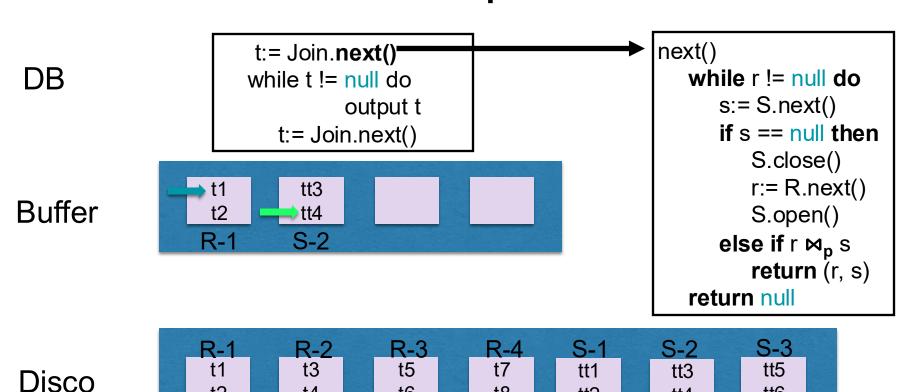
tt4

t6



tt2

tt4



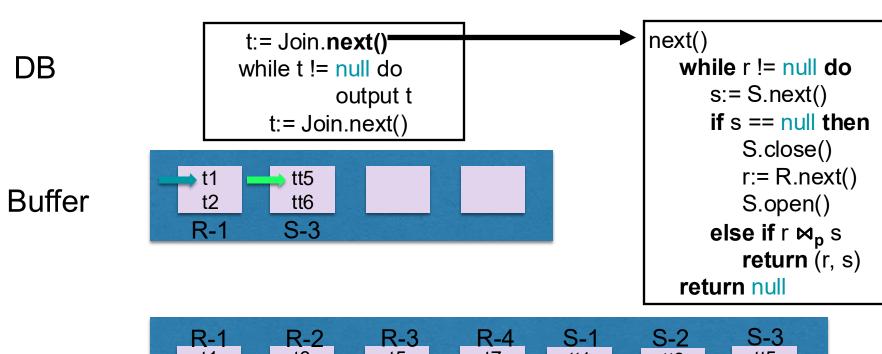
t8

tt2

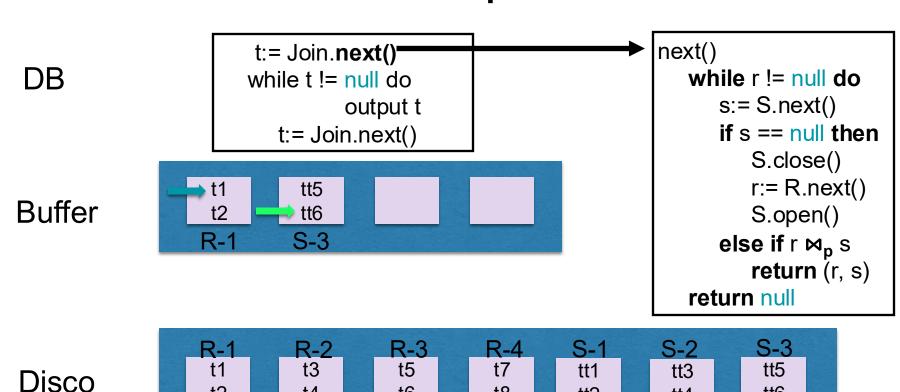
tt6

tt4

t6



R-1	R-2	R-3	R-4	S-1	S-2	S-3
t1	t3	t5	t7	tt1	tt3	tt5
t2	t4	t6	R-4 t7 t8	tt2	tt4	S-3 tt5 tt6
MARKET LANGE	April 100 person	CONTRACTOR	DESCRIPTION OF THE PERSON NAMED IN	THE PERSON NAMED IN	NO PERSONAL PROPERTY.	THE RESIDENCE



