
Costos

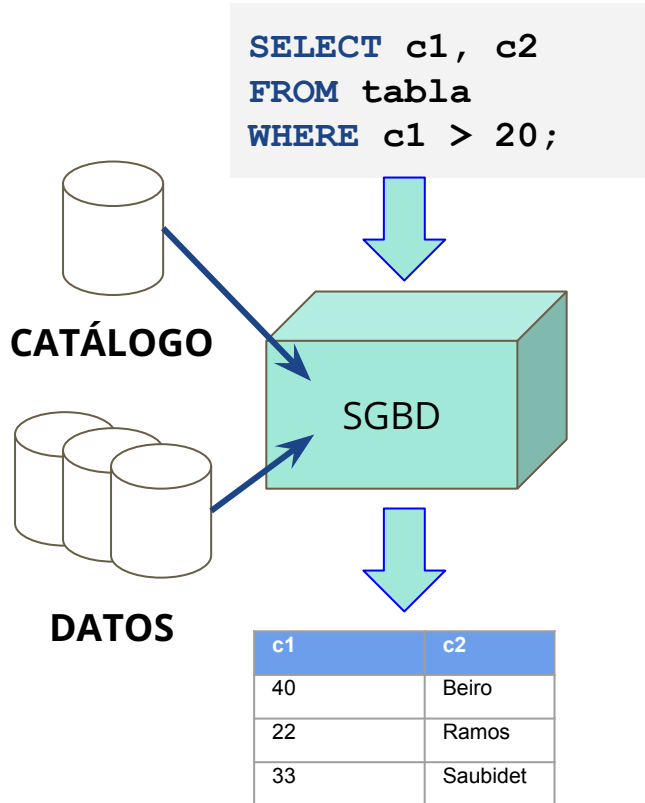
— Resolución de consultas —



Introducción

Esquema de procesamiento

Esquema de procesamiento de una consulta



1. Parser y Translator
 - Rechazar consulta inválida
2. Optimizador
 - Conversión a expresión de A.R.
 - Expresión equivalente de A.R.
 - Estrategia para cada operador
3. Evaluación del plan de ejecución
 - Devolver el resultado en base a los datos

Planes de consulta y de ejecución

- La optimización de una consulta inicia con una expresión de álgebra relacional equivalente a la consulta SQL
- Mediante heurísticas la expresión se convierte en una expresión equivalente, obteniéndose un **plan de consulta**
- En base a datos sobre los datos de la base (metadatos) que se encuentran en el catálogo, se elige cómo resolver cada operador de álgebra relacional, obteniéndose el **plan de ejecución**
 - Estructuras de datos, índices y algoritmos a usar

Minimización de costos

- Se busca que el costo de resolver la consulta sea el menor posible pero hay distintos costos
 - Costo de acceso a disco (lectura o escritura)
 - Costo de procesamiento
 - Costo de uso de memoria
 - Costo de uso de redes
- Nos enfocaremos únicamente en los costos de acceso a disco, pero en ciertas bases los otros costos pueden tener una influencia importante

Reglas de equivalencia

Planes de consulta y de ejecución

- Al convertir una consulta del lenguaje SQL a álgebra relacional, se pasa de un lenguaje declarativo a uno procedural
- Si existen varias expresiones de álgebra relacional equivalentes a la consulta SQL, es probable que sus costos no sean iguales
- Interesa poder encontrar consultas equivalentes pero que nos den la mejor performance
 - Revisar todas las consultas equivalentes es costoso, por lo que se usa heurísticas que suelen funcionar bien en la mayoría de los casos

Equivalencia en selección y proyección

- Equivalencia en cascada y conmutatividad
- Selección:
 - $\sigma_{c1 \wedge c2 \wedge \dots \wedge cn} (R) = \sigma_{c1}(\sigma_{c2}(\dots(\sigma_{cn}(R))\dots))$
 - $\sigma_{c1 \vee c2 \vee \dots \vee cn} (R) = \sigma_{c1} (R) \cup \sigma_{c2} (R) \cup \dots \cup \sigma_{cn} (R)$
 - $\sigma_{c1}(\sigma_{c2} (R)) = \sigma_{c2}(\sigma_{c1} (R))$
- Proyección
 - $\pi_{X1}(\pi_{X2}(\dots(\pi_{Xn}(R))\dots)) = \pi_{X1}(R)$
 - $\pi_X(\sigma_{\text{cond}}(R)) = \sigma_{\text{cond}}(\pi_X(R))$
 - Sólo si la condición usa únicamente atributos en X

Equiv. en Prod. Cartesiano, Junta y Op. de Conjuntos

- Conmutatividad y Asociatividad
- Producto cartesiano y Junta
 - $R \times S = S \times R$ y $R * S = S * R$
 - $(R \times S) \times T = R \times (S \times T)$
 - $(R * S) * T = R * (S * T)$
- Operaciones de conjuntos
 - $R \cup S = S \cup R$ y $R \cap S = S \cap R$
 - $(R \cup S) \cup T = R \cup (S \cup T)$
 - $(R \cap S) \cap T = R \cap (S \cap T)$

Equivalencia entre junta y producto cartesiano

- Hacer una junta es equivalente a realizar el producto cartesiano entre las tablas y luego revisar la condición de junta con una selección
 - $\sigma_c(R \bowtie S) = R \bowtie_c S$

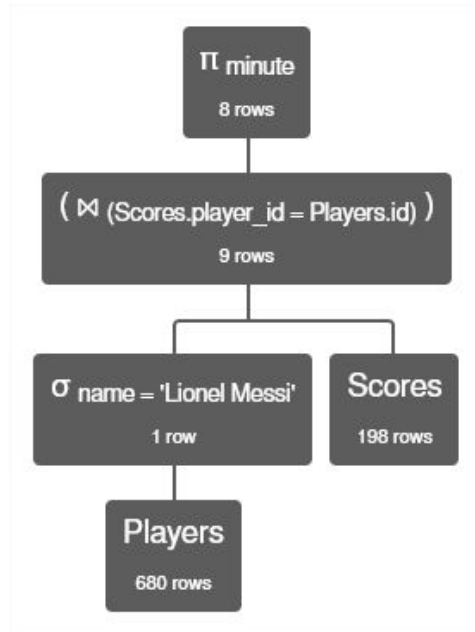
Otras equivalencias

- Distribución de la selección en la junta
 - $\sigma_c(R * S) = \sigma_{cR}(R) * \sigma_{cS}(S)$
 - Si la condición c puede escribirse como $cR \wedge cS$ donde cR y cS involucran atributos sólo de R y S respectivamente
- Distribución de la proyección en la junta
 - $\pi_X(R * S) = \pi_{XR}(R) * \pi_{XS}(S)$
 - Si todos los atributos de junta están contenidos en X , se construyen XR y XS como conjuntos que tienen los atributos de X que están en R o S respectivamente

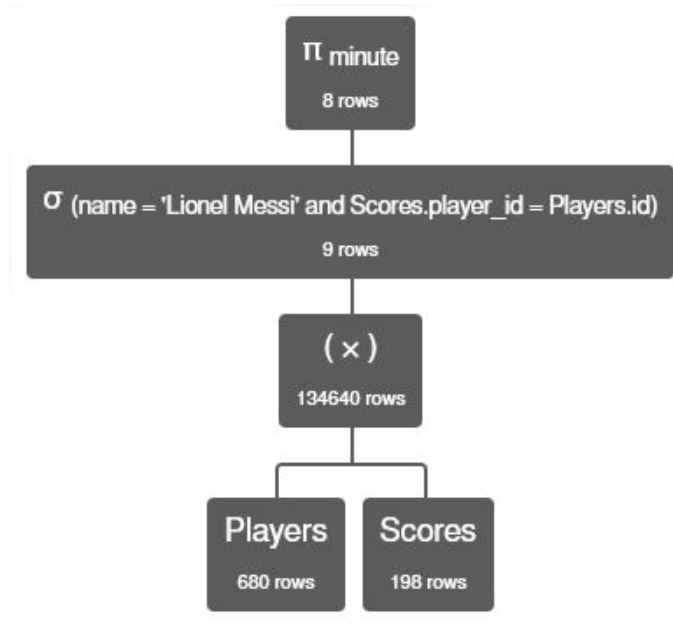
Heurísticas de optimización

Otras equivalencias

- Dos consultas equivalentes pueden tener costos distintos



VS

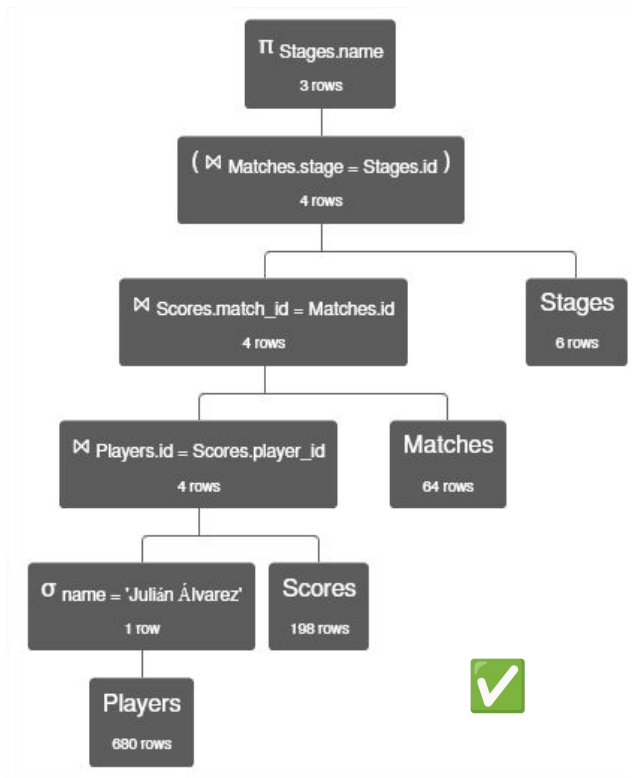


¿En qué minutos hizo lionel messi goles en el mundial?

