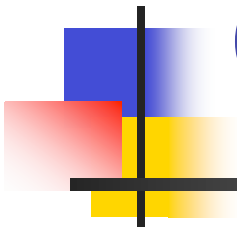


Evaluación y Optimización de Consultas Centralizadas





Por qué usar un DBMS?

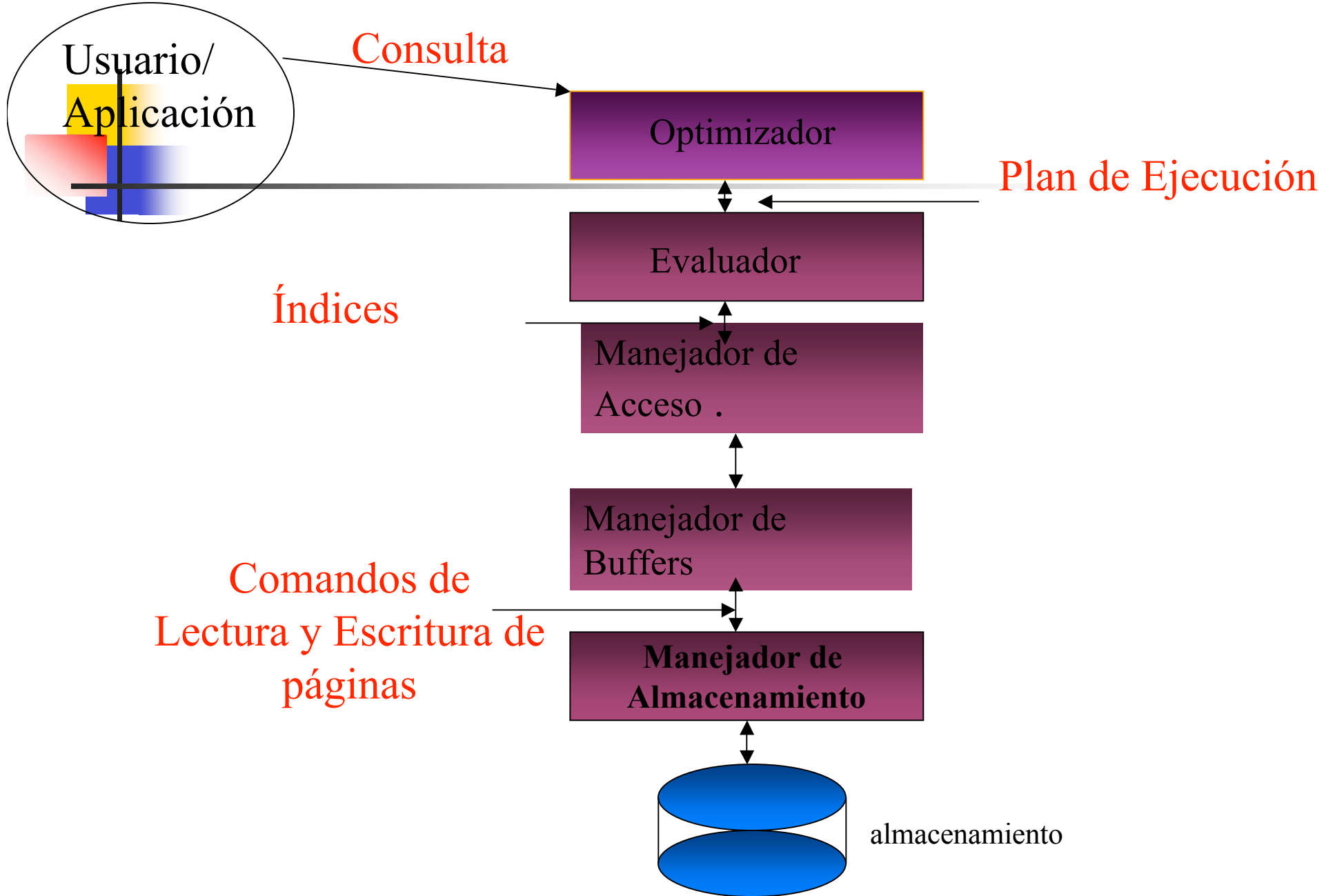
Si cualquier programa puede manejar datos?

- Grandes volúmenes de datos(Giga's, Tera's)
- Los datos son estructurados
- Los datos deben persistir
- Los datos son valiosos
- Los datos deben ser accedidos eficientemente
- Los datos deben ser accedidos concurrentemente
- Los datos deben ser accedidos sólo por personas autorizadas
- Los datos deben ser manipulados por transacciones que llevan la BD de una estado consiste a otro estado consistente.

Funcionalidad de un DBMS



- Manejo de memoria persistente
- Manejo de transacciones
- Manejo de recuperación
- Manejo de integridad
- Separación entre la visión lógica y física de los datos.
 - Lenguajes de interrogación de alto nivel.
 - Procesamiento de consultas eficiente
- Interfaces con lenguajes de programación





Complejidad de Lenguajes Lógicos

- Poder expresivo de un lenguaje L: es el conjunto de funciones que pueden ser escritas en L.
- Requerimiento lenguaje de consultas para ser *relacionalmente completo*:
 - Ser capaz de expresar todas las consultas expresables en álgebra relacional.
 - Cálculo Relacional, SQL sin agregación son *relacionalmente completos*. Se conocen también como lenguajes FO.



Complejidad de Lenguajes Lógicos- Complejidad de los Datos

Sea q una consulta, DB una instancia de un esquema relacional R y A el conjunto de las respuestas de q en DB ,

$q: \text{set}(DB) \rightarrow \text{set}(A)$,

Es decir, q es una correspondencia desde el conjunto de instancias de R al conjunto de posibles respuestas. La medida de complejidad determina la cantidad de veces que instancias de R deben ser consultadas para producir A .



Complejidad de Lenguajes Lógicos- Complejidad de los Datos

Formalmente, el modelo de computación puede ser una máquina de Turing y una instancia BD de un esquema R de tamaño n , se codifica en una cinta $O(n)$. Todas las consultas en la BD pueden ser vistas como máquinas de Turing. En caso que q sea evaluado en tiempo polinomial, se requerirán un número polinomial de pasos en la cinta (BD) para encontrar la respuesta, es decir, $O(n^k)$, donde k es un número positivo.

El conjunto de máquinas que pueden encontrar la respuesta en un número polinomial de pasos se denominan funciones DB-PTIME.



Complejidad de Lenguajes Lógicos- Complejidad de los Datos

- Se dice que un lenguaje L es DB-PTIME si cada función que pueden expresar, se computa en tiempo polinomial en función de n.
- Se dice que un lenguaje L es DB-PTIME complete, si L es DB-PTIME y L puede expresar todas las funciones que son DB-PTIME computables.
- Los Lenguajes FO son DB-PTIME computables. Técnicas sofisticadas de optimización y evaluación son usadas por los DBMS para mantener los exponentes y coeficientes de $O(n^k)$ bajos.
- Los Lenguajes FO no son DB-PTIME complete porque existen funciones, por ejemplo, la clausura transitiva, que son DB-PTIME, que no pueden ser expresadas en lenguajes FO.



Optimización y Evaluación

- Optimización: proceso mediante el cual se identifica una estrategia eficiente de ejecutar una consulta.
- Evaluación: mecanismo mediante el cual una consulta es ejecutada.

Objetivo: Técnicas sofisticadas de optimización y evaluación son usadas por los DBMS para mantener los exponentes y coeficientes de $O(n^k)$ bajos.

Recordar que las consultas FO acíclicas son BD-PTIME. Mientras que las consultas cíclicas pueden requerir BD-EXPTIME.