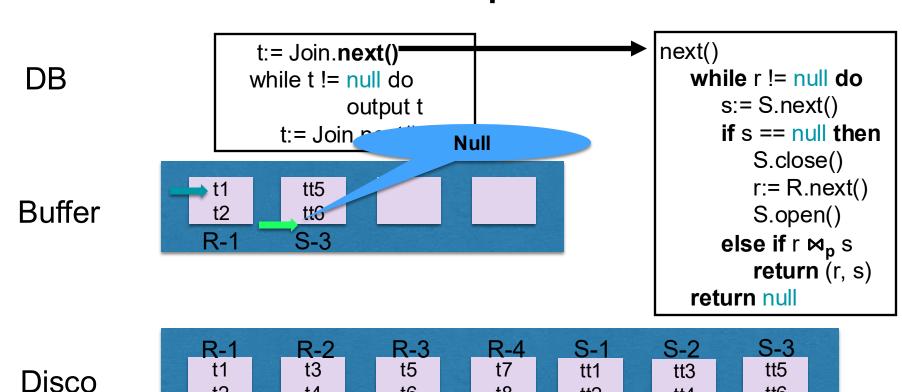
t8

tt2

tt6

tt4

t6



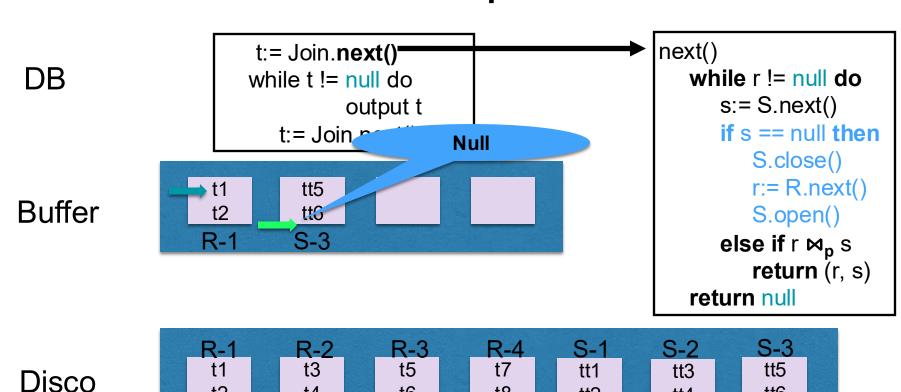
t8

tt2

tt6

tt4

t6



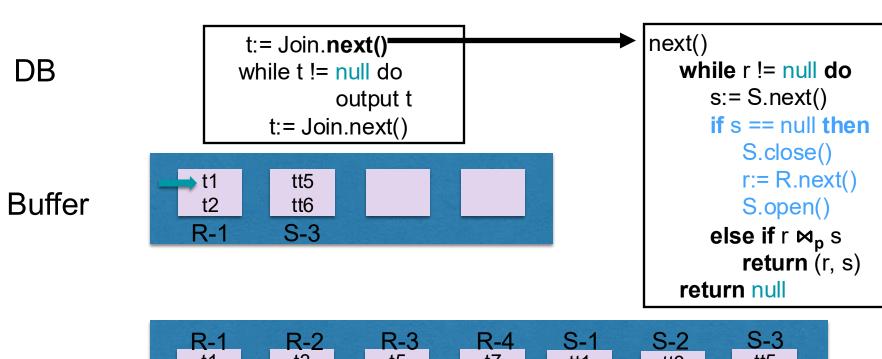
t8

tt2

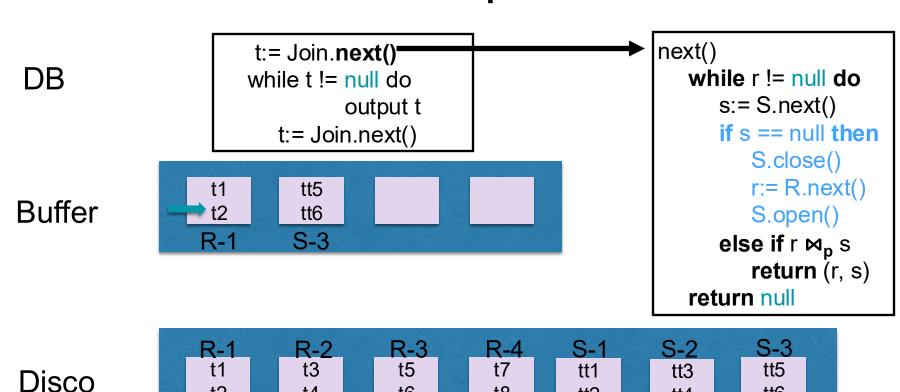
tt6

tt4

t6



R-1	R-2	R-3	R-4 t7 t8	S-1	S-2	S-3
t1	t3	t5	t7	tt1	tt3	tt5
t2	t4	t6	t8	tt2	tt4	S-3 tt5 tt6
E450 MG		Q5/18/68/	ILLEGES UND	NAME OF STREET	ON THE PARTY.	TARREST STREET



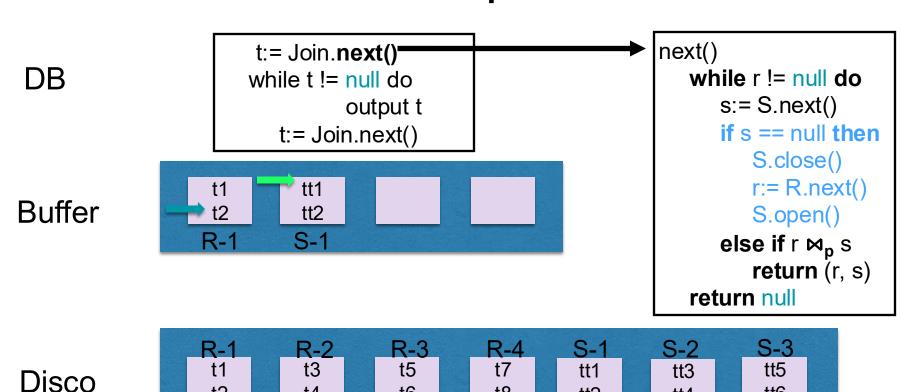
t8

tt2

tt6

tt4

t6