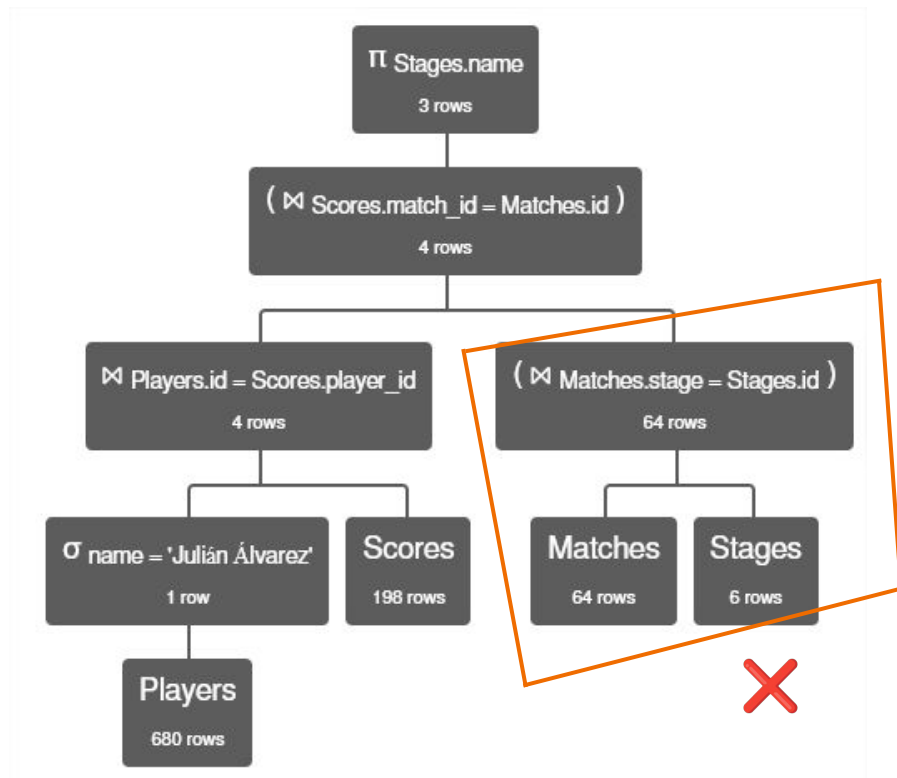
Otras equivalencias

- Determinar todos los árboles equivalentes según las reglas vistas es muy costoso
- Se utilizan heurísticas para agilizar el proceso
 - El objetivo general es reducir el tamaño de relaciones intermedias
- Para aprovechar **pipelining** se prefieren árboles “left deep” en el que los hijos derechos siempre son accesos a tablas

Árboles left deep



VS

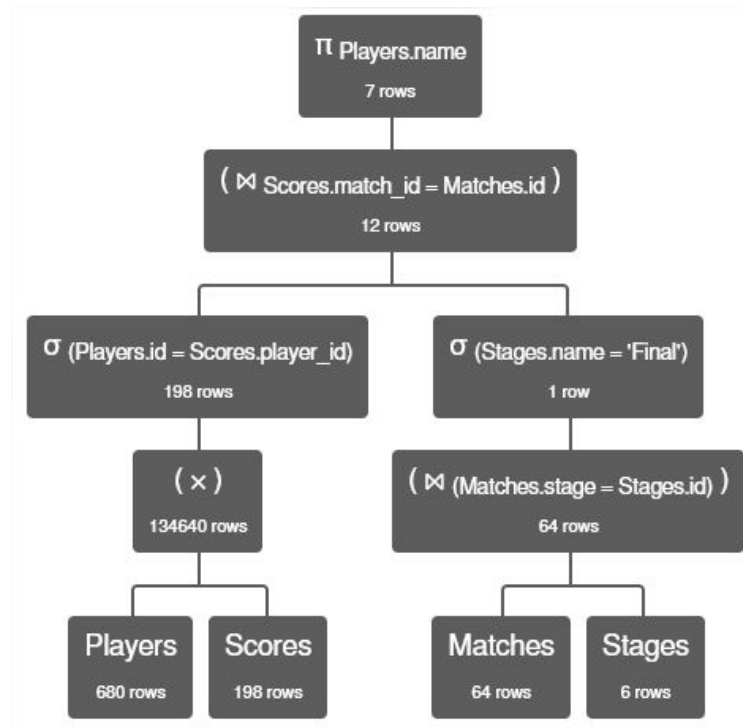


Reglas generales

- Realizar las selecciones lo antes posible
- Reemplazar productos cartesianos por juntas cuando se pueda
- Proyectar para descartar atributos no utilizados lo antes posible
 - Priorizar selección antes que proyección
 - Tener cuidado si el árbol no queda left deep (evaluar pipelining)
- Realizar las juntas más restrictivas primero

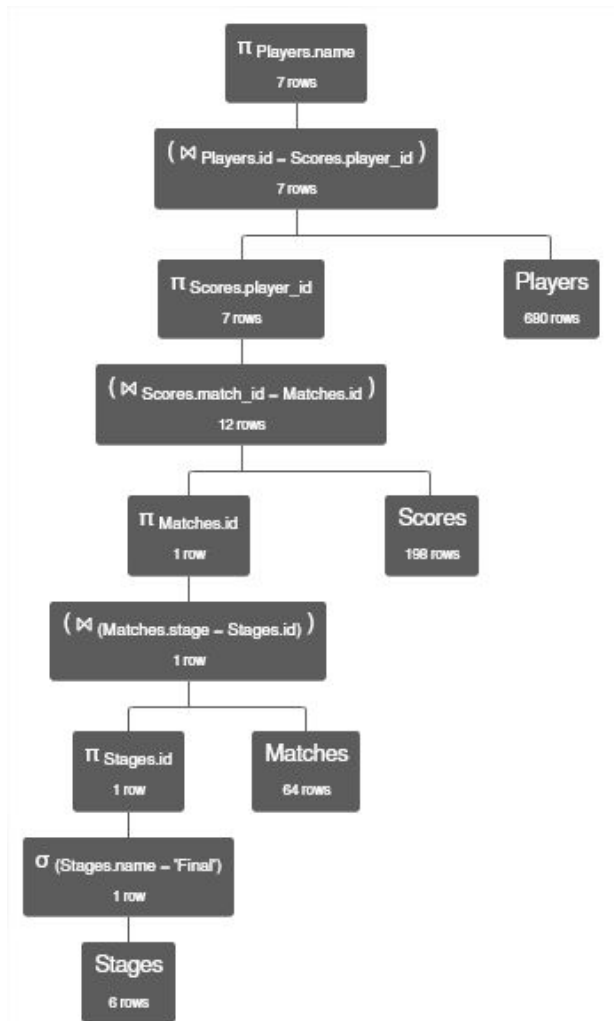
Optimización de consultas - Ejemplo

- Optimice el plan del siguiente árbol de consulta que obtiene el nombre de los jugadores que hicieron gol en la final



Optimización de consultas

- Ejemplo resuelto



Metadatos

Datos acerca de los datos

Resolución de cada operador algebraico

- Una vez elegido el árbol de consulta a utilizar, se debe elegir de qué modo resolver cada operador algebraico
- Existen distintos algoritmos para cada operador cuyo costo depende de muchos factores
 - Cantidad de datos
 - Tamaño de datos
 - Existencia de índices
 - Memoria disponible

Estimación del costo

- Para poder elegir un método sobre otro, el SGBD estima cuánto le costará resolver la consulta con cada método y elige el de menor costo estimado
 - La estimación tiene un costo mucho menor a ejecutar la consulta
- Para esto, tiene información sobre los datos de las tablas
 - Guarda estos metadatos en el catálogo
 - PostgreSQL usa la tabla `pg_statistics` y su vista `pg_stats`

Catálogo - Información disponible

- Utilizaremos la siguiente notación para la información del catálogo que se posee sobre cada tabla:
 - **$n(R)$** : cantidad de filas de la tabla R
 - **$B(R)$** : cantidad de bloques que ocupa la tabla R
 - **$F(R)$** : (factor de bloque) cantidad de filas por bloque en la tabla R. Se puede calcular como $n(R) / B(R)$
 - **$V(A, R)$** : (variabilidad de A en R) cantidad de distintos valores que tiene el atributo A en la tabla R
- También información sobre índices
 - **$Height(I(A,R))$** : Altura del índice, para índices de tipo árbol

Catálogo - costo de actualización

- Mantener actualizado el catálogo ante cada operación de ABM puede ser muy costoso
- Los motores suelen hacerlo con cierta periodicidad, o cuando están ociosos, o cuando el usuario lo indique explícitamente
- A veces incluso actualizan la información sobre un muestreo de los datos y no sobre todos ellos
- Igualmente valores aproximados son útiles para hacer buenas estimaciones de costos

Índices

Índices

- Análogos a los índices (table of content) de los libros

C	
C, lenguaje de programación	
desajuste de impedancia y, 253	
PHP, utilizando, 788	
SQL, incrustado y, 254–257	
SQL/CLI, utilizando, 266–268	
C++, lenguaje de programación	
almacenamiento persistente y, 17	
estructura de base de datos y, 9–10	
SQL con, 204	
vinculación, 645–647	
cabeza de lectura/escritura, 394	
cadena de caracteres, tipos de datos, 207–208, 219	
cadenas en PHP, 790–791	
cálculo relacional	
cuantificador existencial en el, 171–173	
cuantificador universal en el, 171, 175–176	
de dominio, 177–179	
de tupla, 169–171, 204	
	caracteres separadores, 401
	cartesiano, producto, 126, 152–155, 230
	cartuchos, 671, 705
	CASE (ingeniería de software asistida por computador)
	DDL, utilizando, 364
	diseño de bases de datos, utilizando, 39
	notación, 901–903
	panorámica de, 378–381
	recopilación de requisitos y análisis, utilizando, 353
	catálogo
	DBMS, 9
	SQL, 205
	categoría
	cuándo usar, 104
	definición, 105
	mapeado, 199, 648
	modelado mediante, 100–101
	categoría parcial, 101
	categoría total, 101

- En un libro me dicen en qué página está cada tema
- En un SGBD en qué bloques están las filas con un cierto valor en una o más columnas

Índices - Generalidades

- Se puede indexar los datos de una tabla por una o más expresiones
- El índice, generalmente implementado como un árbol B+, guardará para cada posible valor de las expresiones, en qué bloque o bloques hay filas con ese valor
- Esto agiliza la búsqueda por esas expresiones
 - También búsquedas por rangos
 - En índices por varias expresiones, sólo agiliza en búsquedas que las usan en el orden definido

Índices - Mantenimiento

- Tener índices empeora la performance de los cambios en la base ya que hay un costo de mantenimiento de los mismos
 - En altas se agregan entradas a todos los índices
 - En bajas se quitan entradas en todos los índices
 - En modificaciones que cambien el valor de las expresiones de los índices, debe quitarse la entrada antigua y agregarse la nueva
- Además los índices ocupan espacio en disco
 - Menor al de la tabla pero debe considerarse

Índices - Árboles B+

- Los árboles B+ tienen ciertas reglas de actualización que les dan propiedades buenas para ser usados como índices
 - Salvo el nodo raíz, todos los nodos se encuentran con al menos un 50% de ocupación
 - Todos los nodos hoja tienen la misma altura
 - Operaciones de ABM intentan reorganizar lo menos posible los nodos del árbol

