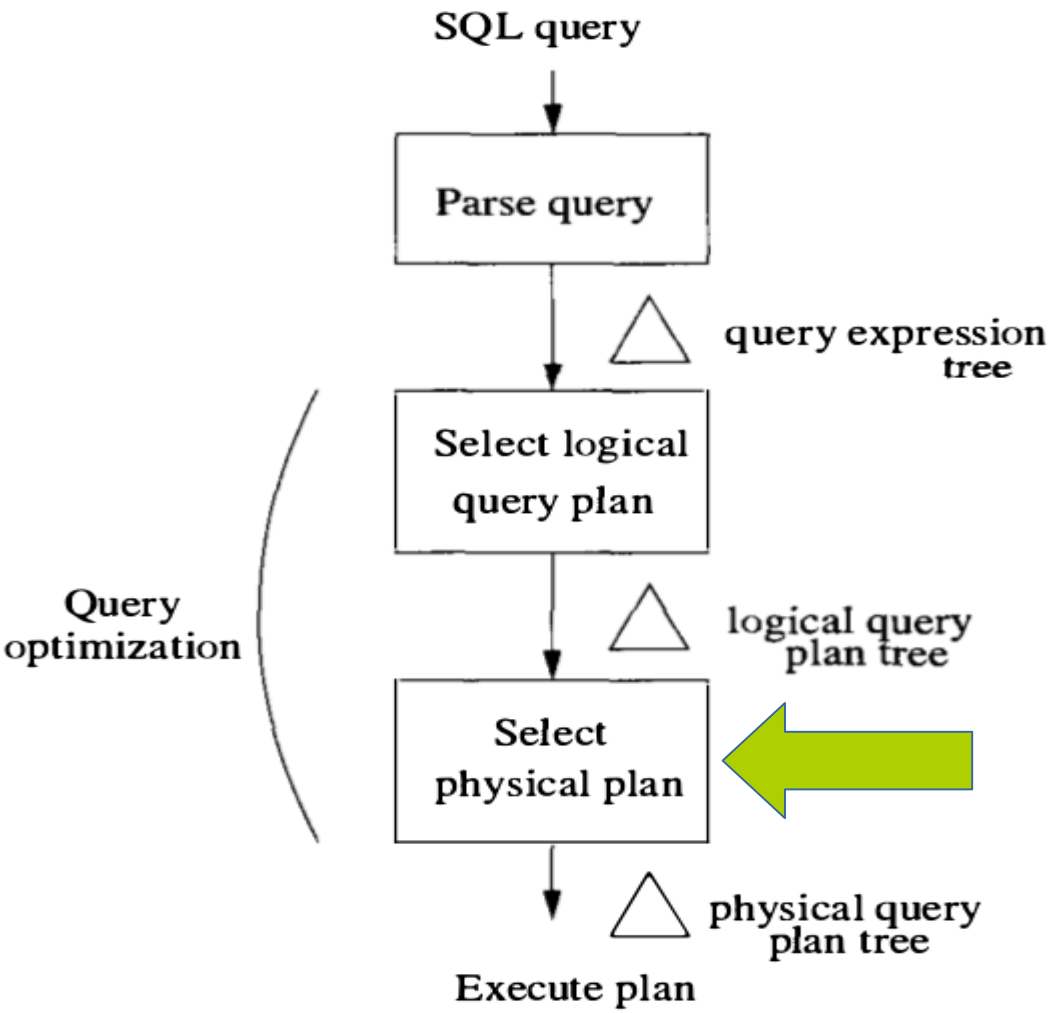
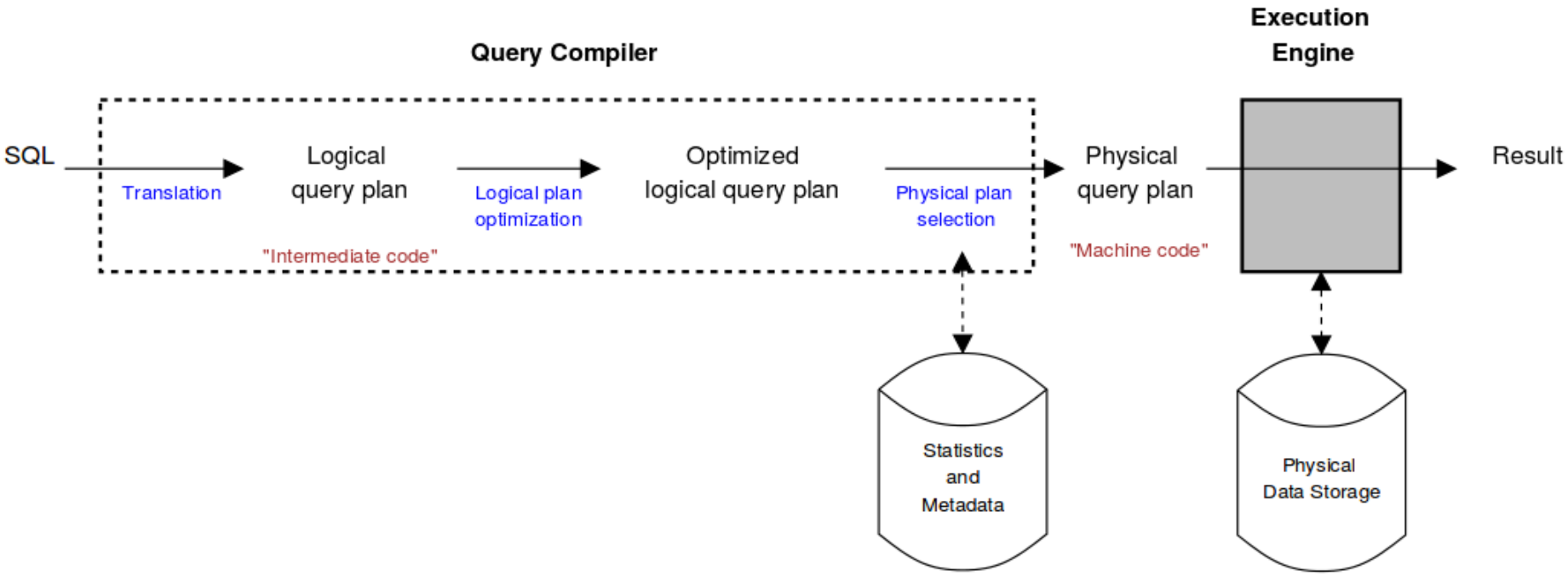
Universidad del Valle