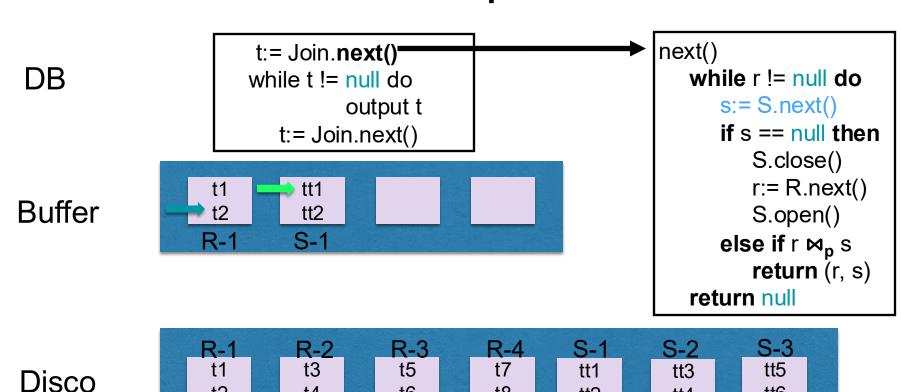
t8

tt2

tt6

tt4

t6



t8

tt2

tt6

tt4

t6

Es una implementación directa basada en un loop

Para cada tupla de **R** debemos leer **S** entera, aparte de leer **R** entera una vez

Costo en I/O es:

Costo(R) + Tuplas(R)·Costo(S)

Si **R** y **S** son tablas de 16 MB, cada página es de 8 KB y las tuplas son de 300 bytes

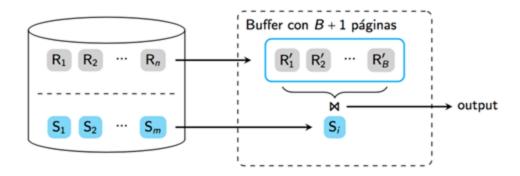
Cada relación tiene 2048 páginas y 55.000 tuplas aproximadamente