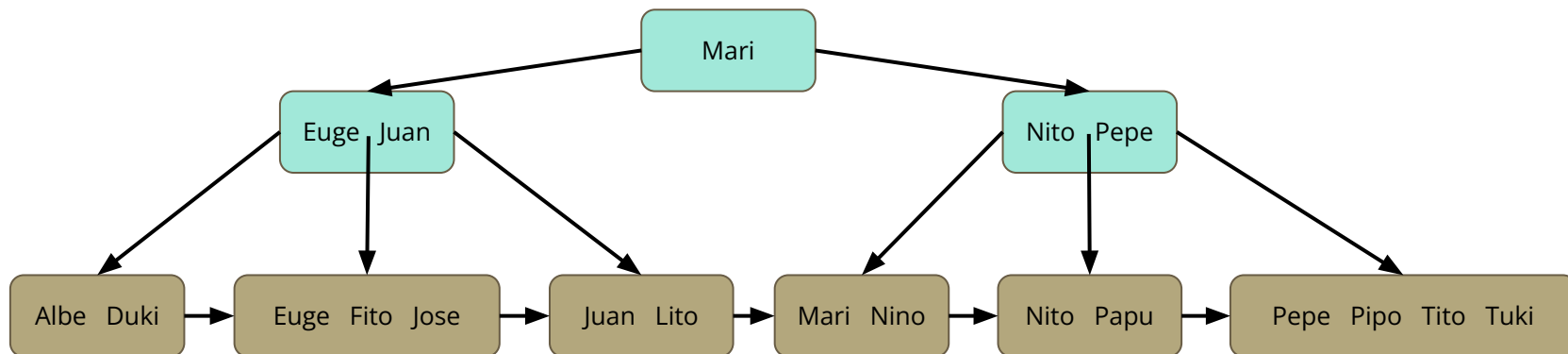
Índices - Árboles B+ - Tipos de nodo

- Tienen dos tipos de nodo, **nodos hoja** y **nodos internos**
- La cantidad de claves que entren por nodo dependen del tamaño del nodo y del tamaño las claves a agregar al árbol
- Cada nodo interno tiene k claves y $k + 1$ punteros
 - Los punteros apuntan a nodos de un nivel inferior
- Cada nodo hoja tiene k claves y $k + 1$ punteros
 - k punteros uno por cada clave
 - 1 puntero extra al siguiente nodo hoja en orden

Índices - Árboles B+ - Tipos de nodo

- Los árboles B+ guardan pares clave/valor
 - Al usar los árboles B+ como índice las **claves** a guardar son los valores de las expresiones indexadas y el **valor** a guardar son los bloques donde hay filas con esos valores
- Los nodos hoja son los que guardan la clave y el valor
 - Están conectados para facilitar búsquedas por rango
- Los nodos no hoja, en caso de existir, guardan algunas de las claves del árbol (no todas) con el fin de poder llegar fácilmente a los nodos que contengan una clave buscada

Índices - Árboles B+ - Ejemplo de búsqueda



- Buscar a "Papu" implica leer la raíz, ir al puntero de la derecha (Papu > Mari), luego al del medio (Nito < Papu < Pepe) y ahí se llegó al bloque que contiene "Papu"

Versión interactiva similar en <https://planetscale.com/blog/btrees-and-database-indexes>

Índices - Árboles B+ - Costo de búsqueda

- El costo de usar el índice para buscar un valor está dado por
 1. La **altura** del índice (este costo es igual para cualquier clave buscada)
 2. La **cantidad de bloques** en el que haya filas con el valor buscado (este costo depende del valor buscado)
 - En índices por claves candidatas este costo extra es 1

Índices - Clustering

- Algunos motores tienen la opción de hacer índices de clustering, en los cuales los datos **físicamente** están **ordenados** por la clave del índice
- Esto permite reducir el costo de acceso
 - Si un valor del índice está en 5 filas y entran 10 filas por bloque, esas 5 filas casi seguro estarán en un único bloque si el índice es de clustering
 - Si no es de clustering, probablemente estarán en 5 bloques distintos

Índices - Clustering

- El costo de mantener un índice de clustering es mayor ya que se debe estar reestructurando no solo el índice sino también el archivo con los datos ante ABMs
- Sólo puede haber un índice de clustering por tabla
- El índice de clustering también agiliza consultas por rangos
- PostgreSQL no tiene índices de clustering pero tiene el comando **CLUSTER** para ordenar físicamente una tabla en base a un índice (posteriores modificaciones pueden hacer perder el orden)

Costos de Operadores

Costos de Operadores - Generalidades

- Analizaremos primero el costo de resolver cada operador por separado
- No consideraremos el costo de almacenar el resultado final
 - No siempre se almacenan los resultados intermedios
 - Los resultados finales pueden enviarse a quien hace la consulta sin almacenarse
 - Igualmente estimaremos los bloques que ocupa el resultado
- Luego veremos cómo combinar el costo de varios operadores

Costos de Operadores - Generalidades

- El costo lo mediremos en bloques de disco accedidos
 - Ya sea para leer o escribir
- El costo real puede diferir del estimado
 - Las estimaciones se hace en base a suposiciones que no coinciden 100% con la realidad
 - Las bases de datos mantienen cache de datos ya accedidos en memoria para ahorrarse lecturas
- En la mayoría de los casos igualmente el costo estimado permitirá elegir el método que será óptimo al ejecutarse

Costos de Operadores - Simplicidad

- Veremos algoritmos sencillos con el fin de comprender estrategias de resolución
- En muchos casos se pueden optimizar pero con una algoritmia más compleja
 - Preferimos obviar estas optimizaciones con el fin de la comprensión
- La estimación de costos es una fórmula matemática que se resuelve rápidamente, en mucho menos tiempo que ejecutar la consulta y saber realmente cuánto demora

Costos de Operadores

σ - Selección

Selección - File Scan

- La forma más sencilla de resolver una selección es recorrer bloque a bloque todos los datos de la tabla y evaluar fila a fila si se cumple o no la condición
- Esta estrategia se la conoce como **File Scan** o también como búsqueda lineal
- Se puede utilizar con cualquier tipo de condición
- El costo es $\text{Cost}(\sigma_c(R)) = B(R)$

Selección - Index Scan

- Cuando la condición es de la siguiente forma

$$\sigma_{A_i=v}(R)$$

- Y hay un índice por la columna A_i , puede convenir utilizar un índice para resolver la consulta (**Index Scan**)
 - Costo de acceder al índice por la clave de búsqueda v
 - Costo de encontrar los N bloques que, según el índice indica, contienen filas con $A_i = v$

Selección - Index Scan

- Si el índice no es de clustering:

$$\text{Cost}(\sigma_{A_i=v}(R)) = \text{Height}(I(A,R)) + \lceil n(R) / V(A,R) \rceil$$

- Asumimos que las filas a devolver están en distintos bloques y distribución uniforme de los valores de A
- Si el índice es de clustering, entonces las filas van a estar en bloques contiguos:



$$\text{Cost}(\sigma_{A_i=v}(R)) = \text{Height}(I(A,R)) + \lceil n(R) / (V(A,R) * F(R)) \rceil$$

$$\text{Equivalente a: } \text{Cost}(\sigma_{A_i=v}(R)) = \text{Height}(I(A,R)) + \lceil B(R) / V(A,R) \rceil$$

Selección - Index Scan

- Si la condición no es por igualdad, puede extenderse la fórmula anterior pero en general el costo es mayor, salvo rangos acotados

$$\sigma_{A_i \geq v_1 \wedge A_i \leq v_2}(R)$$

- El índice a aprovecharse puede tener más atributos que A_i (índice compuesto), pero debe comenzar con ellos
 - Índice por A_i, A_j, A_k : Funciona 
 - Índice por A_j, A_i : No sirve 

Selección - Conjunción de condiciones

- Si hay una conjunción de condiciones, igualmente puede aprovecharse un índice por alguna de las columnas

$$\sigma_{A_i=v1 \wedge A_j=v2}(R)$$

- Se usa el índice por el atributo indexado (asumamos A1)
- Se accede a las filas con $A_i = v1$ y se revisa previo a devolverlas que también cumplan que $A_j = v2$
- También se podría haber aprovechado un índice compuesto que comience con A_i y A_j
- Si hay dos índices distintos por cada atributo, se usan ambos y se accede a los bloques devueltos por ambos

Selección - Disyunción de condiciones

- Si hay una disyunción de condiciones, es más complicado aprovechar índices

$$\sigma_{A_i=v1 \vee A_j=v2}(R)$$

- Si no tenemos un índice por alguno de los atributos, hay que hacer File Scan
- Si tenemos índices por ambos, podríamos acceder a los dos índices y a los datos devueltos por cada uno de ellos (unión entre los resultados de ambos)

Selección - Ejemplo

- En base a los siguientes datos de la tabla de Publicaciones, estime el costo de las siguientes consultas
 - Publicaciones(Id, Autor, Tipo, Titulo, Contenido)
1. σ Autor='Mariano Beiró' (Publicaciones)
 2. σ Tipo='C' (Publicaciones)
 3. σ Autor='Mariano Beiró' \wedge Tipo='C' (Publicaciones)

n(Publicaciones)	1,000,000
B(Publicaciones)	100,000
V(Autor, Publicaciones)	200,000
V(Tipo, Publicaciones)	10
Height(I(Autor,Publicaciones))	4
Height(I(Tipo,Publicaciones))	2

Los índices no son de clustering

Selección - Ejemplo resuelto

1. σ Autor='Mariano Beiró' (Publicaciones)

El costo es $4 + 1,000,000 / 200,000 = 9$

2. σ Tipo='C' (Publicaciones)

De usar índice el costo sería $2 + 100,000$

Conviene un file scan con costo 100,000

3. σ Autor='Mariano Beiró' \wedge Tipo='C' (Publicaciones)

Igual costo que el primer caso: 9

* El índice de tipo no se aprovecha (me devolvería todos los bloques que el de autor)

n(Publicaciones)	1,000,000
B(Publicaciones)	100,000
V(Autor, Publicaciones)	200,000
V(Tipo, Publicaciones)	10
Height(I(Autor,Publicaciones))	4
Height(I(Tipo,Publicaciones))	2

Selección - Distribución no uniforme de los valores

- Cuando los valores no tienen una distribución uniforme, la estimación de la fórmula vista no es buena

$$\text{Cost}(\sigma_{A_i=v}(R)) = \text{Height}(I(A,R)) + \lceil n(R) / V(A,R) \rceil$$

- Si estoy buscando clientes por país, quizás el 95 % de los clientes son de Argentina y el 0.1% de un país remoto
- La fórmula debería adaptarse ya que los costos son muy distintos
 - Probablemente no convenga Index Scan para Argentina

Selección - Histogramas

- Para mejorar la estimación, se utilizan Histogramas que guardan la frecuencia de los valores que tiene cada columna
 - A veces no todos, sino los N más frecuentes
 - Útil para valores discretos y con repeticiones

Selección - Ejemplo con histogramas

- Clientes(id, nombre, país,)
 - El índice por país no es de clustering

n(Clientes)	100,000
B(Clientes)	10,000
V(país, Clientes)	50
Height(I(país, Clientes))	3

$\text{Cost}(\sigma_{\text{país}='Brasil'}(\text{Clientes})) = ?$

$\text{Cost}(\sigma_{\text{país}='Argentina'}(\text{Clientes})) = ?$

$\text{Cost}(\sigma_{\text{país}='Tonga'}(\text{Clientes})) = ?$

- Histograma con 3 valores más frecuentes de país

	Argentina	Uruguay	Brasil
Clientes.país	95,000	3,000	1,000

Selección - Distribución no uniforme de los valores

- Para Brasil, se accedieran a 1,000 bloques ya que el histograma nos dice que mil filas son de brasil

$$\text{Cost}(\sigma_{\text{país}='Brasil'}(\text{Clientes})) = 3 + 1000 = 1003$$