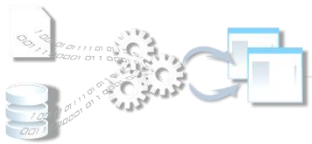
Physical-Query-Plan

Physical-Query-Plan



Physical-Query-Plan





Physical-Query-Plan Operators

Operadores de consulta Físicos

A menudo, los operadores físicos son implementaciones particulares para uno de los operadores del álgebra relacional.

También se necesitan otros operadores diferentes a los del Algebre relacional

Physical-Query-Plan Operators

- Table-Scan

Obtener uno por uno los bloques que contienen las tuplas de R

- Index-Scan

Usar un índice para obtener todas las tuplas de la relación R

- Sort-scan

Toma una relación R y una especificación de los atributos en los que se quiere ordenar , y devuelve la relación R en forma ordenada

Algunas medidas de costo

Parámetros

M : Memoria disponible buffers (Tamaño del buffer = Tamaño del bloque en disco)

Dada una relación R

B(R) Número de bloques de *R*

T(R) Número de tuplas en *R*

V(R,a) : Número de valores distintos para el atributo a en R

number of disk I/Os.

Iteradores

Open()
Inicializa el estado del operador
Establece parámetros tales como la condición de selección

get_next()
Devuelve la siguiente tupla en el resultado y ajusta las estructuras de datos necesarias para permitir las posteriores tuplas a obtener.

close()

Esta función termina la iteración después de obtener todas las tuplas

Iteradores

```
Open() {
     b := the first block of R;
     t := the first tuple of block b;
}

GetNext() {
     IF (t is past the last tuple on block b) {
          increment b to the next block;
          IF (there is no next block)
               RETURN NotFound;
          ELSE /* b is a new block */
               t := first tuple on block b;
     }  /* now we are ready to return t and increment */
     oldt := t;
     increment t to the next tuple of b;
     RETURN oldt;
}

Close() {
}
```