Costo de un I/O es 0.1 ms, entonces el join tarda:

3.1 horas

¿Cómo podemos optimizar esto? Con Block Nested Loop Joins

Aprovechamos mejor el buffer



Queremos hacer un join entre **R** y **S**, cuando se satisface un predicado **p**

```
fillBuffer()
                                      Buff = empty
open()
                                      r:= R.next()
   R.open()
                                      while r != null do
   fillBuffer()
                                         Buff = Buff union r
                                         if Buff.isFull() then
close()
                                             break
   R.close()
                                         r:=R.next()
   S.close()
                                      S.open()
                                      S.next()
```

Queremos hacer un join entre **R** y **S**, cuando se satisface un predicado **p**

```
next()
   while Buff != empty do
      while s != null do
          r:= Buffer.next()
          if r == null then
             Buffer.reset()
             s:=S.next()
          else if (r,s) satisfacen p then
             return (r,s)
      fillBuffer()
   return null
```

DB R⋈_pS

Buffer



R-1	R-2	R-3	R-4		S-1	S-2	S-3
t1	t3	t5	t7		tt1	tt3	S-3 tt5 tt6
t2	t4	t6	t8	S.	tt2	tt4	tt6
- CONTRACT			THE PERSON NAMED IN		NAME OF TAXABLE PARTY.	NO FEBRUARY	THE RESERVE

DB R⋈_pS.open()

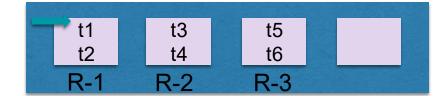
Buffer



R-1	R-2	R-3	R-4		S-1	S-2		S-3
t1	t3	t5	t7		tt1	tt3		S-3 tt5 tt6
t2	t4	t6	t8	Ø.	tt2	tt4	匮	tt6
					OF STREET			

DB R⋈_pS.open()

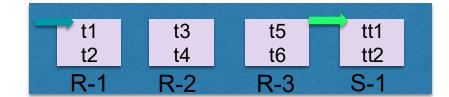
Buffer



R-1	R-2	R-3	R-4	S-1	S-2	S-3
t1	t3	t5	t7 t8	tt1	tt3 tt4	S-3 tt5 tt6
t2	t4	t6	t8	tt2	tt4	tt6
HERMING!	NAME OF TAXABLE PARTY.	CSYNVER!	DISCOURAGE STATE OF	NAME OF TAXABLE PARTY.	NO PERSONNELLE	CANTENDER

DB R⋈_pS.open()

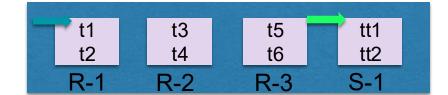
Buffer



R-	R-2	R-3	R-4	S-1	S-2	S-3
t1	R-2 t3 t4	t5	t7	tt1	tt3	tt5
t2	t4	t6	t8	tt2	tt4	tt6

DB while $(R \bowtie_p S.next())$

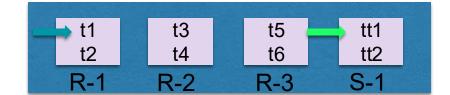
Buffer



R-1 R-2 R-3 R-4 S-1 S-2 S-1 t1 t3 t5 t7 tt1 tt3 tt5 t2 t4 t6 t8 tt2 tt4 tt6	R-1	-1 R-2	R-3	R-4	S-1	S-2	S-3
	t1	1 t3	t5	t7	tt1	tt3	tt5
t2 t4 t6 t8 tt2 tt4 tt6	t2	2 t4	t6	t8	tt2	tt4	tt6

DB while $(R \bowtie_p S.next())$

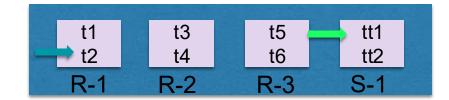
Buffer



R-1	R-2	R-3	R-4	S-1	S-2	S-3
R-1 t1 t2	t3	t5	R-4 t7 t8	tt1	tt3	S-3 tt5 tt6
t2	t4	t6	t8	tt2	tt4	tt6
HERMAC	Marine Service	resimilari	DESCRIPTION OF THE PERSON NAMED IN	NAME OF TAXABLE PARTY.	VOLENCE SHOW	CAST STREET