- Para Argentina conviene File Scan

$$\text{Cost}(\sigma_{\text{país}='Argentina'}(\text{Clientes})) = 10,000$$

Se sabe que son mil por el histograma

- Para tonga, se usa la fórmula quitando los valores conocidos

$$\text{Cost}(\sigma_{A_i=v}(R)) = 3 + \lceil 1,000 / 47 \rceil = 3 + 22 = 25$$

100K clientes menos los 95K de Argentina, 3K de Uruguay 1K de Brasil

50 países menos los 3 de los que se conoce su frecuencia

Selección - Estimación de la cardinalidad

- Es útil saber cuántas filas fueron devueltas por el operador
- Si no se posee histograma, se estima la cardinalidad según la variabilidad

$$n(\sigma_{A_i=v}(R)) = \lceil n(R) / V(A,R) \rceil$$

- Si se tiene histograma y se conoce el valor, se lo utiliza
- Si se tiene histograma pero no se conoce el valor, se estima según la variabilidad
 - En vez de $n(R)$ se usa $n(R)$ menos las frecuencias conocidas
 - A la variabilidad $V(A,R)$ se le resta la cantidad de frecuencias conocidas

Selección - Estimación de cantidad de bloques

- También puede interesar saber cuántos bloques fueron devueltas por el operador
- La selección no cambia el tamaño de las filas, así que el factor de bloque $F(R)$ se mantiene
- Conociendo el factor de bloque de R y la cardinalidad de la selección, se puede estimar la cantidad de bloques devueltos

$$B(\sigma_{A_i=v}(R)) = \lceil n(\sigma_{A_i=v}(R)) / F(R) \rceil$$

Selección - Cardinalidad y Bloques en el ejemplo

- $F(\text{Clientes}): 100,000 / 10,000 = 10$ (Entran 10 filas por bloque)
- Para Brasil y Argentina se usa el histograma para la cardinalidad
 - $n(\sigma_{\text{país}='Brasil'}(\text{Clientes})) = 1,000$ (usa 100 bloques)
 - $n(\sigma_{\text{país}='Argentina'}(\text{Clientes})) = 95,000$ (usa 9,500 bloques)
- Para tonga, se estima quitando los valores conocidos
 - $n(\sigma_{\text{país}='Tonga'}(\text{Clientes})) = \lceil (100,000 - 99,000) / (50 - 3) \rceil$
 - $n(\sigma_{\text{país}='Tonga'}(\text{Clientes})) = \lceil 1,000 / 47 \rceil = 22$ (usa 3 bloques)

Costos de Operadores

π - Proyección

Proyección

- Es importante saber si hay que eliminar o no duplicados
- Si la consulta no tiene **DISTINCT** (el resultado es un multiset), o si se proyecta por una superclave, no es necesario eliminar duplicados
- El costo es $\text{Cost}(\pi_x(R)) = B(R)$
- Si hay que eliminar duplicados, hay que mantenerlos en memoria para saber si devolver o no un valor posterior
 - Si $B(\pi_x(R)) > \text{memoria disponible}$, el costo deja de ser $B(R)$

Proyección - Estimación de la cardinalidad

- Si no evitamos duplicados, salen tantas filas como había
 - $n(\pi_X(R)) = n(R)$
- Si evitamos duplicados, podemos llegar a estimar según las variabilidades
 - $n(\pi_{A1}(R)) = V(A1, R)$
 - $n(\pi_{A1, A2}(R)) = V(A1, R) * V(A2, R)$ si no hay correlación entre los atributos

Proyección - Estimación de cantidad de bloques

- La cantidad de bloques puede ser menor ya que la fila ocupa menos
 - Se modifica $F(R)$
- Tenemos que tener de algún modo conocimiento de lo que ocupan los atributos proyectados vs los originales

Personas(DNI, nombre, fecha_nac, genero)	
DNI	4 bytes
nombre	15 bytes
fecha_nac	4 bytes
genero	1 byte

$$F(\pi_{\text{DNI}}(\text{Personas})) = F(\text{Personas}) * 4/24$$

$$B(\pi_{\text{DNI}}(\text{Personas})) = B(\text{Personas}) * 4/24$$

Proyección - Sort externo

- Cuando el resultado no entra en memoria y hay que eliminar duplicados, con leer la tabla no alcanza
 - La última fila puede producir un valor que ya se envió con la primera fila
- Para resolver este problema, podemos primero ordenar la tabla por el atributo de proyección con un **Sort externo**
 - Esta operación tiene un costo mayor
 - En la última etapa, al leerse en forma ordenada los datos, se devuelve el primero de cada uno de ellos

Sort externo - Etapas

- Tenemos M bloques de memoria disponibles
- **Primera etapa:** Generamos particiones ordenadas de $M-1$ bloques en memoria (Sort interno)
 - 1 bloque de memoria lo usamos para acumular la salida
- **Segunda etapa:** Recorremos $M-1$ particiones ordenadas a la vez, y generamos una única partición con los datos ordenados de esas particiones (Merge)

Sort externo - Ejemplo

Bloque	Datos	Bloque	Datos
1	1, 20	11	84,97
2	7, 29	12	4,35
3	3, 39	13	88,93
4	33, 52	14	15,25
5	61, 69	15	48,49
6	53, 2	16	72,65
7	50, 51	17	70,22
8	9, 80	18	79,5
9	6, 37		
10	60, 11		

Memoria			

- Archivo de 18 bloques
- 2 filas por bloque
- 4 bloques de memoria para trabajar ($M = 4$)

Sort externo - Ejemplo etapa 1 (sort interno)

Bloque	Datos
1	1, 20
2	7, 29
3	3, 39
4	33, 52
5	61, 69
6	53, 2
7	50, 51
8	9, 80
9	6, 37
10	60, 11

Bloque	Datos
11	84,97
12	4,35
13	88,93
14	15,25
15	48,49
16	72,65
17	70,22
18	79,5

Memoria			
1, 20	7, 29	3, 39	



- Se leen 3 bloques y se genera una partición ordenada

Part1
1,3
7,20
29,39

Sort externo - Ejemplo etapa 1 (sort interno)

Bloque	Datos
1	1, 20
2	7, 29
3	3, 39
4	33, 52
5	61, 69
6	53, 2
7	50, 51
8	9, 80
9	6, 37
10	60, 11

Bloque	Datos
11	84,97
12	4,35
13	88,93
14	15,25
15	48,49
16	72,65
17	70,22
18	79,5

Memoria			
33, 52	61, 69	53, 2	



- Se leen otros 3 bloques y se genera otra partición

Part2
2,33
52,53
61,69

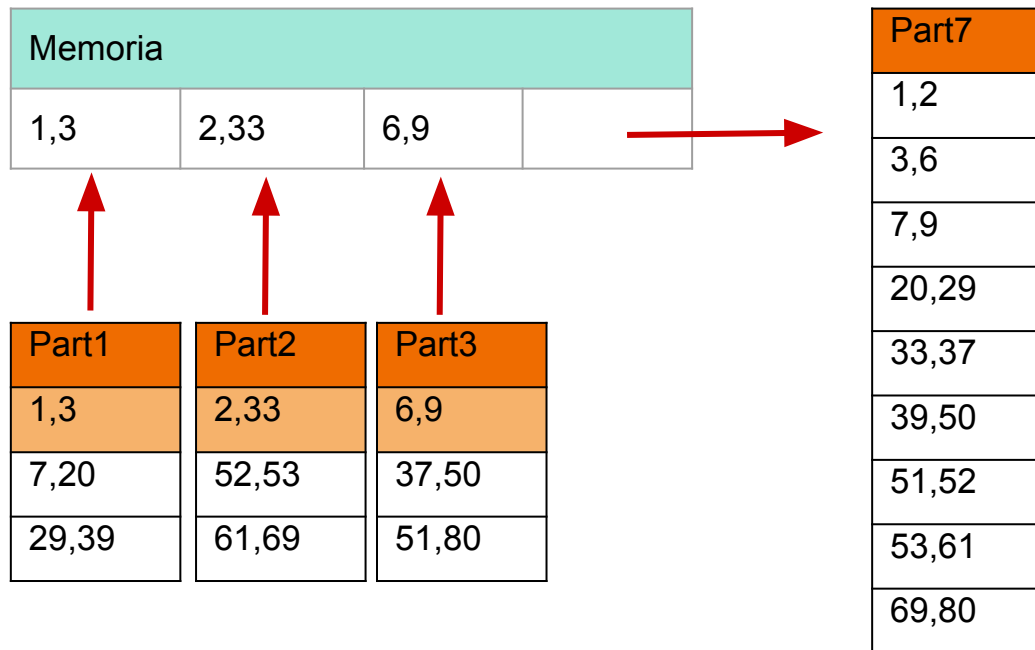
Sort externo - Ejemplo etapa 1 (sort interno)

Bloque	Datos	Bloque	Datos
1	1, 20	11	84,97
2	7, 29	12	4,35
3	3, 39	13	88,93
4	33, 52	14	15,25
5	61, 69	15	48,49
6	53, 2	16	72,65
7	50, 51	17	70,22
8	9, 80	18	79,5
9	6, 37		
10	60, 11		

Part1	Part2	Part3	Part4	Part5	Part6
1,3	2,33	6,9	4,11	15,25	5,22
7,20	52,53	37,50	35,60	48,49	65,70
29,39	61,69	51,80	84,97	88,93	72,79

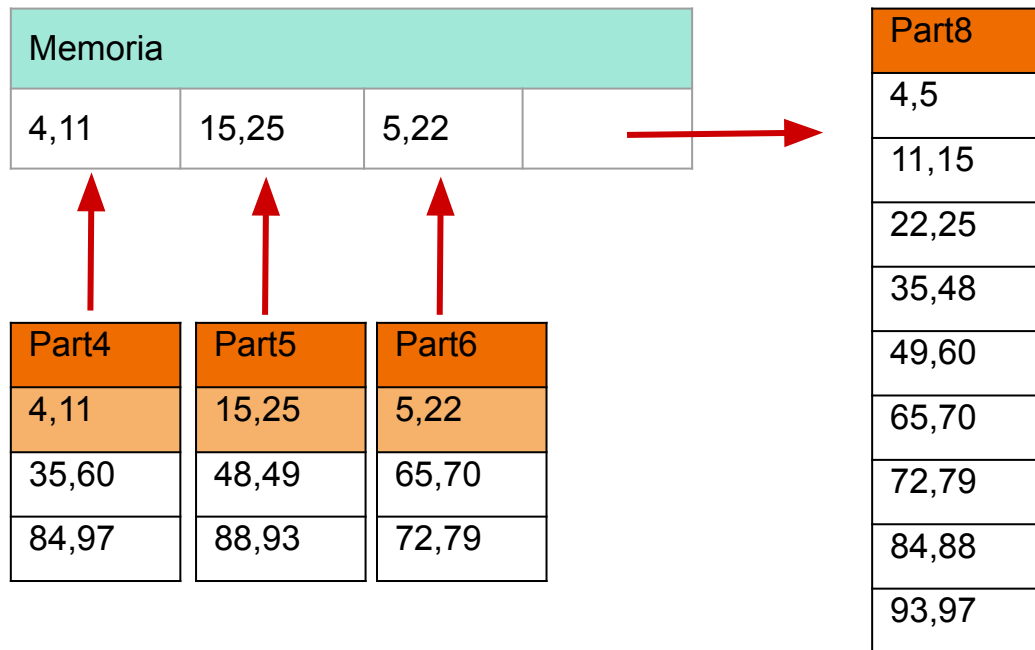
- Al fin de la primer etapa se leyeron y grabaron tantos bloques como tenia el archivo ($B(R)$)
- Se generaron $B(R) / (M-1)$ particiones ordenadas