Ejemplo de una consulta Cíclica-Tablas

R_i

A _i	A _{i+1}
0	A
0	B
1	A
1	B
A	0
A	1
B	0
B	1

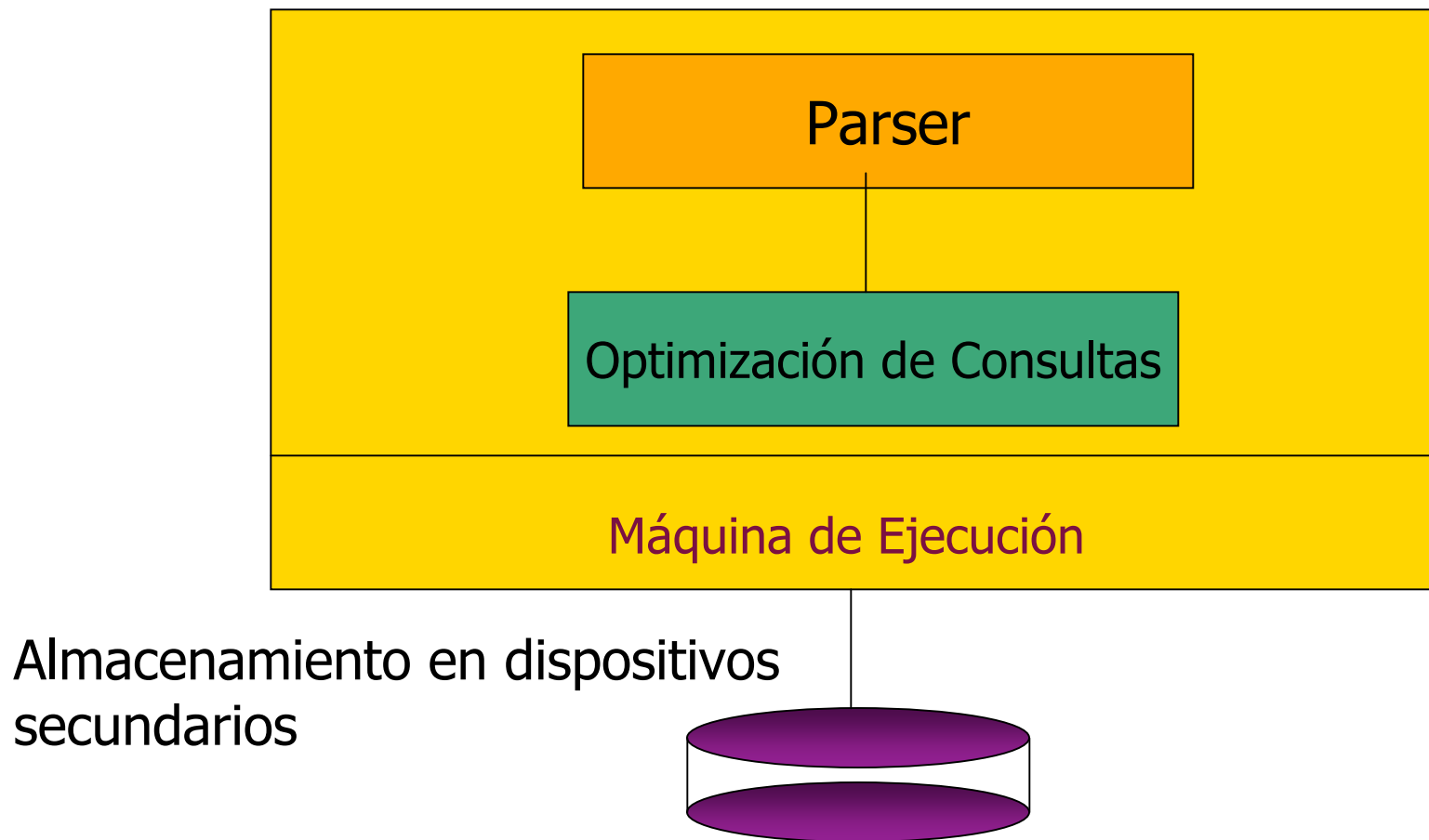
R_n

A _n	A ₁
0	A
0	B
1	A
1	B
A	0
A	1
B	0
B	1

Consulta:

R₁ Join R₂ Join...Join R_n

Arquitectura





Diferentes Costos de Evaluación

- **Estadísticas:**
 - **Producto**
 - Cuatro millones de tuplas
 - Diez mil valores diferentes en el atributo **Categoría**.
 - Tuplas uniformemente distribuidas en los valores del atributo **Categoría**.
 - Cada categoría es producida por un único fabricante.
 - **Fabricante**
 - Dos mil tuplas.



Diferentes Costos de Evaluación

- Tamaño en tuplas de T1:
 - 4.000.000 tuplas
- Tamaño en tuplas de T2:
 - 400 tuplas
- Tamaño en tuplas de T3:
 - 1 tuplas
- Se puede hacer algo mejor?



Diferentes Costos de Evaluación

- **Producto(Nombre,Precio,Categoria,Fabricante)**
- **Fabricante(Fabricante,RazonSocial,NumeroEmpleados)**

“El número de empleados de las empresas que fabrican productos de la categoría *chocolate*”

- $T'1: \sigma_{(Categoria="chocolate")}(Producto)$
- $T'2: \pi_{Fabricante}(T'1)$
- $T'3: \pi_{NumeroEmpleados}(T'2 \text{ Join Fabricante})$

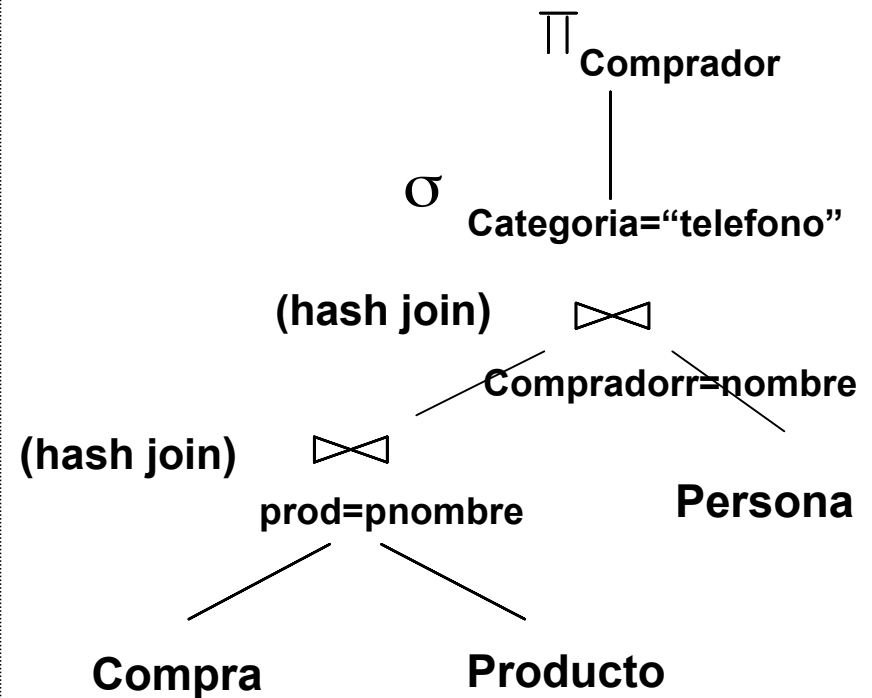
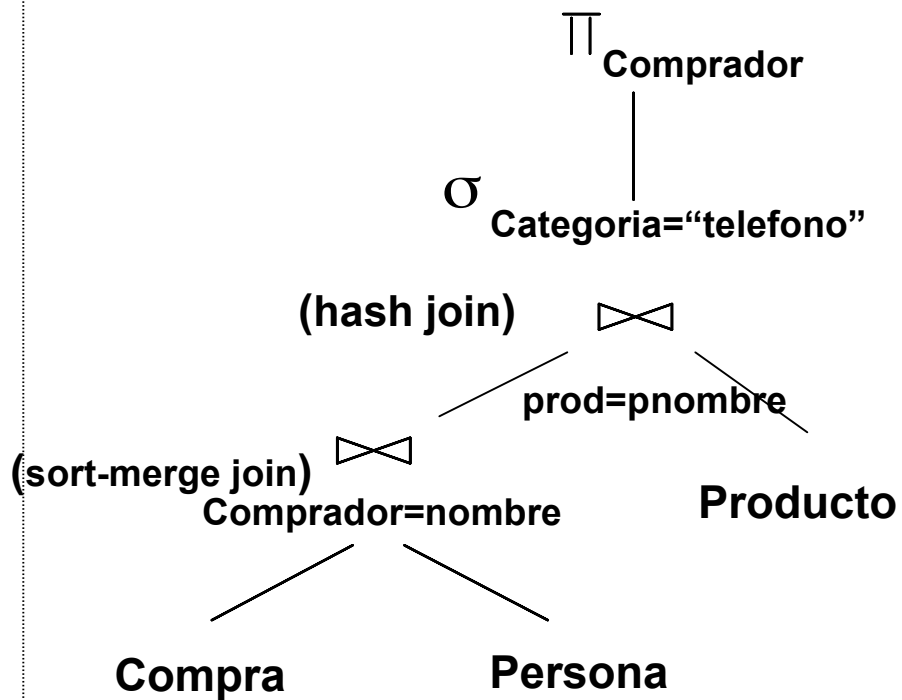


Diferentes Costos de Evaluación

- Tamaño en tuplas de $T'1$:
 - 400 tuplas
- Tamaño en tuplas de $T'2$:
 - 1 tuplas
- Tamaño en tuplas de $T'3$:
 - 1 tuplas

Planes de Ejecución

Dar los nombres de las personas que han comprado un producto en la categoría de teléfonos



Existen muchas maneras de evaluar una consulta de SQL.