DB while $(R \bowtie_p S.next())$

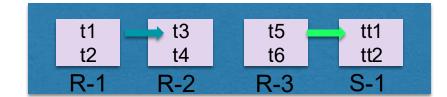
Buffer



R-1	R-2	R-3	R-4		S-1	S-2	S-3
t1	t3	R-3 t5 t6	R-4 t7 t8	툲	tt1	tt3	S-3 tt5 tt6
t2	t4	t6	t8	S.	tt2	tt4	tt6
					1200	MENT OF	

DB while $(R \bowtie_p S.next())$

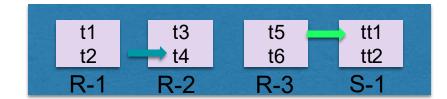
Buffer



R-1	R-2	R-3	R-4		S-1	S-2	S-3
t1	t3 t4	t5	t7	ħ,	tt1 tt2	tt3	tt5
t2	t4	t6	t8		tt2	tt4	tt6
MARINAGE PARTY	AD DESCRIPTION	450 P/ 600/	CHARLES IN		NAME OF THE PERSON NAMED IN	YOU DESCRIPTION OF	CONTRACTOR S

DB while $(R \bowtie_p S.next())$

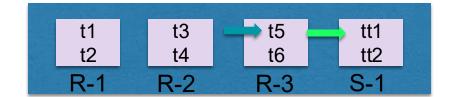
Buffer



R-1	R-2	R-3	R-4	S-1	S-2	S-3
t1	t3	t5	t7 t8	tt1	tt3	S-3 tt5 tt6
t2	t4	t6	t8	tt2	tt4	tt6
				NAME OF TAXABLE PARTY.	OF THE PERSON	

DB while $(R \bowtie_p S.next())$

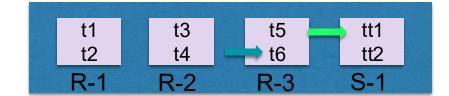
Buffer



R-1	R-2	R-3	R-4		S-1	S-2	S-3
t1	t3 t4	t5	t7	ħ,	tt1 tt2	tt3	tt5
t2	t4	t6	t8		tt2	tt4	tt6
MARINAGE PARTY	AD DESCRIPTION	450 P/ 600/	CHARLES IN		NAME OF THE PERSON NAMED IN	YOU DESCRIPTION OF	CONTRACTOR S

DB while $(R \bowtie_p S.next())$

Buffer



R-1	R-2	R-3	R-4	S-1	S-2	S-3
t1	t3	t5	t7	tt1	tt3	S-3 tt5 tt6
t2	t4	t6	t8	tt2	tt4	tt6
				1200	AND STREET	

DB while ($R\bowtie_p S.next()$)

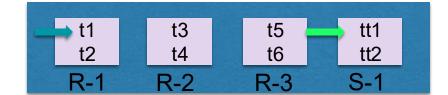
End of Buffer de R

Buffer t1 t3 t5 t1 t2 t4 t6 t2 R-1 R-2 R-3 S-1



DB while $(R \bowtie_p S.next())$

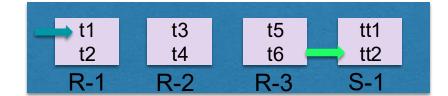
Buffer



R-1	R-2	R-3	R-4	S-1	S-2	S-3
t1	R-2 t3 t4	t5	t7	tt1	tt3	S-3 tt5 tt6
t2	t4	t6	t8	tt2	tt4	tt6
Market March	THE RESIDENCE		STREET, SQUARE,	NAME OF TAXABLE PARTY.	NO FEBRUARY	