Sort externo - Ejemplo etapa 2 (merge)



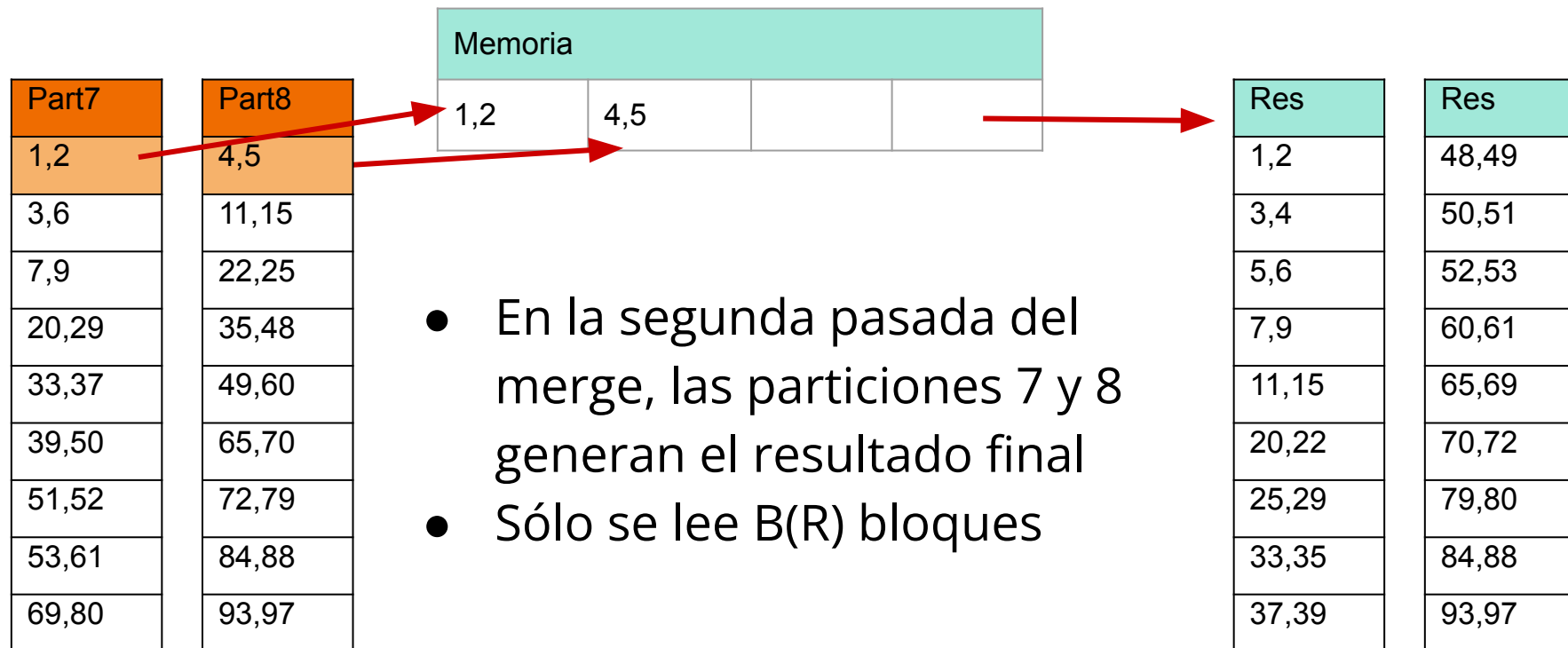
- Se hace merge de a 3 particiones
- Se toma siempre el menor número
- Cuando se usó un bloque de una partición, se avanza al siguiente

Sort externo - Ejemplo etapa 2 (merge)



- Lo mismo para las siguientes 3 particiones
- Fin de la primera pasada del merge
- Se habrán leído B(R) y escrito B(R) bloques

Sort externo - Ejemplo etapa 2 (merge)



Sort externo - Costo

- La cantidad total de etapas está dada por $\log_{M-1}(B(R))$
 - Hasta 3 bloques: Solo primera etapa (sort interno)
 - De 3 a 9 bloques: Primera etapa y 1 pasada de merge
 - 10 a 27 bloques: Primera etapa y 2 pasadas de merge
- Siempre se escribe y lee $B(R)$ salvo en la última etapa que sólo se lee
- Definimos un costo de ordenamiento basado en M bloques de memoria:

$$\text{Cost}(\text{Ord}_M(R)) = 2 * B(R) * \lceil \log_{M-1}(B(R)) \rceil - B(R)$$

Proyección - Costo con sort externo

- Para resolver la proyección con sort externo, salvo la primera lectura, no se trabaja con toda la fila
 - Se trabaja con las columnas proyectadas
 - Tienen un $B(\pi x(R)) < B(R)$
- Adaptamos la fórmula trabajando con el $B(\pi x(R))$
 - Sumamos $B(R)$ y restamos $B(\pi x(R))$ para compensar

$$\text{Cost}(B(\pi x(R))) = 2 * B(\pi x(R)) * \lceil \log_{M-1}(B(\pi x(R))) \rceil - \underbrace{2 * B(\pi x(R)) + B(R)}$$

Acá se restó $B(\pi x(R))$ y se sumó $B(R)$

Proyección - Ejemplo completo

- Estime el costo, cardinalidad y cantidad de bloques de la proyección sin repetidos $\pi_{\text{Nombre, Apellido}}(\text{Clientes})$
- Sabiendo que:
 - Se dispone de $M=11$ bloques de memoria
 - El nombre y apellido ocupan un 20% del tamaño de la fila completa de Clientes
 - $n(\text{Clientes}) = 100,000$
 - $B(\text{Clientes}) = 10,000$
 - $V(\text{Nombre, Clientes}) = 200$
 - $V(\text{Apellido, Clientes}) = 400$

Proyección - Ejemplo completo resuelto

- $B(\pi_{\text{Nombre, Apellido}}(\text{Clientes})) = 10,000 * 0.2 = 2,000$

$$\text{Cost}(B(\pi_X(R))) = 2 * B(\pi_X(R)) * \lceil \log_{M-1}(B(\pi_X(R))) \rceil - 2 * B(\pi_X(R)) + B(R)$$

$$\text{Cost}(B(\pi_X(R))) = 4,000 * \lceil \log_{10}(2,000) \rceil - 4,000 + 10,000$$

$$\text{Cost}(B(\pi_X(R))) = 4,000 * 4 + 6,000 = 22,000$$

- $n(\pi_{\text{Nombre, Apellido}}(\text{Clientes})) = 200 * 400 = 80,000$
- $F(\pi_{\text{Nombre, Apellido}}(\text{Clientes})) = 10 / 0.2 = 50$
- $B(\pi_{\text{Nombre, Apellido}}(\text{Clientes})) = 1,600$

Costos de Operadores

- \cup \cap - Operadores de conjunto

Operadores de conjunto

- Para los tres casos, se debe trabajar con ambas tablas ordenadas
 - Si no entran en memoria y hay que ordenarlas, usar un sort externo
 - Al costo de ordenamiento se le suma grabar a disco la tabla ordenada
 - Luego se leen ambas tablas en orden y se procesan las filas
- Costo total:

$$\text{Cost}(R \cup R - S) = \text{Cost}(\text{OrdM}(R)) + \text{Cost}(\text{OrdM}(S)) + 2 * (B(R) + B(S))$$

Operadores de conjunto - consideraciones

- Por defecto se trabaja como el álgebra relacional, sin repetidos
 - Se pueden adaptar los algoritmos que veremos para trabajar con multi-sets, permitiendo que estén repetidos los datos en la respuesta
- Los datos de entrada pueden tener repetidos
 - R y S podrían no ser tablas sino resultados de operaciones anteriores

Operadores de conjunto - Unión

- Se recorren ordenadas las filas r y s de R y S
- Se deben devolver todas las filas
 - Si $r = s$ se devuelve una sola de ellas y se avanza sobre ambas tablas hasta que cambien
 - Si son distintas, se devuelve la menor de ellas y se avanza en su tabla hasta que cambie
 - Cuando se llega al final de una tabla, se devuelve todo lo que queda en la otra, sin duplicados

Operadores de conjunto - Intersección

- Se recorren ordenadas las filas r y s de R y S
- Se deben devolver las filas que están en ambas
 - Si $r = s$ se devuelve una sola de ellas y se avanza sobre ambas tablas hasta que cambien
 - Si son distintas, se avanza la tabla de la que tiene el menor valor hasta que cambie, sin devolver nada
 - Cuando se llega al final de una tabla, se termina el algoritmo

Operadores de conjunto - Resta

- Se recorren ordenadas las filas r y s de R y S
- Se deben devolver las filas que están sólo en r
 - Si $r = s$ se avanza sobre ambas tablas hasta que cambien de valor, sin devolver nada
 - Si $r > s$ se avanza la tabla S hasta que cambie de valor
 - Si $r < s$ se devuelve r y se avanza R hasta que cambie de valor
 - Si termina R , se finaliza el algoritmo
 - Si termina S , se devuelven todos los r restantes, sin repetidos, y se termina el algoritmo

Operadores de conjunto - Cardinalidad y bloques

- Estimar la cardinalidad es difícil ya que no se conoce la cardinalidad de la intersección
 - En la unión, debería restarse de $n(R) + n(S)$
 - En la resta debería restarse de $n(R)$
- El factor $F(R)$ se mantiene, con lo que si se supiera la cardinalidad se puede calcular la cantidad de bloques

Operadores de conjunto - Ejemplo completo

- Estime el costo, cardinalidad y cantidad de bloques de la siguiente consulta que devuelve los ids de cliente que son Premium y/o gold:

Premium \cup Gold

Asuma que un 50% de los clientes premium también son gold y que se tiene $M=11$ bloques de memoria

Premium(<u>idcliente</u>)		Gold(<u>idcliente</u>)	
n(Premium)	5,000	n(Gold)	100,000
B(Premium)	50	B(Gold)	1,000

Operadores de conjunto - Resolución

- El costo de ordenar cada tabla es

$$\text{Cost}(\text{Ord}_{11}(\text{Premium})) = 2 * 50 * \lceil \log_{10}(50) \rceil - 50 = 150$$

$$\text{Cost}(\text{Ord}_{11}(\text{Gold})) = 2 * 1,000 * \lceil \log_{10}(1,000) \rceil - 1,000 = 5,000$$

- El costo total es $150 + 5,000 + 2 * (50 + 1,000) = 7,250$
- La intersección entre ambos es la mitad de los premium, es decir 2,500.
 - Entonces la unión devuelve $100,000 + 5,000 - 2,500 = 102,500$ filas
- Entran 100 por bloque entonces se devolverán 1,025 bloques

Costos de Operadores

⌘ Join / Junta

Join - Distintas alternativas

- El Join suele ser la operación más demandante
- Existen 4 alternativas para resolverla
 - Loops anidados por bloque
 - Loop con único índice
 - Sort-merge
 - Junta Hash grace
- La elección depende de la memoria disponible, los tamaños de las relaciones y la existencia de índices