Optimización de Consultas

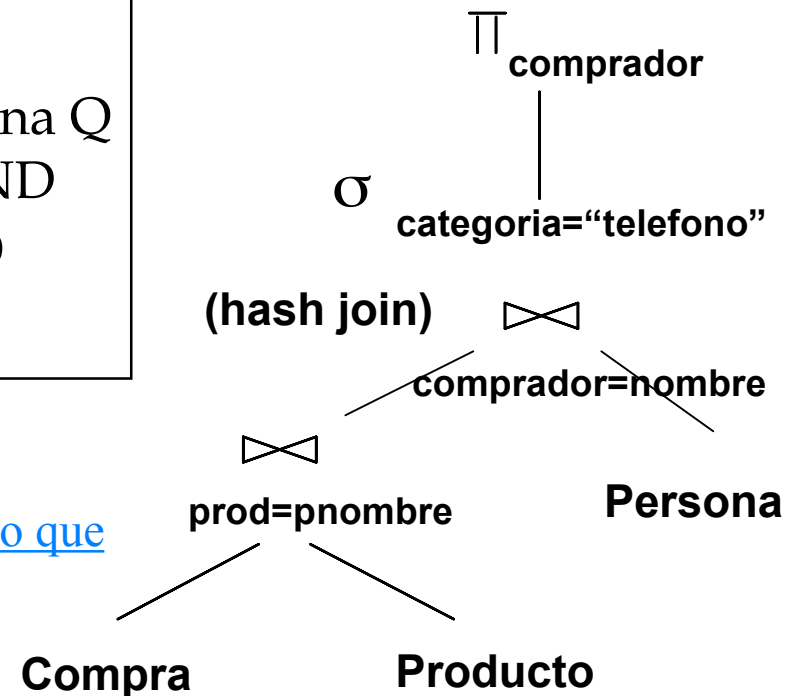
Objetivo:

Consulta en SQL

```
SELECT C.comprador
FROM Producto P, Compra C, Persona Q
WHERE P.Categoria="Telefono" AND
C.comprador=Q.nombre AND
C.prod=P.pnombre
```

Plan: Árbol de operadores del álgebra relacional
donde cada operador está anotado con el algoritmo que
lo implementa

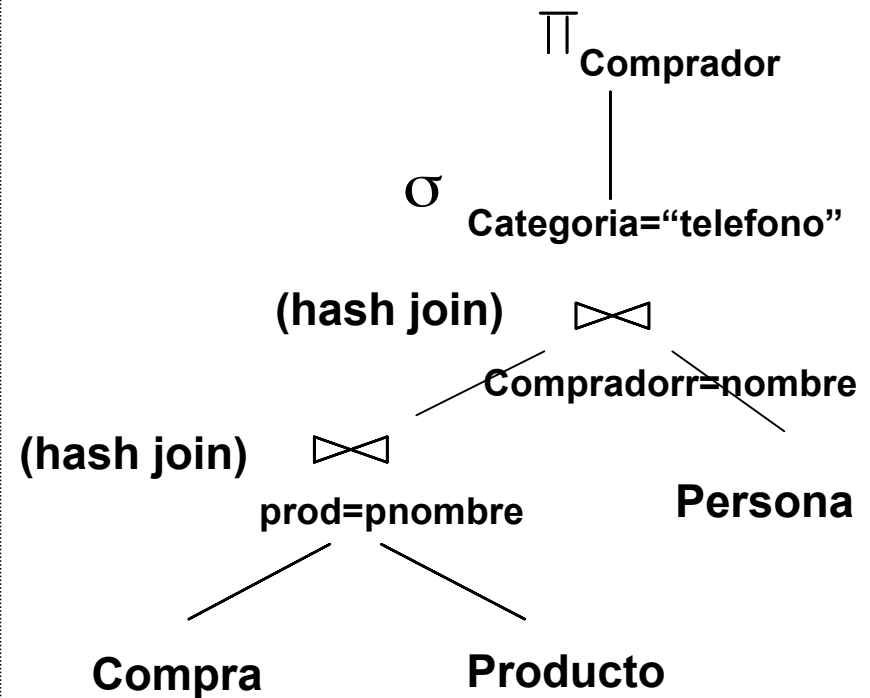
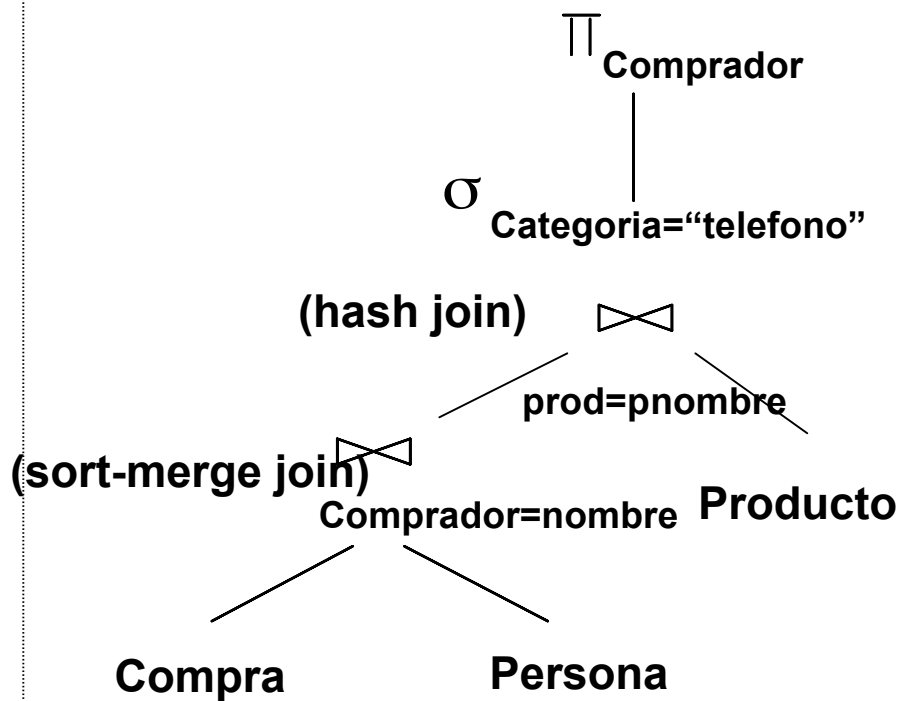
Plan de Ejecución:



Idealmente: se desea conseguir el mejor. **En la práctica:** se evitan los peores

Planes de Ejecución

Dar los nombres de las personas que han comprado un producto en la categoría de teléfonos



Existen muchas maneras de evaluar una consulta de SQL.

Optimización de Consultas

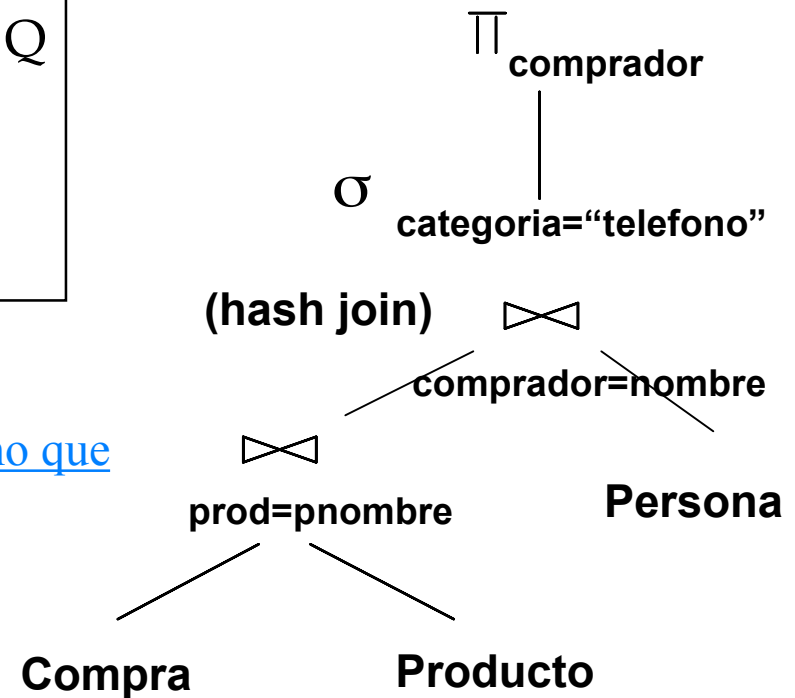
Objetivo:

Consulta en SQL

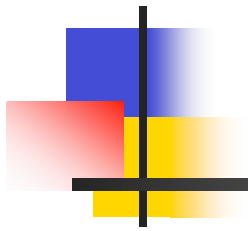
Plan de Ejecución:

```
SELECT C.comprador
FROM Producto P, Compra C, Persona Q
WHERE P.Categoria="Telefono" AND
C.comprador=Q.nombre AND
C.prod=P.pnombre
```

Plan: Árbol de operadores del álgebra relacional
donde cada operador está anotado con el algoritmo que
lo implementa



Idealmente: se desea conseguir el mejor. **En la práctica:** se evitan los peores



Almacenamiento Secundario



Archivos de Registros

- Una página o bloque es la unidad de transferencia entre memoria principal y memoria secundaria.
- Archivos: una colección de páginas cada una conteniendo un conjunto de registros. Debe soportar:
 - insertar/borrar/modificar un registro
 - Leer un registro particular (especificando *record id*)
 - scan todos los registros.



Tipos de Archivos

- Heap files: registros almacenados en el orden en que son insertados.
- Archivos Ordenados: los registros son ordenados haciendo uso de alguna clave.
- Archivos Hashed:
 - Colección de buckets. Bucket = *página primaria más cero o más páginas de overflow*.
 - *Función Hashing h*: $h(r) = b$, donde b es el bucket donde se encuentra r.



Modelo de Costo

Como una buena aproximación se ignora el costo de procesamiento en memoria principal

- **B:** Número de páginas.
- **R:** Número de registros por página.
- **D:** tiempo promedio de leer un bloque.
- Se ignora el beneficio de tener pre-cargadas las páginas en memoria.