Algunos algoritmos

Forma de Acceso

- sorting-based
- hash-based
- index-based

Dependiendo del costo

- one-pass
- two-pass
- multi-pass (more than 2)

Forma en que trabajan los operadores

- tuple-at-a-time, unary
- full-relation, unary
- full-relation, binary

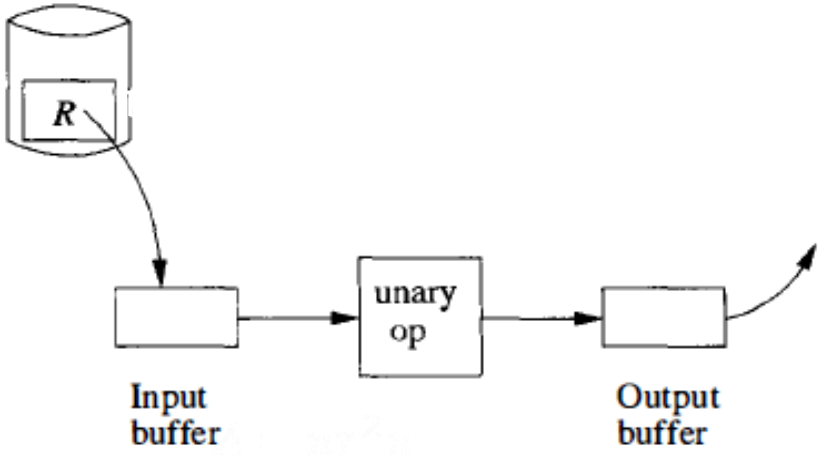
Algoritmos de un paso

One-Pass Algorithms for Database Operations

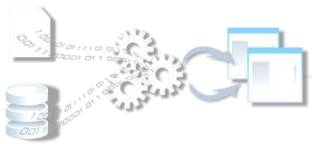
- Sorting-based
- Hash-based
- Index-based

Algoritmos de un paso: Unario

σ, π



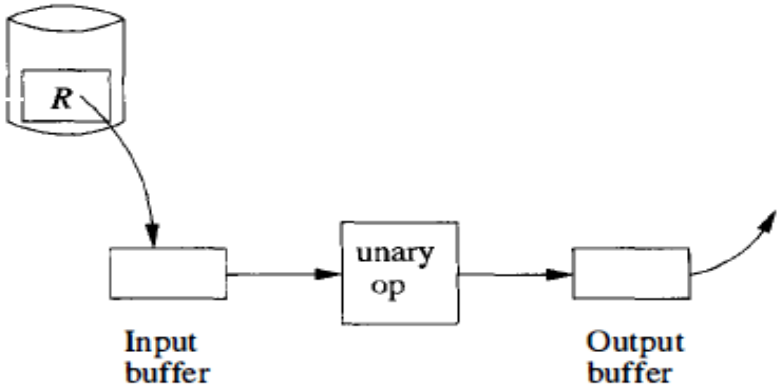
- Leer los bloques de **R** secuencialmente en un buffer de entrada
- Realizar la operación: σ, π
- Mover las tuplas proyectadas o seleccionadas a un búfer de salida