Join - Loops anidados por bloque

- Si tengo en memoria un bloque de R y un bloque de S, puedo hacer el join entre todas las filas de esos bloques
 - Comparo cada fila de R con todas las de S y veo si cumple la condición de Join
 - Es el único de los 4 métodos que sirve para cualquier tipo de condición
- El problema entonces es lograr cruzar en memoria a cada bloque de R con cada bloque de S

Join - Loops anidados por bloque

- Podría leer el primer bloque de R y todos los bloques de S haciendo el join, luego el segundo bloque de R y todos los de S haciendo el join, hasta leer el último bloque de R y todos los de S
- Esto requiere al menos 3 bloques de memoria
 - Dos para ir leyendo R y S bloque a bloque
 - Uno para ir acumulando los resultados

Join - Loops anidados por bloque - Costo

- El costo entonces estará dado por
$$\text{Cost}(R \bowtie S) = B(R) + B(R) * B(S)$$
 - La tabla R se lee completamente una vez
 - La tabla S se lee completamente una vez por cada bloque de R
- Si S fuera más chica que R convendría hacerlo al revés
$$\text{Cost}(R \bowtie S) = B(S) + B(S) * B(R)$$

Join - Loops anidados por bloque - Costo $M > 3$

- Puedo mejorar el algoritmo si tuviera más memoria disponible
- Si tengo 4 bloques, puedo leer de a 2 bloques de R y comparar todo S contra esos dos bloques
 - Se reducen a la mitad las veces que se lee S completamente
- El costo pasaría a ser $B(R) + \lceil B(R)/2 \rceil * B(S)$

Join - Loops anidados por bloque - Costo $M > 3$

- Lo mejor que puede pasar es que M tenga dos bloques más que una de las tablas (la de menor tamaño), entonces cargo primero toda esa tabla en memoria y luego proceso la otra bloque a bloque juntando con todas las filas que ya tengo en memoria
- El costo es $B(R) + B(S)$

Join - Loops anidados por bloque - Fórmula

- Generalizando, si tengo M bloques de memoria disponible y R es la relación con menor cantidad de bloques, el costo por loops anidados es:

$$\text{Cost}(R \bowtie S) = B(R) + \lceil B(R) / (M-2) \rceil * B(S)$$

Join - Loop con único índice

- Cuando la condición es de igualdad y una tabla (S) tiene un índice sobre los campos de la condición, puede llegar a aprovecharse ese índice
- Se recorre bloque a bloque la otra tabla (R) y para cada fila, se hace un index scan en la tabla con índice (S)
 - Ejemplo: Se recorre la tabla de alumnos, y para cada alumno se hace un index scan en la tabla de notas por su valor de padrón

Join - Loop con único índice

- Si hay índice en ambas tablas, se debe ver si conviene empezar por una o la otra
 - No se puede usar ambos índices a la vez
- El costo depende de si el índice es o no de clustering

Join - Loop con único índice - Fórmula

- Si se usa el índice I sobre el atributo A de la tabla S y no es de clustering, el costo es

$$\text{Cost}(R \bowtie S) = B(R) + n(R) * (\text{Height}(I(A,S)) + \lceil n(S) / V(A,S) \rceil)$$

- En cambio, si el índice es de clustering

$$\text{Cost}(R \bowtie S) = B(R) + n(R) * (\text{Height}(I(A,S)) + \lceil B(S) / V(A,S) \rceil)$$

Join - Ejercicio

- Estimar el costo de la junta natural Clientes \bowtie Ordenes utilizando el método de loop con único índice.
- Indicar desde qué valor de memoria M es conveniente resolverlo con el método de loops anidados

Clientes (<u>nro_cliente</u> , nombre		Ordenes (<u>nro_orden</u> , nro_cliente,)	
n(Clientes)	5,000	n(Ordenes)	10,000
F(Clientes)	20	F(Ordenes)	25
		V(nro_cliente, Ordenes)	2,500

- Hay un índice de clustering por nro_cliente en Ordenes con altura 4

Join - Ejercicio - Resolución con loop con único índice

- $B(\text{Clientes}) = 5,000 / 20 = 250$
- $B(\text{Ordenes}) = 10,000 / 25 = 400$
- $\text{Cost}(R \bowtie S) = 250 + 5000 * (4 + \lceil 400 / 2500 \rceil) = 25,250$

Clientes (<u>nro_cliente</u> , nombre		Ordenes (<u>nro_orden</u> , nro_cliente,)	
n(Clientes)	5,000	n(Ordenes)	10,000
F(Clientes)	20	F(Ordenes)	25
		V(nro_cliente, Ordenes)	2,500

- Hay un índice de clustering por nro_cliente en Ordenes con altura 4

Join - Ejercicio - Resolución con loops anidados

- $B(\text{Clientes}) = 5,000 / 20 = 250$
- $B(\text{Ordenes}) = 10,000 / 25 = 400$
- $\text{Cost}(R \bowtie S) = 250 + \lceil 250 / (M-2) \rceil * 400$
 - Busco M tal que el costo sea menor a 25,250
 - Con $M \geq 7$ comienza a convenir usar loops anidados
 - El costo sería $250 + 5 * 400 = 20,250$

Join - Sort Merge

- Cuando la condición es de igualdad y las tablas están ordenadas por los atributos de la condición, se puede ir procesando bloque a bloque de modo similar a los operadores de conjunto
 - La limitante de memoria sería el poder almacenar todos los bloques que tienen un mismo valor en los atributos de la condición
- El costo vendría por leer bloque a bloque ambas tablas

Join - Sort Merge

- Como en general las tablas no están ordenadas por los atributos del join, podemos pensar el costo del sort merge como la suma de:
 - Ordenar la tabla R y guardar el resultado
 - Ordenar la tabla S y guardar el resultado
 - Leer los resultados ordenados y procesar el join
- Si alguna estuviera ordenada, se ahorra el costo de ordenar y de grabar a disco

Join - Sort Merge

- El costo total entonces es:

$$\text{Cost}(R \bowtie S) = \text{Cost}(\text{Ord}_M(R)) + \text{Cost}(\text{Ord}_M(S)) + 2 * (B(R) + B(S))$$

- Recordando el costo de ordenar una tabla como:

$$\text{Cost}(\text{Ord}_M(R)) = 2 * B(R) * \lceil \log_{M-1}(B(R)) \rceil - B(R)$$

- Queda entonces una fórmula final de

$$\begin{aligned} \text{Cost}(R \bowtie S) = & 2 * B(R) * \lceil \log_{M-1}(B(R)) \rceil + B(R) \\ & + 2 * B(S) * \lceil \log_{M-1}(B(S)) \rceil + B(S) \end{aligned}$$

Join - Junta Hash Grace

- Vimos antes que si la tabla más pequeña entra en memoria (con 2 bloques extra) el costo total es $B(R) + B(S)$
- Cuando la tabla más pequeña no entra en memoria, sería conveniente poder dividir ambas tablas en distintas **particiones** de tal modo que siempre las particiones de la menor tabla entren en memoria
 - Se harían N joins con un costo $B(R_i) + B(S_i)$ de cada particion

Join - Junta Hash Grace

- Para que esto funcione, las filas de R y de S que deben juntarse deben quedar en la misma partición i
- Para lograr esto, se aplica una función de hash a las columnas por las que se haga el join y el resultado determina la partición
 - Si dos filas van a juntarse, tienen el mismo valor y por ende irán a la misma partición
 - Únicamente para joins por igualdad

Join - Junta Hash Grace - Etapas

- En una primera etapa, se lee cada tabla, generandose N particiones. Cada fila se guarda únicamente en la partición que corresponde
 - Se leen y escriben tantos bloques como tenga la tabla
- En una segunda etapa, se hace un join por loops anidados de cada partición
 - Al entrar en memoria, cada partición tiene un costo de $B(R_i) + B(S_i)$, que sumados dan un costo de $B(R) + B(S)$

Join - Junta Hash Grace - Costo final

- En total se leyó $B(R)$ y $B(S)$, se escribió $B(R)$ y $B(S)$ y luego se leyeron entre todas las particiones nuevamente $B(R)$ y $B(S)$ bloques

$$\text{Cost}(R \bowtie S) = 3 * B(R) + 3 * B(S)$$

- Este costo igualmente está **muy limitado** por la **memoria disponible**

Join - Junta Hash Grace - Límites de la memoria

- Para poder hacer este método, necesitamos **definir la cantidad de particiones** (N) a hacer, y esto está limitado por la memoria disponible (M)
- En la primera etapa voy leyendo cada tabla bloque a bloque, y para acumular las filas en las particiones que corresponde, no puedo usar más de los otros M-1 bloques
- **Primer límite:** $N < M - 1$

Join - Junta Hash Grace - Límites de la memoria

- Si la función de hash distribuye bien los valores, cada partición tendrá $B(R)/N$ y $B(S)/N$ bloques
- Las particiones de la menor tabla deberán entrar completamente en memoria dejando 2 bloques extra para poder usar loops anidados de forma eficiente
- **Segundo límite:** $\min(\lceil B(R)/N \rceil ; \lceil B(S)/N \rceil) < M - 2$

Join - Junta Hash Grace - Límites de la memoria

- Un tercer límite está definido por la variabilidad de los atributos de la junta
- Si la cantidad de valores distintos es menor a N , no voy a poder llenar N particiones sino $V(A,R)$ particiones
 - Algunas particiones quedarán vacías, y las que tengan datos tendrán un tamaño mayor a $B(R)/N$ y no entrarán en memoria
- **Tercer límite:** $\min(\lceil B(R)/V(A,R) \rceil ; \lceil B(S)/V(A,S) \rceil) < M - 2$

Join - Junta Hash Grace - Ejemplo

- Se quiere hacer un join entre las tablas de alumno y de notas por padrón
- $B(\text{Alumnos}) = 100$, $B(\text{Notas}) = 300$, $M = 50$
- Se pueden armar 10 particiones usando como función de hash el resto de la división por 10 del padrón
 - El último dígito
 - No es una buena función de hash, sólo sirve para ejemplificar

Join - Junta Hash Grace - Ejemplo

- En la primera pasada se precisan 11 bloques de memoria,
 - 1 para leer la tabla y 10 para acumular particiones, que se van grabando a medida que se llenan.
- En la segunda etapa se procesan por separado las 10 particiones. Cada partición de alumnos tiene 10 bloques así que entra completa en memoria
 - Alumnos y sus notas quedan en la misma partición
- El costo total es $3 * (100 + 300) = 1200$ bloques

Join - Resumen de costos

- Loops anidados

$$\text{Cost}(R \bowtie S) = B(R) + \lceil B(R) / (M-2) \rceil * B(S)$$

- Único loop con índice

$$\text{Cost}(R \bowtie S) = B(R) + n(R) * (\text{Costo Index scan})$$

- Sort Merge

$$\text{Cost}(R \bowtie S) = 2 * B(R) * \lceil \log_{M-1}(B(R)) \rceil + B(R) + 2 * B(S) * \lceil \log_{M-1}(B(S)) \rceil + B(S)$$