DB while $(R \bowtie_p S.next())$

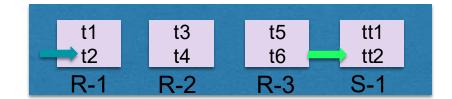
Buffer



R-1	R-2	R-3	R-4		S-1	S-2		S-3
t1	t3	t5	t7	5	tt1	tt3		S-3 tt5 tt6
t2	t4	t6	t8	S.	tt2	tt4	B	tt6
						**************************************		-110

DB while $(R \bowtie_p S.next())$

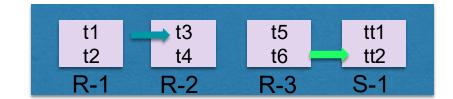
Buffer



R-1	R-2	R-3	R-4		S-1	S-2	S-3
t1	t3	t5	R-4 t7 t8	ħ,	tt1	tt3 tt4	tt5
t2	t4	t6	t8		tt2	tt4	tt6
l2	ι4	ιο	ιο		llZ	114	L

DB while $(R \bowtie_p S.next())$

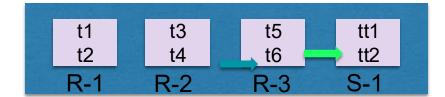
Buffer



R-1	150	R-2	R-3	R-4		S-1	S-2	S-3
t1		R-2 t3 t4	t5	t7	7	tt1	tt3 tt4	S-3 tt5 tt6
t2		t4	t6	t8		tt2	tt4	tt6
esemped.		STATE OF THE PARTY.	CONTRACTOR OF THE PARTY OF THE	AND DESCRIPTION	-	THE RESERVE	WILLIAM CONTRACTOR	CAST CINED

DB while $(R \bowtie_p S.next())$

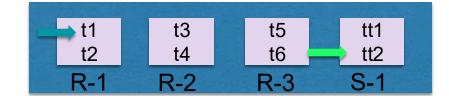
Buffer



S-3	S-2	S-1	R-4	R-3	R-2	R-1
tt5	tt3 tt4	tt1 tt2	t7	t5	t3 t4	t1 t2
tt6	tt4	tt2	t8	t6	t4	t2
	tt4	tt2	t8	t6	t4	t2

DB while $(R \bowtie_p S.next())$

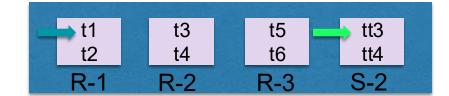
Buffer



R-1	R-2	R-3	R-4		S-1	S-2	S-3
t1	t3	t5	R-4 t7 t8		tt1	tt3	tt5
t2	t4	t6	t8	S.	tt2	tt4	tt6
MASSIMILARY					NAME OF THE	OF STREET	

DB while $(R \bowtie_p S.next())$

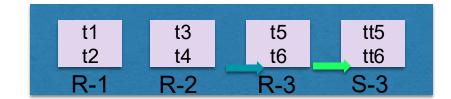
Buffer



R-1	R-2	R-3	R-4	S-1		S-3
t1 t2	t3 t4	R-3 t5 t6	t7 t8	tt1	tt3 tt4	S-3 tt5 tt6
t2	t4	t6	t8	tt2	tt4	tt6
				STATE OF	ON PURSON	

DB while $(R \bowtie_p S.next())$

Buffer



R-1	R-2	R-3	R-4	S-1	S-2	S-3
R-1 t1 t2	t3	t5	R-4 t7 t8	tt1	tt3	S-3 tt5 tt6
t2	t4	t6	t8	tt2	tt4	tt6
HERMAC	Marine Service	resimilari	ATTENDED TO	NAME OF TAXABLE PARTY.	VOLENCE SHOW	CAMP STREET

DB while $(R \bowtie_p S.next())$

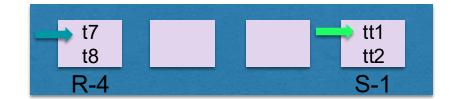
Buffer



R-1	R-2	R-3	R-4		S-1	S-2	S-3
t1	t3	R-3 t5 t6	R-4 t7 t8	툲	tt1	tt3	S-3 tt5 tt6
t2	t4	t6	t8	S.	tt2	tt4	tt6
					1200	MENT OF	

DB while $(R \bowtie_p S.next())$

Buffer



R-1	R-2	R-3	R-4	S-1	S-2	S-3	
t1	t3	t5	R-4 t7 t8	tt1	tt3	tt5	
t2	t4	t6	t8	tt2	tt4	tt6	
				OF STREET	METERS.		