Tipos de Archivos

- **Heap Files:**
 - Las inserciones siempre se hacen al final.
- **Archivos Ordenados:**
 - El archivo se debe compactar después de eliminaciones.
 - Selecciones en el campo de ordenamiento.
- **Archivos Hashed:**
 - No buckets de overflow
 - 80% de ocupación, es decir, el número de páginas requerido para almacenar el archivo es 1.25B.



Costo de las Operaciones

	Heap File	Archivo Ordenado	Archivo Hashed
Scan todos los regs	BD	BD	1.25 BD
Búsqueda igualdad	0.5 BD	$D \log_2 B$	D
Búsqueda por rango	BD	$D (\log_2 B + \# \text{ de páginas que hacen match})$	1.25 BD
Inserción	2D		
Eliminación	Costo(Busqueda) +D	Costo(Busqueda) +BD	Costo(Busqueda) +D



Indices

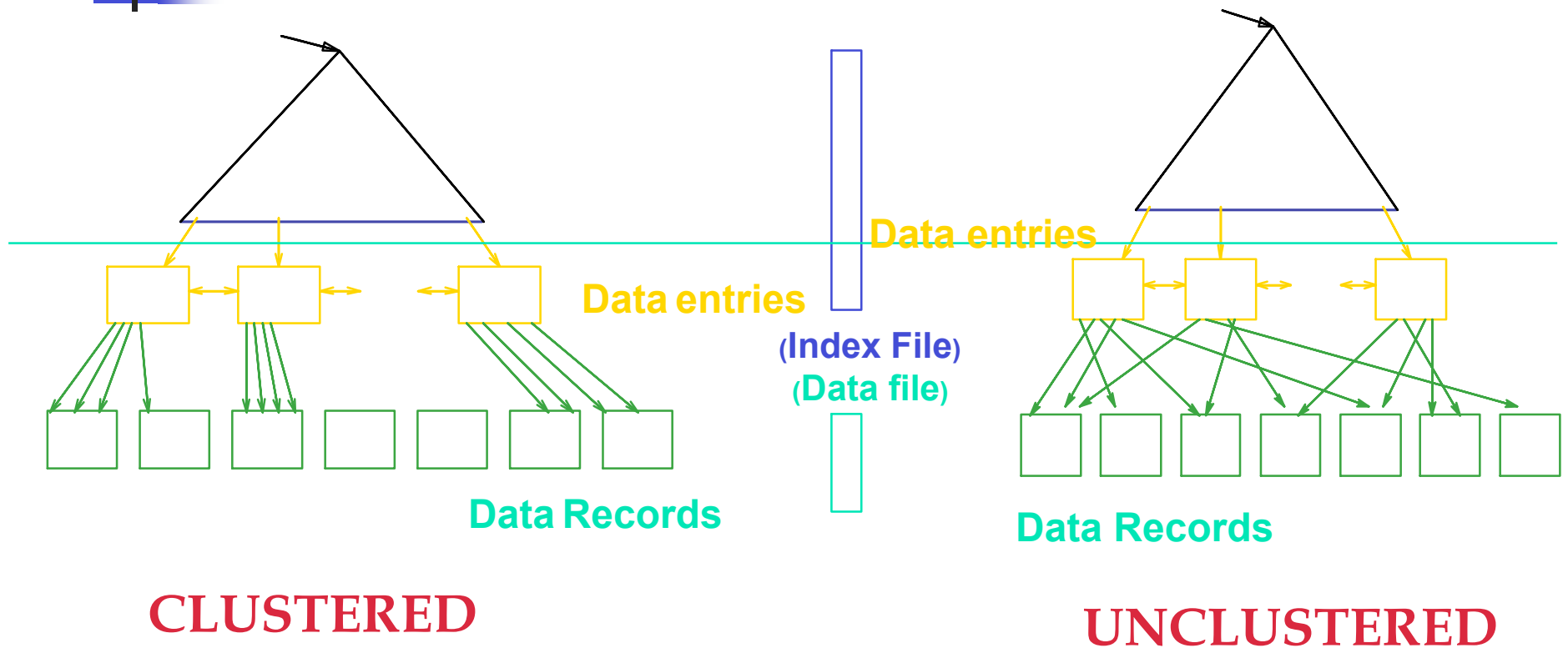
- Un índice acelera la búsqueda en un archivo de datos. Permite acceso directo.
- Un índice contiene una colección de *data entries*.

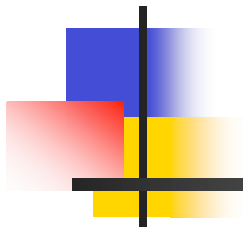


Clasificación de los índices

- *Primarios vs. secundarios*: si la clave de búsqueda contiene una clave, entonces es primario.
- *Clustered vs. unclustered*: si el orden de los registros de datos es el mismo o cercano al orden de los data entries, entonces se llama índice clustered.

Clustered vs. Unclustered Index





Evaluación de Consultas



TÉCNICAS DE EVALUACIÓN DE LOS OPERADORES RELACIONALES

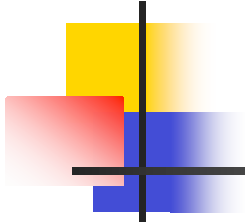
- SELECCIÓN.
- PROYECCIÓN.
- JOIN.



SELECCIÓN

- Considere la siguiente consulta q :
 - Select Est.Carnet From EST
Where Est.Nombre="Luis Perez"
- Maneras de implementar la consulta q :
 - Haciendo un recorrido de la relación completa y seleccionando aquellas que satisfagan la condición.
 - **Si el archivo contiene 1000 páginas, el costo de esta operación es 1000 I-O's.**
 - Si existe un índice en la tabla **EST** sobre el atributo **Nombre**, hacer uso del índice para identificar los registros que satisfacen q .
 - **Si el índice es un B+-tree clustered, el costo de esta operación es $H + P$ I-O's. Donde H es el numero de niveles del árbol y P es el número de páginas que ocupan los registros que satisfacen q .**

TÉCNICAS PARA EVALUAR PROYECCIONES:



Técnica I (Proyecciones Basadas en Ordenamientos):

- 1) Recorrer el archivo y producir el conjunto de tuplas que contienen los atributos deseados.
- 2) Ordenar las tuplas haciendo uso de todos los atributos como clave de ordenamiento.
- 3) Recorrer el resultado, comparando las tuplas adyacentes y descartar los duplicados.

Costo: $M + T + O(T \log T) + T$, donde M número de páginas del archivo, T número de páginas del resultado temporal



TÉCNICAS PARA EVALUAR PROYECCIONES:

Técnica II (Proyecciones Basadas en Hashing): Se disponen de B páginas

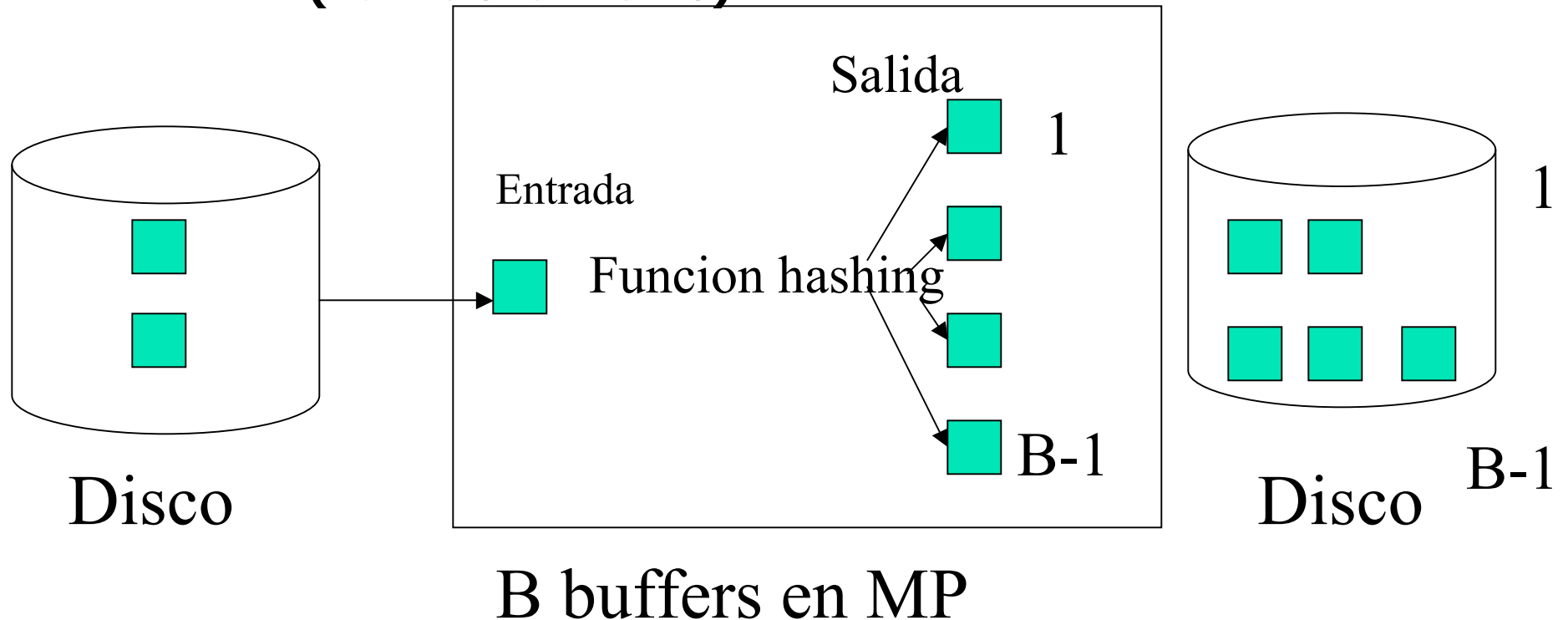
FASE I (Particionamiento), 1 página para entrada y $B-1$ páginas para salida.