- Junta Hash Grace

$$\text{Cost}(R \bowtie S) = 3 * B(R) + 3 * B(S)$$

Join - Ejercicio comparativo

- Calcule el costo de hacer el join entre R y S sabiendo que no se poseen índices a aprovechar, que $B(R) = 1,000$ y que $B(S) = 10,000$. Indique qué método es conveniente para tres casos distintos de memoria:
 1. $M = 12$
 2. $M = 102$
 3. $M = 1,002$

Join - Ejercicio comparativo - Loops anidados

- Usando la fórmula, el costo es
$$\text{Cost}(R \bowtie S) = 1,000 + \lceil 1,000 / (M-2) \rceil * 10,000$$
- Para cada caso el costo termina siendo
 1. $M = 12 \Rightarrow \text{Cost}(R \bowtie S) = 1,001,000$
 2. $M = 102 \Rightarrow \text{Cost}(R \bowtie S) = 101,000$
 3. $M = 1,002 \Rightarrow \text{Cost}(R \bowtie S) = 11,000$
- Loop con único índice no puede hacerse (no hay índice)

Join - Ejercicio comparativo - Sort Merge

- Usando la fórmula, el costo es

$$\text{Cost}(R \bowtie S) = 2,000 * \lceil \log_{M-1}(1,000) \rceil + 1,000$$

$$+ 20,000 * \lceil \log_{M-1}(10,000) \rceil + 10,000$$
- Para cada caso el costo termina siendo
 1. $M = 12 \Rightarrow \text{Cost}(R \bowtie S) = 97,000$
 2. $M = 102 \Rightarrow \text{Cost}(R \bowtie S) = 55,000$
 3. $M = 1,002 \Rightarrow \text{Cost}(R \bowtie S) = 53,000$

Logaritmos	1,000	10,000
$\lceil \log_{11}(\dots) \rceil$	3	4
$\lceil \log_{101}(\dots) \rceil$	2	2
$\lceil \log_{1001}(\dots) \rceil$	1	2

Join - Ejercicio comparativo - Hash Grace

- Usando la fórmula, el costo es siempre igual
 $\text{Cost}(R \bowtie S) = 3 * (1,000 + 10,000)$
- Con $M=12$ se pueden hacer 11 particiones, las particiones de R tendrán 90 bloques con lo que no entran en memoria
- Recién con 102 bloques se puede hacer hash grace
 2. $M = 102 \Rightarrow \text{Cost}(R \bowtie S) = 33,000$
 3. $M = 1,002 \Rightarrow \text{Cost}(R \bowtie S) = 33,000$

Join - Ejercicio comparativo - tabla final

M	Loops anidados	Sort Merge	Hash Grace	Mejor método
12	1,001,000	97,000		Sort Merge
102	101,000	55,000	33,000	Hash Grace
1002	11,000	53,000	33,000	Loops Anidados

Join - Cardinalidad y cantidad de bloques

- La cantidad de filas que salen depende de la cantidad de filas de cada tabla y la distribución de los valores de los atributos de junta
 - Nunca tendremos más de $n(R) * n(S)$ filas
- Pero el Join también cambia el factor de bloque F
 - Una fila del join tiene los campos de ambas tablas, por ende ocupa más espacio, y entran menos por bloque

Join - Cardinalidad

- Para estimar la cardinalidad de la junta con condición de igualdad, debemos asumir:
 - Distribución equitativa de las filas entre los valores de los atributos de junta
 - Con un histograma podemos mejorar la estimación
 - Que los valores de la tabla con menor variabilidad en los atributos de junta están incluidos entre los de la tabla con mayor variabilidad

Join - Cardinalidad sin histograma

- Asumamos junta por $R.A = S.A$ y distribución equitativa
- En R habrá $V(A,R)$ conjuntos de $n(R)/V(A,R)$ filas para cada valor del atributo A
- En S habrá $V(A,S)$ conjuntos de $n(S)/V(A,S)$ filas
- Las filas de cada conjunto se juntan entre si, quedando $n(R) * n(S) / (V(A,R) * V(A,S))$ filas juntadas
 - ¿Cuántos conjuntos en total se juntan entre si?

Join - Cardinalidad sin histograma

- ¿Cuántos conjuntos en total se juntan entre sí?
- Si $V(A,R) < V(A,S)$, se juntan $V(A,R)$ conjuntos
 - Total: $V(A,R) * (n(R) * n(S) / (V(A,R) * V(A,S)))$
- Si $V(A,S) < V(A,R)$, se juntan $V(A,S)$ conjuntos
 - Total: $V(A,S) * (n(R) * n(S) / (V(A,R) * V(A,S)))$
- Formula general:

$$n(R \bowtie S) = n(R) * n(S) / \max(V(A,R), V(A,S))$$

Join - Junta por más de un campo

- Si la junta es por varios atributos, entonces hay que considerar en cada tabla la multiplicación de variabilidades
 - Salvo que tuviéramos un conocimiento de cómo se vinculan ambos atributos
- Formula adaptada:
$$n(R \bowtie S) = n(R) * n(S) / \max(V(A1,R) * V(A2,R) , V(A1,S) * V(A2,S))$$

Join - Ejemplo cardinalidad sin histogramas

- Clientes(id, nombre, país, ...) y Proveedores(id, nombre, país, ...)
- $n(\text{Clientes} \bowtie_{\text{clientes.pais} = \text{Proveedores.pais}} \text{Proveedores}) = ?$

$n(\text{Clientes})$	100,000
$V(\text{país}, \text{Clientes})$	50
$B(\text{Clientes})$	10,000
$n(\text{Proveedores})$	10,000
$V(\text{país}, \text{Proveedores})$	40
$B(\text{Proveedores})$	2,500

Join - Ejemplo cardinalidad sin histogramas

- Clientes(id, nombre, país, ...) y Proveedores(id, nombre, país, ...)
- $n(\text{Clientes} \bowtie_{\text{clientes.pais} = \text{Proveedores.pais}} \text{Proveedores}) = ?$

$n(\text{Clientes})$	100,000
$V(\text{país}, \text{Clientes})$	50
$B(\text{Clientes})$	10,000
$n(\text{Proveedores})$	10,000
$V(\text{país}, \text{Proveedores})$	40
$B(\text{Proveedores})$	2,500

- $100,000 * 10,000 / \max(50, 40)$
- $n(\text{Clientes} \bowtie_{\text{clientes.pais} = \text{Proveedores.pais}} \text{Proveedores}) = 20,000,000$

Join - Cardinalidad con histogramas

- Un histograma me puede ayudar a estimar mejor la cardinalidad
- Si un valor de atributo aparece en ambos histogramas, las filas de cada tabla de ese valor se juntaran entre si, multiplicandose
- Si un valor de atributo aparece en uno solo de los histogramas, estimar la cantidad de filas en la otra tabla y multiplicar
- Para el resto de los atributos, utilizar la fórmula anterior

Join - Ejemplo cardinalidad con histogramas

- Clientes(id, nombre, país, ...) y Proveedores(id, nombre, país, ...)
- $n(\text{Clientes} \bowtie_{\text{clientes.pais} = \text{Proveedores.pais}} \text{Proveedores}) = ?$
- Histograma de 3 valores más frecuentes en cada tabla

$n(\text{Clientes})$	100,000
$V(\text{país, Clientes})$	50
$B(\text{Clientes})$	10,000
$n(\text{Proveedores})$	10,000
$V(\text{país, Proveedores})$	40
$B(\text{Proveedores})$	2,500

	Argentina	Uruguay	Brasil
Clientes.pais	95,000	3,000	1,000

	Argentina	Brasil	Perú
Proveedores.pais	8,000	1,000	900

Join - Ejemplo cardinalidad con histogramas

- Armamos una única tabla, agregando el “otros”
- Los valores conocidos se juntan entre sí
 - $95,000 * 8,000$ combinaciones para argentina
 - $1,000 * 1,000$ para brasil

	Argentina	Uruguay	Brasil	Perú	Otros	V(Otros)
Clientes.pais	95,000	3,000	1,000		1,000	47
Proveedores.pais	8,000		1,000	900	100	37

- Estimamos los proveedores de uruguay y clientes de Perú

Join - Ejemplo cardinalidad con histogramas

- Usamos $n(\text{otros}) / V(\text{otros})$ para la estimación
- Quitamos de “otros” lo que pusimos en cada país y restamos la variabilidad de otros también
 - El n y el V de la tabla deben ser los originales

	Argentina	Uruguay	Brasil	Perú	Otros	V(Otros)
Cientes.pais	95,000	3,000	1,000	21	979	46
Proveedores.pais	8,000	3	1,000	900	97	36

- Ahora tenemos $3,000 * 3$ filas de uruguay y $21 * 900$ de Perú

Join - Ejemplo cardinalidad con histogramas

- Para estimar cómo se combinan los otros, usar la fórmula sin histograma
 - En este caso $979 * 97 / \text{máx}(46, 36)$
- Finalmente se suman los de los países y los de otros

	Argentina	Uruguay	Brasil	Perú	Otros	V(Otros)
Clientes.pais	95,000	3,000	1,000	21	979	46
Proveedores.pais	8,000	3	1,000	900	97	36

Join - Ejemplo cardinalidad con histogramas

- Argentina: 760,000,000
- Uruguay: 9,000
- Otros: 2,065
- Brasil: 1,000,000
- Perú: 18,900

	Argentina	Uruguay	Brasil	Perú	Otros	V(Otros)
Cientes.pais	95,000	3,000	1,000	21	979	46
Proveedores.pais	8,000	3	1,000	900	97	36

- $n(\text{Clientes} \bowtie_{\text{clientes.pais} = \text{Proveedores.pais}} \text{Proveedores}) = 761,029,965$

Join - Cantidad de bloques

- Para estimar el factor de bloque de una fila de $R \bowtie S$ podemos pensar que el espacio ocupado es igual al espacio ocupado por una fila de R más el de una fila de S
- Cada fila de R ocupa $1/F(R)$ bloques y cada fila de S , $1/F(S)$ bloques
- El inverso de los factores de bloque se puede sumar

$$1/F(R \bowtie S) = 1/F(R) + 1/F(S)$$

Join - Cantidad de bloques

- Es importante redondear hacia abajo
 - Si entran 2 filas y media por bloque, en realidad entran dos

$$F(R \bowtie S) = \lfloor 1 / (1/F(R) + 1/F(S)) \rfloor$$

- Teniendo el nuevo factor de bloque y la cardinalidad, podemos saber cuántos bloques ocupa

$$B(R \bowtie S) = n(R \bowtie S) / F(R \bowtie S)$$

Join - Ejemplo cantidad de bloques

- $F(\text{Clientes}) = 10$ y $F(\text{Proveedores}) = 4$
- $F(\text{Clientes} \bowtie \text{Proveedores}) = \lfloor 1 / (0,35) \rfloor = 2$

$n(\text{Clientes})$	100,000
$V(\text{país}, \text{Clientes})$	50
$B(\text{Clientes})$	10,000
$n(\text{Proveedores})$	10,000
$V(\text{país}, \text{Proveedores})$	40
$B(\text{Proveedores})$	2,500