DB while $(R \bowtie_p S.next())$

Buffer



R-1	R-2	R-3	R-4	S-1	S-2	S-3	
t1	t3	t5	R-4 t7 t8	tt1	tt3	tt5 tt6	
t2	t4	t6	t8	tt2	tt4	tt6	
				CONTRACTOR OF	AND STREET		

Block Nested Loop Join

Ahora cargamos muchas páginas de R a buffer

Por cada vez que llenamos el buffer recorremos a **S** entera una vez

Costo en I/O es:

Costo(**R**) + (Páginas(**R**)/(Buffer-2))·Costo(**S**)

Block Nested Loop Join

Si **R** y **S** son tablas de 16 MB, cada página es de 8 KB con un **buffer** de 1 MB

Cada relación tiene 2048 páginas y en **buffer** caben 128 páginas

Costo de un I/O es 0.1 ms, entonces el join tarda:

3.4 segundos

Block Nested Loop Join

Sin embargo existen algoritmos muchos más eficientes

Estos algoritmos se basan en Hashing o en Sorting

Además hacen usos de índices, como por ejemplo el B+ Tree

Sorting

Los algoritmos de sorting son conocidos en programación

¿Pero qué estudiarlos otra vez?

Para ordenar tuplas que exceden por mucho el tamaño de la memoria RAM

En los DBMS, se utiliza el algoritmo External Merge Sort

Pero antes, veamos Merge y MergeSort

Merge y MergeSort

Mezcla (merge) de secuencias ordenadas

Proponemos el siguiente algoritmo para combinar dos secuencias ordenadas para formar una nueva **ordenada**

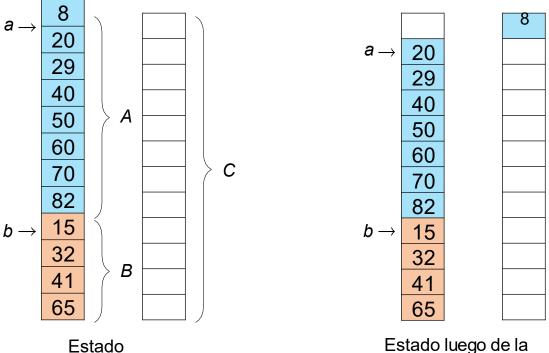
input: Secuencias ordenadas A y B

output: Nueva secuencia ordenada C

Merge(A, B):

- Iniciamos C vacía
- Sean a y b los primeros elementos de A y B
- Extraer de su secuencia respectiva el menor entre a y b
- Insertar el elemento extraído al final de C
- 5 Si quedan elementos en A y B, volver a la línea 2
- 6 Concatenar C con la secuencia que aún tenga elementos
- 7 return C

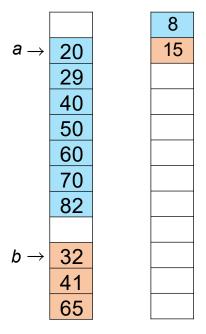
Merge: Ejemplo de ejecución



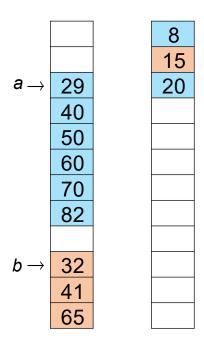
inicial

Estado luego de la primera iteración

Merge: Ejemplo de ejecución

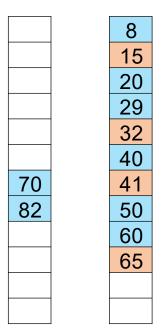


Estado luego de la segunda iteración

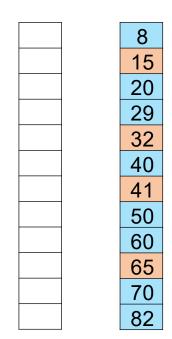


Estado luego de la tercera iteración

Merge: Ejemplo de ejecución



Estado luego de insertar en *C* el útimo elemento de *B*



Estado luego de concatenar el resto de *A*

Dividir para conquistar

El plan para usar Merge en un algoritmo de ordenación sigue la estrategia dividir para conquistar