- **Para cada tupla en la página de entrada,**
 - **Descartar los atributos no proyectados.**
 - **Crear $B-1$ particiones en disco.**
 - **Aplicar una función de hashing a la combinación de los atributos proyectados. La función debe distribuir uniformemente las tuplas en $B-1$ páginas.**
 - **Vaciar el contenido de cada página de salida R_i en la partición P_i**

TÉCNICAS PARA EVALUAR PROYECCIONES:

Técnica II (Proyecciones Basadas en Hashing): Se disponen de B páginas

FASE I (Particionamiento)





TÉCNICAS PARA EVALUAR PROYECCIONES

Técnica II (Proyecciones Basadas en Hashing): Se disponen de B páginas

FASE II (Eliminación de Duplicados)

Por cada partición P_i ,

- **Leer cada página de la partición P_i , y aplicar una segunda función de hashing h_2 a la combinación de todos los campos, e insertar el resultado en una tabla de hashing en memoria principal. Para cada colisión comparar los valores. Descartar duplicados.**
- **Después que la partición completa ha sido leída, escribir las tuplas en la tabla de hashing en disco.**



TÉCNICAS PARA EVALUAR PROYECCIONES

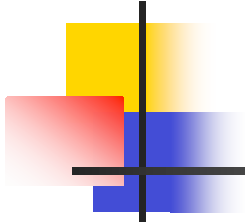
Costo Técnica II (Proyecciones Basadas en Hashing):

Fase I: $(M + T)$ I/-O's, donde M es el número de páginas de la tabla y T es el número de páginas resultado de la proyección

Fase II: T.

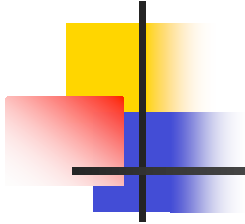
Costo Total: $M + 2T$

TÉCNICAS PARA EVALUAR EL JOIN



- **Nested-Loop Join.**
- **Block Nested-Loop Join.**
- **Index Nested-Loop Join.**
- **Sort-Merge Join.**
- **Hash Join.**

NESTED-LOOP JOIN: A Join B



For each tupla a en A do

For each tupla b en B do

if $a_i = b_i$ then add (a, b) to result

M: es el número de páginas en A .

P_a : número de tuplas de A en una página.

N : número de páginas en B .

P_b : número de tuplas de B en una página.

Costo: $M + P_a * M * N$



BLOCK NESTED-LOOP JOIN: A Join B

Hip: Existe suficiente memoria principal para almacenar A y dos buffers extra.

For each bloque de T-2 paginas de A do

For each pagina de B do{

for all matching in-memory tuples a en A and b en B

add (a,b) to result }

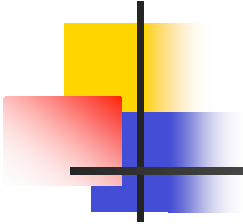
M es el número de páginas en A.

N número de páginas en B.

T número de páginas en memoria principal disponibles

Costo: $M + \text{techo}(M/T-2) * N$

INDEX NESTED-LOOP JOIN: A Join B



Hip: Existe un índice para B sobre los atributos usados en el Join.

For each tupla a en A do

For each tupla b en B where $a_i = b_i$

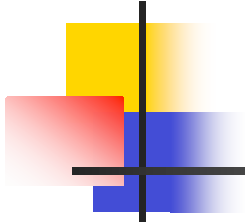
add (a, b) to result

M es el número de páginas en A.

N número de páginas en B.

T número de páginas en memoria principal disponibles

Costo: $M + \text{costo}(B)$.



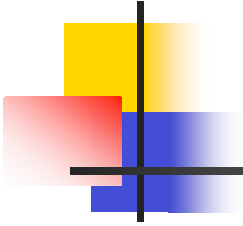
Si se tiene un clustered B+-tree, $\text{Costo}(B)=4+1$

**Si se tiene un unclustered B+-tree,
 $\text{Cost}(B)=4+B\text{-matching-tuple}$**

Si se tiene clustered hashing, $\text{Cost}(B)=2+1$

Si se tiene unclustered hashing, $\text{Costo}(B)=2_B\text{-matching-tuples}$

SORT-MERGE JOIN: A Join B



- *Ordenar las relaciones A y B por los atributos en el Join.*
- *Mezclar las tablas obtenidas en el punto anterior.*

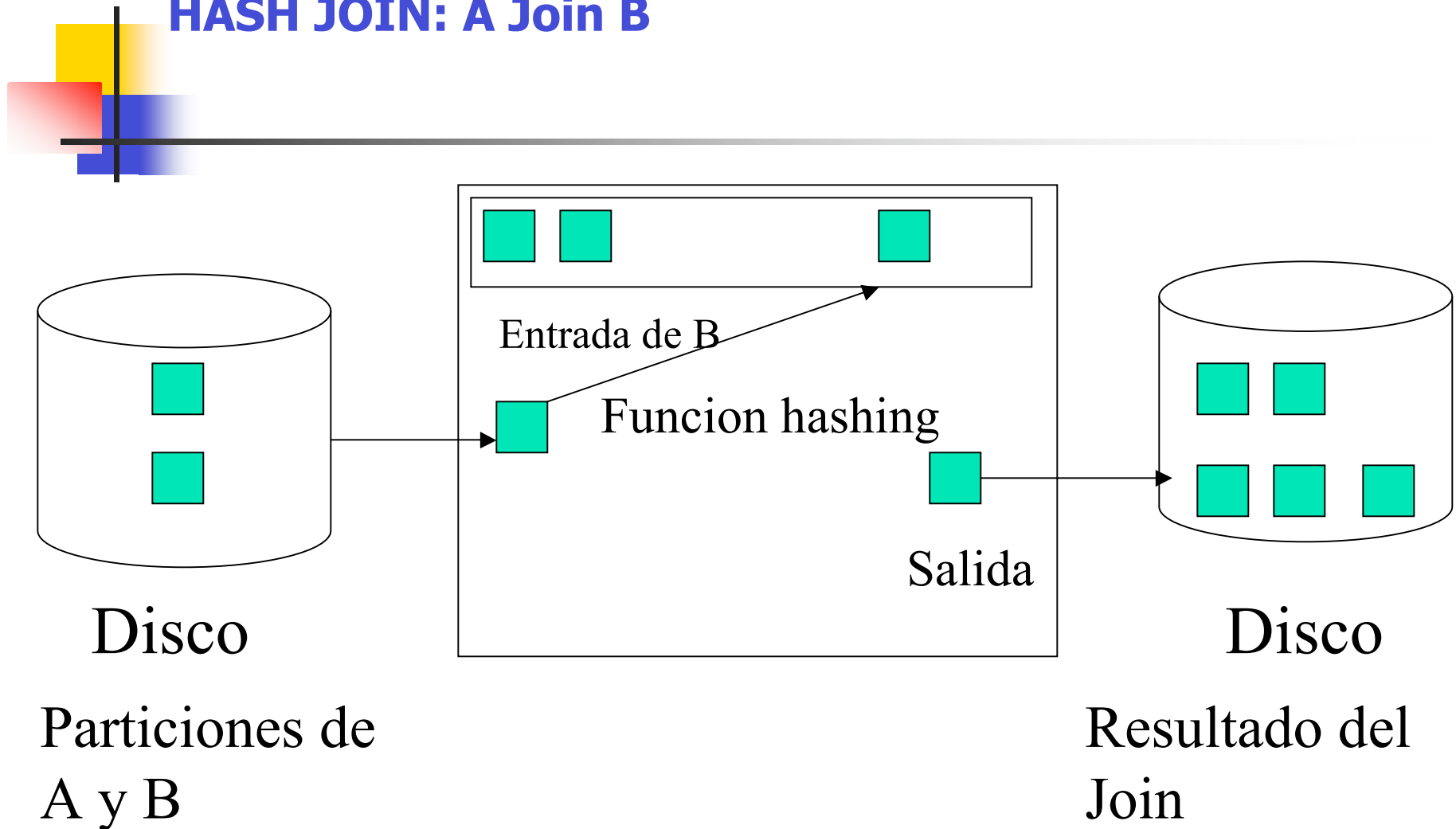
M es el número de páginas en A.

N número de páginas en B.

Costo de ordenar las tablas: $O(M \log M) + O(N \log N)$

Costo de Mezclar las tablas ordenadas: $M + N$

HASH JOIN: A Join B



La idea es aplicar la misma función de hashing h sobre el atributo de join, a las tuplas de ambas relaciones.