- $n(\text{Clientes} \bowtie_{\text{clientes.pais} = \text{Proveedores.pais}} \text{Proveedores}) = 761,029,965$
- $B(\text{Clientes} \bowtie_{\text{clientes.pais} = \text{Proveedores.pais}} \text{Proveedores}) = 380,514,983$

Combinación de operadores

Pipelining

Combinación de operadores

- Muchas consultas requieren de más de un operador de álgebra relacional para ser resueltas
- La entrada de un operador es la salida de otro
 - Usar $n(\text{operador})$, $B(\text{operador})$ y $F(\text{operador})$ para los nuevos cálculos
- Los resultados intermedios pueden almacenarse temporalmente en archivos (agrega $2 * B(\text{operador})$ de costo)
- Otra opción más eficiente es hacer **pipelining**

Pipelining

- En muchos casos el resultado parcial de un operador puede ser procesado por el siguiente operador de la consulta
 - No es necesario tener todos los bloques de $O1(R)$ para calcular $O2(O1(R))$
 - Cuando termine de procesarse $O1$, $O2$ habrá procesado la salida completa de $O1(R)$
- Al no tener que materializar la salida de $O1$, conviene siempre que se pueda hacer pipelining

Pipelining - Selección a la salida de operadores

- Una selección aplicada a la salida de un operador no agrega costo alguno
 - Es “hacer un if” con la condición de la selección y descartar las filas que no la cumplan
- Cada bloque que sale del operador anterior se copia en memoria, descartando las filas que corresponda. Al llenarse este nuevo bloque se pasa al siguiente operador
 - No hay acceso a disco, por ende no hay costo

Pipelining - Proyección a la salida de operadores

- Si la proyección no precisa evitar duplicados, se puede hacer sin costo extra a la salida de cualquier operador
 - Simplemente de cada fila completa recibida en el bloque, se queda con la/s columna/s proyectadas
- Si precisa evitar duplicados, puede ir acumulando las proyecciones en memoria hasta ocupar el espacio disponible
 - Cuando se ocupa todo, se envía a disco ordenado
 - Se hace sort externo, pero se evita la primer lectura de $B(R)$, teniendo un costo menor

Pipelining - Junta a la salida de operadores

- Al recibir un bloque del operador anterior, se puede hacer la junta para todas sus filas, sin requerir los próximos bloques
- El costo es menor porque se ahorra una lectura de $B(R)$
 - Loops anidados: $\lceil B(R) / (M-2) \rceil * B(S)$
 - Único loop: $n(R) * \text{index_scan}(S)$
 - Junta hash: $2 * B(R) + 3 * B(S)$
 - Sort merge: $B(S) + 2 * B(R) * \lceil \log_{M-1}(B(R)) \rceil$
 $+ 2 * B(S) * \lceil \log_{M-1}(B(S)) \rceil$

Pipelining - Ejemplo final

- Calcule el costo de la siguiente consulta que devuelve el id de los clientes con préstamos viejos (y el id del préstamo)

Clientes(id, nombre, apellido, email,)

Prestamos(id, fecha, id_cliente, monto)

$\pi_{\text{clientes.id, Prestamos.id}} (\text{Clientes} \bowtie_{\text{Clientes.id} = \text{Prestamos.id_cliente}} \sigma_{\text{fecha} < '2020-01-01'} (\text{Prestamos}))$

Pipelining - Ejemplo final - Metadatos

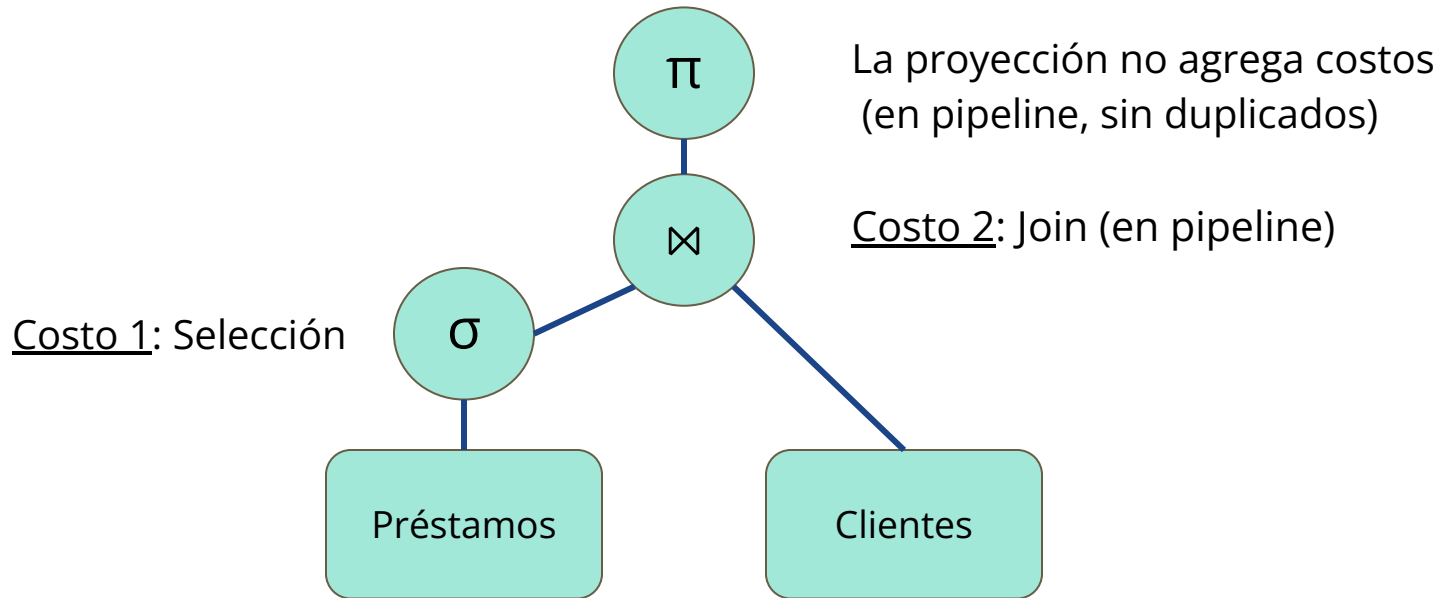
$\Pi_{\text{clientes.id, Prestamos.id}} (\text{Clientes} \bowtie_{\text{Clientes.id = Prestamos.id_cliente}} \sigma_{\text{fecha} < '2020-01-01'} (\text{Prestamos}))$

Clientes		Prestamos	
n(Clientes)	1,000,000	n(Prestamos)	5,000
B(Clientes)	100,000	B(Prestamos)	250
		V(id_cliente, Prestamos)	5,000
H(I(id_cliente, Clientes))	4	H(I(id_cliente, Prestamos))	2

- Los índices no son de clustering
- Un 10% de los préstamos son anteriores a 2020

Pipelining - Ejemplo final - Árbol de consulta

$\pi_{\text{clientes.id, Prestamos.id}} (\text{Clientes} \bowtie_{\text{Clientes.id = Prestamos.id_cliente}} \sigma_{\text{fecha} < '2020-01-01'} (\text{Prestamos}))$



Pipelining - Ejemplo final - Costo 1

- La selección no puede aprovechar el índice
 - Se recorrerán todos los préstamos
 - Sólo se devuelve el 10% que cumple la condición de la fecha
- El costo es el del file scan
 - $C1 = B(\text{Prestamos}) = 250$
- Salen 500 préstamos (un 10% de los que habían) en 25 bloques

Pipelining - Ejemplo final - Costo 2

- Para el join podría usarse el índice
 - Si el costo es menor que recorrer una vez la tabla de clientes, entonces le gana a los otros métodos
- A medida que cada fila de préstamo es devuelta por la selección se hace el index scan
 - Se debe adaptar la fórmula para no considerar el costo $B(R)$
- $C2 = n(\text{selección}) * \text{Index_Scan}$
 $C2 = 500 * (4 + 1) = 2,500$
 - Conviene el loop con índices

Pipelining - Ejemplo final - Costo Final

- La proyección no precisa evitar duplicados
- Se hace en pipeline y no suma costos
 - Luego del join, únicamente nos quedamos con los atributos necesarios
- Costo total = $C1 + C2 = 250 + 2,500 = 2,750$

Resolución de consultas en PostgreSQL

EXPLAIN, índices y pg_stats

Comando EXPLAIN

- El comando de SQL **EXPLAIN** previo a un **SELECT** permite conocer el plan de ejecución de una consulta:
 - La consulta se ejecuta si se usa la opción **ANALYSE**
 - La opción **VERBOSE** da aún más info de la resolución

```
EXPLAIN [ ANALYSE ] [ VERBOSE ]  
SELECT ..... FROM ... ;
```

- Permite conocer si se van a aprovechar índices

Catálogo en PostgreSQL

- Cada motor guarda distinta información de catálogo
- En PostgreSQL tenemos los histogramas en la vista pg_stats

```
SELECT n_distinct      -- Variabilidad
      , most_common_vals -- Valores más frecuentes
      , most_common_freqs -- Frecuencias
FROM pg_stats
WHERE tablename = 'alumnos'
AND attname = 'apellido';
```

Actualización del catálogo

- Cómo y cuándo se actualiza el catálogo depende del gestor
- En PostgreSQL el histograma no se actualiza automáticamente sino que hay que hacerlo con el comando **ANALYSE**

```
ANALYSE [ VERBOSE ]  
[ nombre_tabla [(columna1 [, ..., columnaN])] ]
```

- En vez de recorrer toda la tabla, hace un muestreo haciendo que sea menos costosa la actualización

Índices en PostgreSQL

- Cada motor permite definir ciertas propiedades de los índices
- En PostgreSQL nos interesa el tipo de índice y el operator class

```
CREATE [ UNIQUE ] INDEX nombre_indice
ON nombre_tabla [ USING metodo ]

    ( expresion1 [opclass1]
      [, ..., expresionN [opclassN] ]
    )
[ INCLUDE (columna1 [, ..., columnaN] ) ]
[ WHERE condicion ]
;
```

Btree, hash, gist,
etc..

Define los
operadores
utilizados para la
columna. Ciertos
operadores permiten
usar el índice en
consultas por patrón
con **LIKE**

Bibliografía

Bibliografía

[ELM16] Fundamentals of Database Systems, 7th Edition.

R. Elmasri, S. Navathe, 2016.

Capítulo 17, Capítulo 18

[SILB19] Database System Concepts, 7th Edition.

A. Silberschatz, H. Korth, S. Sudarshan, 2019.

Capítulo 15, Capítulo 16

[CONN15] Database Systems, a Practical Approach to Design, Implementation and Management, 6th Edition.

T. Connolly, C. Begg, 2015.

Capítulo 23

[GM09] Database Systems, The Complete Book, 2nd Edition.

H. García-Molina, J. Ullman, J. Widom, 2009.

Capítulo 15, Capítulo 16

Bibliografía

[PostgreSQL] CREATE INDEX.

<https://www.postgresql.org/docs/current/sql-createindex.html>

[PostgreSQL] EXPLAIN.

<https://www.postgresql.org/docs/current/sql-explain.html>

[PostgreSQL] Operator Classes.

<https://www.postgresql.org/docs/current/indexes-opclass.html>

[PostgreSQL] ANALYZE.

<https://www.postgresql.org/docs/current/sql-analyze.html>