La estrategia sigue los siguientes pasos:

- 1. Dividir el problema original en dos (o m'as) sub-problemas del mismo tipo
- 2. Resolver recursivamente cada sub-problema
- 3. Encontrar solución al problema original **combinando** las soluciones a los sub-problemas

Los sub-problemas son instancias más pequeñas del problema a resolver

Dividir para conquistar y Merge

Podemos usar la estrategia dividir para conquistar en el problema de ordenación, usando Merge

¿En qué parte del dividir para conquistar usaremos Merge?

La idea general para ordenar usando Mergedefine un nuevo algoritmo que llamaremos MergeSort

- 1. Dividir la secuencia original en dos sub-secuencias
- 2. Llamamos recursivamente a MergeSort sobre las dos sub-secuencias
- 3. Combinamos las secuencias ordenadas resultantes mediante Merge

El algoritmo MergeSort

A continuación, tenemos el pseudocodigo del algoritmo recursivo MergeSort

```
input: Secuencia A
output: Secuencia ordenada B

MergeSort (A):

if |A| = 1: return A

Dividir A en mitades A_1 y A_2

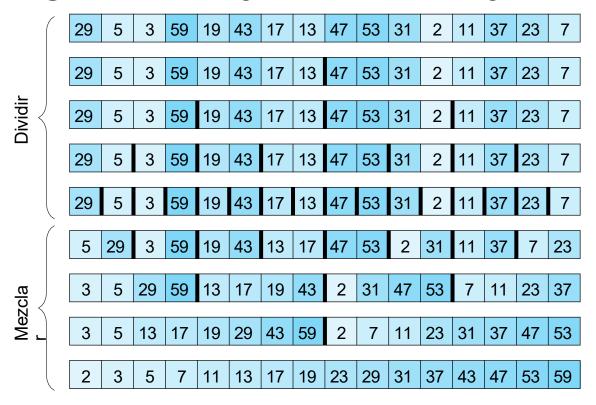
B_1 \leftarrow \text{MergeSort}(A_1)

B_2 \leftarrow \text{MergeSort}(A_2)

B \leftarrow \text{Merge}(B_1, B_2)

return B
```

MergeSort: Ejemplo de ejecución



En los DBMS, se utiliza el algoritmo External Merge Sort

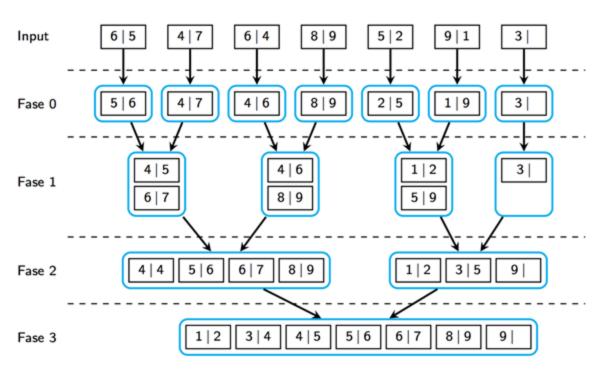
Hablaremos de **Run** como una secuencia de páginas que contiene una conjunto ordenado de tuplas

Algoritmo funciona por fases

Fase 0: creamos los runs iniciales

Fase i:

- Traemos los runs a memoria
- Hacemos el merge de cada par de runs
- Almacenamos el nuevo run a disco (i.e. materializamos resultados intermedios)



Recursos para estudiar

<u>Database Management Systems, 3rd edition, de Raghu Ramakrishnan y</u> <u>Johannes Gehrke.</u> Part IV