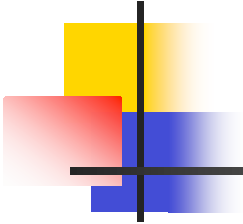


HASH JOIN: A Join B

FASE DE PARTICIÓN:

1. *For each tuple a en A*
 1. *Add a to the page $h(a)$.*
2. *For each tuple b en B .*
 1. *Add b to page $h(b)$*

HASH JOIN: A Join B



FASE DE PRUEBA.

1. *For $L=1, \dots, k$*

1. *For each particion A_l de A ,*

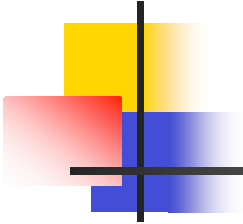
1. *Crear una tabla de hashing en MP, usar una funcion de hashing h_2 .*

2. *For each tuple b en particion B_l de B*

1. *For each tupla a de A en el bucket $h_2(b)$*

1. *If $a_i=b_i$, output (a,b)*

COSTO HASH JOIN: A Join B

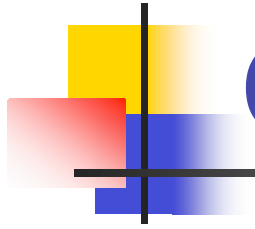


FASE DE PARTICION

$2(M + N)$

FASE DE PRUEBA

$M + N$



Optimización de Consultas:

- **Función objetivo a optimizar:** reducir el tamaño de los resultados intermedios.
- **Basadas en Heurísticas.**
 - Se hace uso de equivalencias entre expresiones del álgebra relacional.
- **Basada en Costos.**
 - Se usa el costo estimado de la ejecución de cada operador.



Optimización Basada en Heurísticas

- Se identifica un **árbol canónico** para una consulta Q.
- Se aplican reglas de equivalencia entre operadores del álgebra relacional, que permiten minimizar el tamaño de los resultados intermedios.
 - **Heurística: empujar lo más abajo en el árbol las selecciones y proyecciones.**

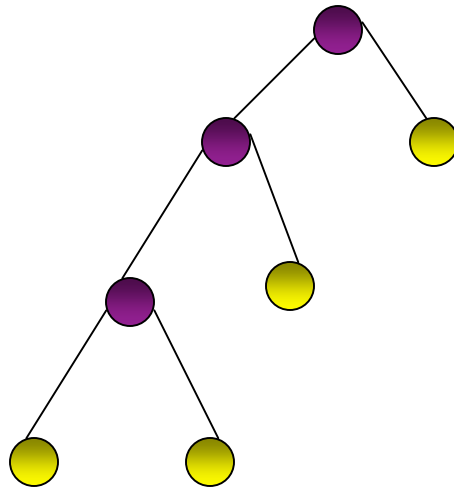


Árbol de Lógico

- Cada hoja del árbol representa una tabla.
- Cada nodo del árbol operador del álgebra relacional.
- Cada nodo del árbol tiene a lo más dos hijos.

Árbol de Lógico Lineal Izquierdo

- Árbol lógico tal que
 - Cada nodo tiene como hijo derecho una tabla.





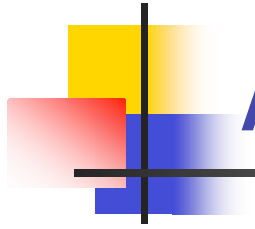
Árbol de Lógico Canónico

- Dada una expresión de SQL
 - **Select** LISTA_ATT
From LISTA_TABLAS
Where Condiciones
- **Árbol lógico lineal izquierdo** tal que:
 - La raíz del árbol corresponde a un nodo unario para el operador proyección de todos los atributos que aparecen en LISTA_ATT. El hijo de este nodo es una sub-árbol canónico de selección.
 - Un sub-árbol canónico de selección, es un árbol unario cuyo nodo raíz corresponde a la selección de todas las condiciones que aparecen en Condiciones. El hijo de este nodo es una sub-árbol canónico de producto cartesiano.
 - Un sub-árbol de producto canónico cartesiano es un árbol binario:
 - Cada nodo corresponde a un producto cartesiano.
 - Hijo derecho es una tabla en LISTA_TABLAS
 - Hijo izquierdo es una tabla en LISTA_TABLAS o un sub- árbol canónico cartesiano



Propiedades de los Operadores del Álgebra Relacional

- Cascada de Selecciones.
- Commutatividad de Selecciones.
- Cascada de Proyecciones.
- Commutatividad/Asociatividad del Producto Cartesiano/Join.
- Distribución de selecciones/proyecciones.
- Commutatividad de selecciones y proyecciones.
- Combinación de la selección y el producto cartesiano.



Árbol de Ejecución

- Árbol de lógico donde cada nodo está anotado con el operador físico con el que se evaluará el operador lógico correspondiente.

Estimación del Costo de un Plan de Ejecución

- Para cada plan se debe estimar:
 - El **costo** de cada operación en un árbol de ejecución.
 - Depende de la cardinalidad de la entrada.
 - El **tamaño del resultado** para cada operación en el árbol!
 - Uso de la información sobre las relaciones de entrada.
 - Asumir para las selecciones y joins independencia de los predicados.
 - Propagar las estimaciones hacia arriba en el árbol.
- Discutiremos el modelo de estimación del **Sistema R**.
 - Muy inexacto pero trabaja en la práctica.



Estadísticas y Catálogos

- Se requiere información sobre las relaciones e índices usados.
Catálogos típicamente contienen al menos:
 - # tuplas (NTuplas) y # páginas (NPáginas) para cada relación.
 - # valores diferentes (NKeys) y Npáginas por índice.
 - Altura del índice, valor más alto y más bajo (Bajo/Alto) para cada índice.
- El catálogo mantiene la información sobre las relaciones.
- Los catálogos se actualizan periódicamente.
 - Actualizar las estadísticas cada vez que exista un cambio es muy caro.
- Alguna información más detallada, por ejemplo histogramas, se almacenan.

Catálogos

Ejemplo:

Puede almacenar información acerca de los atributos de las relaciones en una relación:

Attribute_Cat(attr_name:string,
rel_name:string, type:string, position:string)

Si la base de datos contiene dos relaciones:

Students(sid:string, name:string,
login:string, age:integer, gpa:real)

Faculty(fid: string, fname:string, sal:real)

<i>attr_name</i>	<i>rel_name</i>	<i>type</i>	<i>position</i>
attr_name	Attribute_cat	string	1
rel_name	Attribute_cat	string	2
type	Attribute_cat	string	3
position	Attribute_cat	integer	4
sid	Students	string	1
name	Students	string	2
login	Students	string	3
age	Students	integer	4
gpa	Students	real	5
fid	Faculty	string	1
fname	Faculty	string	2
sal	Faculty	real	3

Estimación del Tamaño y Factores de Reducción/Selectividad

- Un bloque de query :
SELECT attribute list
FROM relation list
WHERE $term_1$ AND ... AND $term_k$
- **Factor de Selectividad:** es la fracción de los datos que satisfacen el predicado.
- **Computar la selectividad de una expresión filtro:**
 - Determinar la selectividad de un predicado atómico usando estadísticas.
 - Derivar la selectividad de una expresión booleana a partir de (a).



Estimación del Tamaño y Factores de Reducción

- **Cardinalidad:** Máximo número de tuplas en el resultado es el resultado del producto de las cardinalidades de las relaciones en la cláusula FROM
- **Factor Reducción (FR)** asociado con cada término refleja el impacto del término en reduciendo el tamaño del resultado. $\text{Cardinalidad del Resultado} = \text{Max \# tuplas} * \text{producto de todo FR's.}$



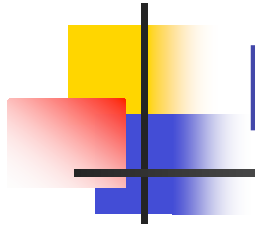
Factores de Selectividad para Predicados Atómicos

- **Consideración Implícita** términos son independientes.
- Término $col=valor$ tiene FS $1/NKeys(I)$, *se tiene un índice sobre col.*
- Término $col1=col2$ tiene FS $1/MAX(NKeys(I1), NKeys(I2))$ *se tiene un índice sobre col1 e índice sobre col2.*
- Término $col>valor$ tiene FS
 - $(Alto(I) - value) / (Alto(I) - Bajo(I))$



Factores de Selectividad para Expresiones Booleanas

- P1 and P2
 - $FS(P1 \text{ and } P2) = FS(P1) * FS(P2)$
- Not(P1)
 - $FS(\text{Not } P1) = 1 - FS(P1)$
- P1 or P2
 - $FS(P1 \text{ or } P2) = FS(P1) + FS(P2) - FS(P1) * FS(P2)$
- Existen múltiples maneras de derivar estadísticas para la misma expresión



Histogramas

- Key to obtaining good cost and size estimates.
- Come in several flavors:
 - Equi-depth
 - Equi-width
- Which is better?
- Compressed histograms: special treatment of frequent values.