Algoritmos de un paso: Unario

σ, π

Tupla a la vez

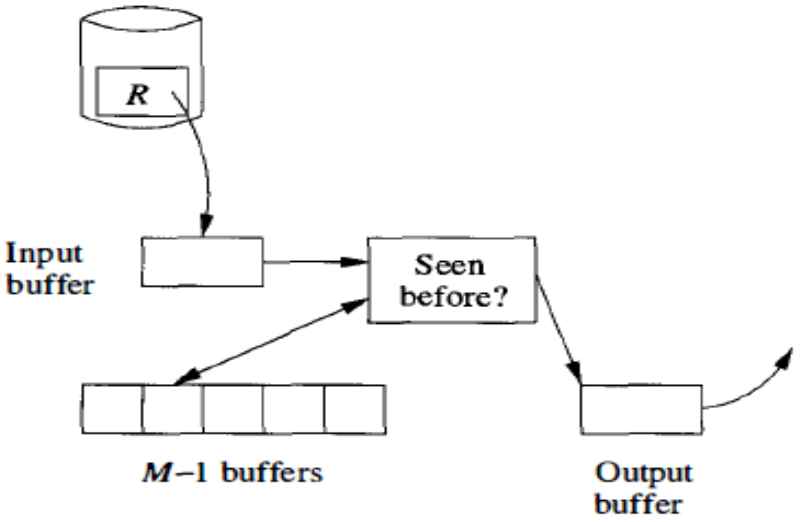


- Leer los bloques de ***R*** secuencialmente en un buffer de entrada
- Realizar la operación: σ, π
- Mover las tuplas proyectadas o seleccionadas a un búfer de salida

✓ Selección de tuplas que satisfacen alguna condición
✓ Usar un índice para la lectura

Algoritmos de un paso: Unario

Eliminación de duplicados



- Mantener en la memoria una copia de cada tupla leída
- Chequear si la tupla fue leída previamente
- Si no fue leída anteriormente, agregarla la buffer de salida

Algoritmos de un paso: Binario

Set Vs Bag

- Unión
- Intersección
- Diferencia

Costo en Disco I/O: $B(R) + B(S)$

Algoritmos de un paso: Binario

- Unión: (S) U (R)

```
Read B(S) into M-1 buffers
Build a search structure
Copy all tuples of S to output
Read each B(R)
    For each t of R
        If t not in S  then
                Add t to outpt
```

Algoritmos de un paso: Binario

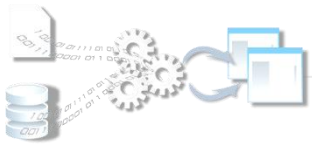
Natural Join

Nested-Loop Join: $R(x,y) \bowtie S(y,z)$

```
FOR each tuple s in S DO
    FOR each tuple r in R DO
        IF r and s join to make a tuple t THEN
            output t;
```

Costo I/O: T (R) x T(S)

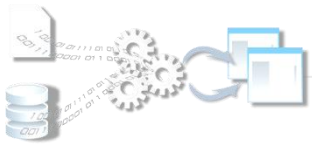
Cómo mejorarlo?



Algoritmos de un paso: Costo

Resumen:

Operators	Approximate M required	Disk I/O
σ, π	1	B
γ, δ	B	B
$\cup$, $\cap$, $-$, $\times$, $\bowtie$	$\min\big(B(R), B(S)\big)$	$B(R) + B(S)$
$\bowtie$	any $M \geq 2$	$B(R)B(S)/M$



Algoritmos de dos pasadas

Two-Pass Algorithms (Sorting)

- ✓ Se utilizan cuando las relaciones son demasiado grandes para colocarlas en la memoria. Basados en ordenamiento.
  - ***Ordenar*** partes de la relación **R** (Merge-Sort, Quick- Sort) .
  - Escribir las partes ordenadas en disco
  - Los datos se vuelven a leer desde el disco para continuar la operación
- ✓ ***¿Por qué sólo dos pasos?***
- ✓ *Se puede extender a más pasos*

Algoritmos de dos pasadas

Eliminación de duplicados con ordenamiento

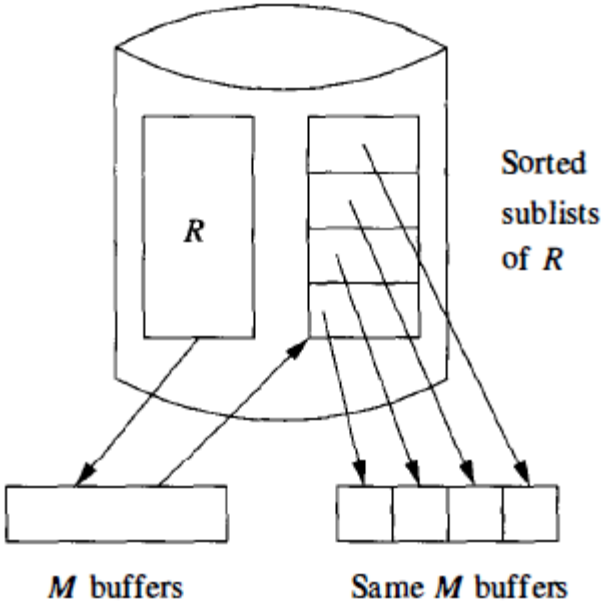
En una primera pasada:

Repetir:

Leer M bloques de R y ponerlos en M
Ordenar estos bloques
Escriba los bloques ordenados en disco (sub-listas ordenadas)

En una segunda pasada

Leer una sub-lista ordenada y colocarla en M
Copiar al buffer de salida cada tupla t
Eliminar todas las demás copias de t



Algoritmos de dos pasadas

Union Usando Ordenamiento (Using Sorting)

Crear una sub-lista ordenada de R en disco
Crear una sub-lista ordenada de S en disco
Colocar cada sub-lista de R y S en M

Repetir
    Buscar la siguiente tup*la* t en todos los buffers
    Copiar t al buffer de salida
    Borrar de los buffer copias de *t*

- Algoritmos para ∩ y -, similares al de la Union

Algoritmos de dos pasadas

Natural Join

$R(X,Y) \bowtie S(Y,Z)$:

Primera pasada:

- Ordenar totalmente R
- Ordenar totalmente S

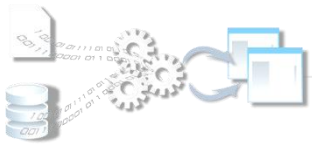
Segunda pasada:

- Leer bloques de R y S, uno a la vez
- Verificar si el atributo de Join se encuentra en R y S

Algoritmos de dos pasadas

Resumen de costo:

Operators	Approximate M required	Disk I/O
γ, δ	$\sqrt{B}$	$3B$
$\cup$, $\cap$, $-$	$\sqrt{B(R)+B(S)}$	$3(B(R)+B(S))$
$\bowtie$	$\sqrt{\max(B(R),B(S))}$	$5(B(R)+B(S))$
$\bowtie$	$\sqrt{B(R)+B(S)}$	$3(B(R)+B(S))$



Algoritmos de dos pasadas

Basados en Índices.

- Algoritmos Basados en índices son especialmente útiles para el operador de selección.
- Otros operadores binarios también utilizan las ventajas de usar índices
- Sin un índice, la selección toma *B(R),* o *T(R)* accesos a disco (I/O's.)
- Si hay un indices el número de I/O's is *B(R)/V(R,a)*

Algoritmos de dos pasadas

Basados en Índices.

Ejemplo:

B (R) = 1,000, T (R) = 20.000, hay un índice en el atributo a y se quiere seleccionar todas las tuplas con a = x.

- Si *R* almacena los datos en cluster y no usa un índice i: 1000 I/O's
- Si *R* no almacena los datos en cluster y no usa un índice : 20,000 I/O's
- Si *V(R,a)* = 100 y datos están cluster, y usan un índice: 1000/100 = 10 I/O's (Promedio)
- Si *V(R,a)* = 20,000 (*a* is a key) y usa un índice: 1 disk I/O

Algoritmos de dos pasadas

Join Basado en Indices.

R (x,y) ⋈ S(y,z), Además hay un índice en el atributo *y*

```
for each block of R
   for each tuple t in the current block
      - use index on S to find tuples of S that match t in the
        attribute(s) Y
      - output the join of these tuples
```

Este algortimo es más eficiente si *R* es más pequeño que *S* y *V(S,y)* es grande

Costo en I/O

- *T(R)T(S) / V(S, Y)*, Datos sin cluster
- *T (R) (max(1 , B (S) / V (S, Y)))*, Datos almacenados en cluster

Algoritmos de dos pasadas

Based on Hashing

Operators	Approximate M required	Disk I/O
γ, δ	$\sqrt{B}$	$3B$
$\cup$, $\cap$, $-$	$\sqrt{B(S)}$	$3\big(B(R)+B(S)\big)$
$\bowtie$	$\sqrt{B(S)}$	$3\big(B(R)+B(S)\big)$
$\bowtie$	$\sqrt{B(S)}$	$\big(3-2M/B(S)\big)\big(B(R)+B(S)\big)$