Planes para Consultas con una sola relación

- **Objetivo:** crear un plan de ejecución para un bloque de consulta.
- **Idea Clave:** considerar cada *camino de acceso* a las tuplas de la relación. Escoger la más barata.
- Las diferentes operaciones son consideradas juntas (si un índice se usa para una selección, la proyección se realiza para cada tupla retornada, y las tuplas identificadas son pasadas en *pipelined*)

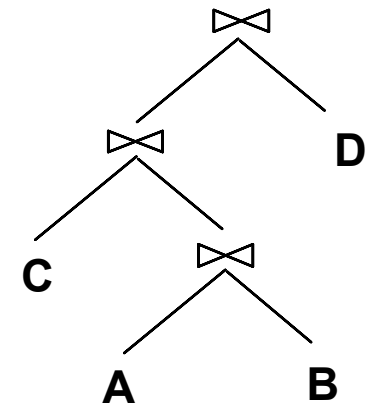
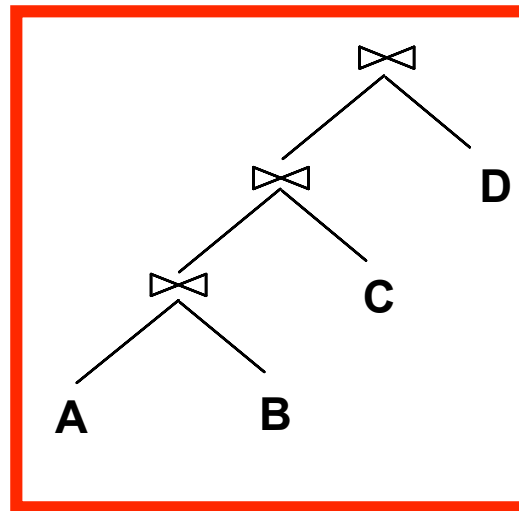
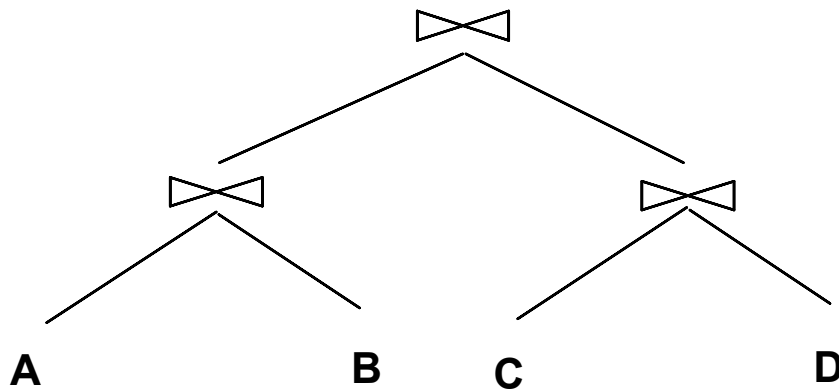


Ejemplo:

- Select P.Cod
From Producto P
Where P.categoria="chocolate"
- Si se tiene un índice en *categoria*:
 - $(1/NKeys(I)) * NTuplas(R) = (1/10) * 40000$ tuplas.
 - **Clustered Index:** $(1/NKeys(I)) * (NPages(I)+NPages(R)) = (1/10) * (50+500)$ páginas (= 55).
 - **Unclustered Index:** $(1/NKeys(I)) * (NPages(I)+NTuples(R)) = (1/10) * (50+40000)$ páginas.
- Si existe un índice en *Cod*:
 - Deberíamos retornar todas las tuplas o páginas. Con un índice **clustered**, el costo es 50+500.
- **File scan** : se retornan todas las páginas (500).

Determinado el orden de los joins.

- En principio se necesita considerar todos los posibles órdenes.





Determinado el orden de los joins

- Si el número de joins incrementa, el número de alternativas crece rápidamente. ***Es necesario restringir el espacio de búsqueda.***
- El Sistema-R: considera únicamente left-deep join trees (árboles lineales izquierdos).



Enumeración de Planes Lineales Izquierdos

- Solución Ingenua:
 - Generar todas las permutaciones de las n relaciones.
 - $n!$ combinaciones.
- **Principio de Optimalidad:** el mejor plan para el join de R_1, \dots, R_{n-1} será parte del mejor plan para el join de R_1, \dots, R_n
- Se puede reducir la complejidad a $n2^n$
- Significativamente más económico que la solución ingenua.



Sistema R

- El uso de estadísticas de instancias de la base de datos para estimar el costo de un plan de evaluación de *queries*.
- Considera solo los planes con *joins* binarios en los cuales la relación *inner* es una relación base (no una relación temporal).
- Consultas no anidadas
- No realiza eliminaciones de duplicados para las proyecciones (a menos que una cláusula DISNTINC lo requiera).
- Un modelo de costos basado en el costo de acceder los datos almacenados en memoria secundaria.



Control de Búsquedas

- Evitar Productos Cartesianos.
 - Retardar los productos cartesianos tan tarde como sea posible.
 - No considerar:
 $((R1 \times R2) \text{ Join } R3)$
Si
 $((R1 \text{ Join } R3) \text{ Join } R2)$
puede realizarse.



Programación Dinámica- Sistema R


- Enumeración usando N pasadas (si N relaciones se consideran):
 - **Paso 1:** Encontrar mejor plan de 1-relacion para cada relación. Todos los nodos posibles de recorrido de las tablas son considerados.
 - **Paso 2:** Encontrar la mejor manera de hacer el join de cada plan con 1-relacion (como un outer) a otra relación. *(Todos planes de 2-relaciones.)* Las soluciones que tienen una solución equivalente más económica, son desechadas. Se consideran que dos soluciones son equivalentes si al hacer el join producen el mismo resultado y en el mismo orden.
 - **Paso N:** Encontrar la mejor manera de hacer el join de planes de (n-1) relaciones (como un outer) con una N'th relación. *(Todos planes de N-relaciones)*
- Para subconjunto de relaciones, retener solo:
 - El plan con más bajo costo y
 - El plan con más bajo costo para cada **orden interesante** de las tuplas.


Algoritmo de Programación Dinámica


Function DynProg

Entrada Rels: Conjunto de relaciones a ser ordenadas.

Salida: un árbol lineal izquierdo para Rels.

 **PartialSolutions** := {Todos los recorridos de las tablas en Rels para todos los atributos involucrados en la consulta}

 Remover de **PartialSolutions** todo sub-plan b con una alternativa equivalente b' y con menor costo.

 For i:=2 to | **Rels** |

3.1 For all pt in **PartialSolutions**

3.1.1 For all Ri in **Rels**

Anadir pt:= JOIN a LocalPartialSolutions

pt / Ri \

end

end

3.2 Remover de **PartialSolutions** todo sub-plan b con una alternativa equivalente b' y con menor costo

3.3 Mover LocalPartialSolution a **PartialSolutions**

end

4. Seleccionar un elemento e de **PartialSolutions**

5. Agregar a e todos los productos cartesianos y demás operaciones unarias que aparecen en el consulta y que no fueron consideradas.



Control de Búsquedas

- Identificar **órdenes interesantes**:

- Aunque

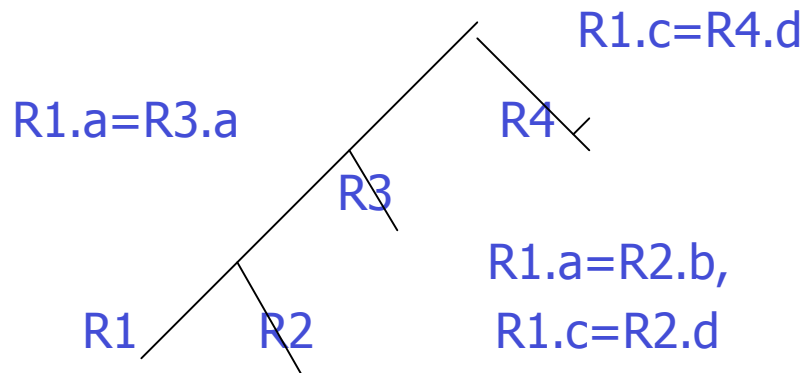
$$\text{Costo}(\text{SM}(\text{R1}, \text{R2})) > \text{Costo}(\text{NL}(\text{R1}, \text{R2}))$$

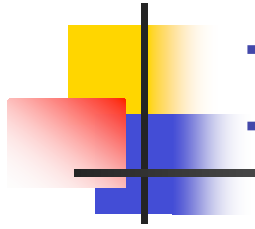
El plan:

SM(SM(R1,R2),R3) puede ser mucho más económico que otras opciones.

Manejando Órdenes Interesantes

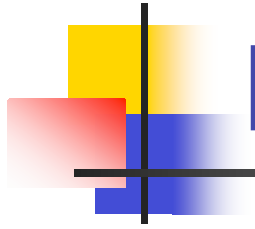
- Identificar todas las columnas que pueden aprovechar un ordenamiento.
- Identificar grupos equivalentes.
- Un plan parcial optimal para cada orden interesante.
- Ejemplo:





Ideas Claves-Sistema R

- Modelos de Costos basados en:
 - Métodos de acceso.
 - Tamaño y Cardinalidad de las relaciones.
- Aprovechamiento de la enumeración:
 - Programación Dinámica.
 - Un plan optimal por cada expresión equivalente.
 - Violación del principio de optimalidad para manejar órdenes interesantes.



Limitaciones del Sistema R

- Modelo de Costos:
 - Estimación del número de valores diferentes por cada atributos (poco preciso)
- Transformaciones:
 - Limitadas al ordenamiento de joins.
- Enumeración:
 - Limitada a bloques simples.



Enumeración de Planes

- **ORDER BY, GROUP BY, agregaciones**, etc. se manejan como un paso final, usando algún orden interesante o un operador de orden adicional.
- Un plan con N-1 relaciones no se combina con una relación adicional a menos que exista una condición de join entre ellas a menos que todas las condiciones de la cláusula WHERE se hayan considerado.
 - Es decir, **evitar en lo posible los Productos Cartesianos**.
- La solución sigue siendo **exponencial** en el número de tablas.

Ejemplo- Enumeración de Planes



Tablas:

Producto:

B+ tree en Categoria
Hash en CodProducto

Compras:

B+ tree on *CodProd*

Consulta:

```
Select P.Nombre  
From Compras C, Producto P  
Where (P.Categoria="chocolate") AND  
      (C.CodCli=100) AND  
      (C.CodP=P.CodP)
```



Ejemplo- Enumeración de Planes

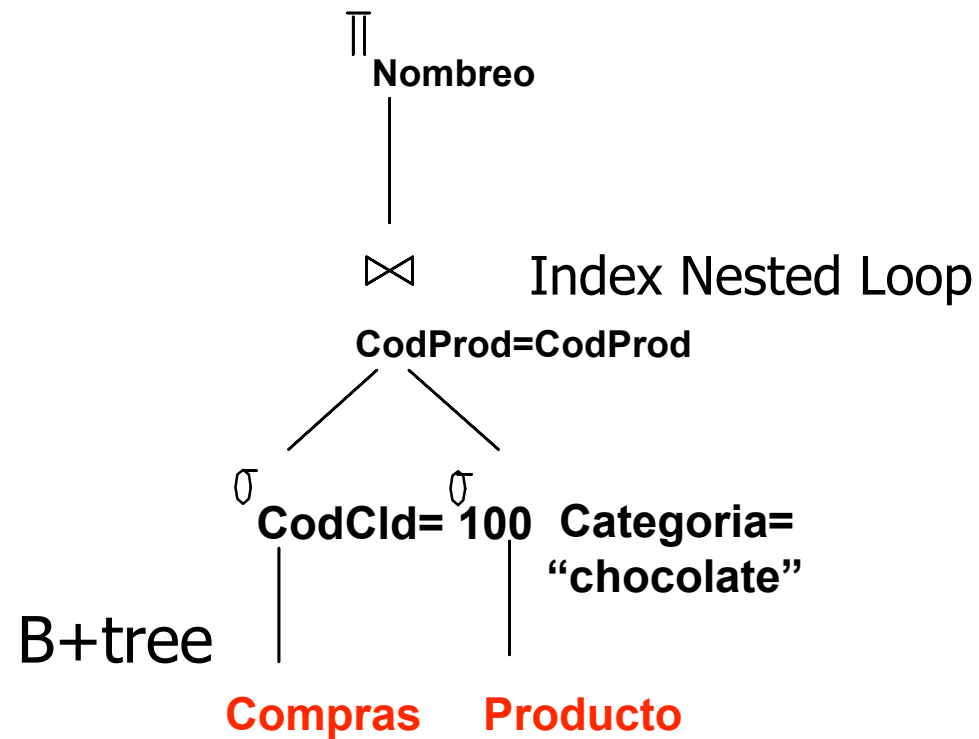
- Paso 1: (basicamente, selección de caminos de acceso)
 - *Producto*: B+ tree permite identificar las tuplas que satisfacen *categoria="chocolate"*, y es probablemente más económico. Sin embargo, si se espera que esta selección devuelva un lote de tuplas y el índice es unclustered, hacer el barrido del archivo puede ser más económico.
 - *Compras*: B+ tree en *CodCli* identifica *CodCl=100*; más económico.



Enumeracion de Planes

- **Paso 2:** Se considera cada plan identificado en el Paso 1, y se coloca como entrada externa(**outer**), y se considera como hacer join de éste solo con una relación.
 - **Por ejemplo., *Compras como externa:***
El Hash index se puede usar para identificar en la tabla de Productos las tuplas que satisfacen la condición CodProd=**outer**.

Plan de Ejecución





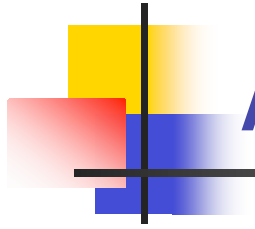
Ejemplo Enumeración Planes

- Producto(Cod(10 bytes), Nombre(40 bytes), Categoria(10 bytes), CodFab(10 bytes))
 - Índice B+ tree clustered sobre Cod.
 - Tamaño tupla: bytes.
- Fabricante(Cod(10 bytes), Nombre(40 bytes), NumEmpleados(16 bytes))
 - Índice B+ tree clustered sobre Cod
- Cliente(CodCliente(10 bytes), Nombre(40 bytes))
 - Índice B+ tree clustered sobre Cod
- Comprar(CodProf(10 bytes), CodCliente, CodFab(10 bytes), Fecha(10 bytes), Monto(32 bytes))
 - Índice B+ tree sobre Cod
- Seleccionar los clientes que han comprados todos los productos de la categoría chocolate.
- Número de Bloques: 20.
- Operadores físicos:
 - Index Nested Loop, Block Nested Loop,
- Enumerar todos los subcaminos identificados por el algoritmo de programación dinámica. Criterio de equivalencia, igual orden de los subobjetivos.



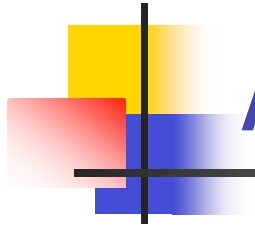
Estadísticas

- **Estadísticas:**
 - **Producto**
 - Cuatro millones de tuplas
 - Diez mil valores diferentes en el atributo **Categoría**.
 - Tuplas uniformemente distribuidas en los valores del atributo **Categoría**.
 - Cada categoría es producida por un único fabricante.
 - **Fabricante**
 - Dos mil tuplas.
 - **Cliente**
 - Mil tuplas
 - **Comprar**
 - Cuarenta millones de tuplas.
 - Distribución Uniforme de Clientes, Productos y Fabricantes



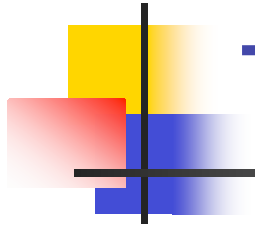
Algoritmos Aleatorios

- El espacio de soluciones es visto como un conjunto de puntos conectados por arcos que definen un conjunto de movimientos.
- Un algoritmo aleatorio realiza caminatas en el espacio de puntos.
- El tipo de soluciones consideradas depende del tipo de espacio considerado.
 - Esto también determina al tipo de movimientos o transformaciones.



Algoritmos Aleatorios

- Se realizan caminatas aleatorias en el espacio de soluciones.
- Las caminatas se realizan en N etapas.
- En cada etapa:
 - Se genera una solución inicial de manera aleatoria.
 - Se aplican M transformaciones hasta conseguir una solución optimal.
- Se selecciona el mejor optimal local.
- Las soluciones pueden tener cualquier forma.



Transformaciones Típicas

- $(A \text{ JOIN } B) \Rightarrow (B \text{ JOIN } A)$
- $(A \text{ JOIN } (B \text{ JOIN } C)) \Rightarrow ((A \text{ JOIN } B) \text{ JOIN } C)$
- $(A \text{ JOIN } B) \text{ JOIN } C \Rightarrow (A \text{ JOIN } C) \text{ JOIN } B$
- $A \text{ JOIN } (B \text{ JOIN } C) \Rightarrow B \text{ JOIN } (A \text{ JOIN } C)$

Se seleccionan aleatoriamente las sub-consultas a transformar.

Después de aplicar una transformación se calcula el costo del nuevo plan. Si el costo es mejor que el costo del mejor identificado hasta el momento, se descartan los anteriores. En caso contrario, se descarta el plan identificado.



Mejoras Iterativas

Entrada: Rel

Salida: OrdenRel

minCost:= ∞

Repeat

state:= "Random starting point" (se genera una solución inicial)

cost:= Cost(state);

Repeat

newstate:= "state after random move" (se selecciona una transformación de manera aleatoria y se aplica a la solución anterior)

newcost:= Cost(newstate)

if newcost < cost then

state:= newstate;

cost:= newcost;

end

Until "local minimum reached" (Se alcance el número de transformaciones)

if cost < mincost then

minstate:= state;

mincost:= cost

end

Until "time limit exceeded" (Se alcance el número de etapas)



Simulated-Annealing

Es una mejora al algoritmo de mejoras iterativas que permite que el mismo salga de una zona donde ha quedado atrapado por conseguir a un mínimo local que contiene un valor muy alto, con repesto a los valor optimales.

Permite movimientos hacia soluciones con valores peores, con cierta probabilidad.

Su comportamiento está determinado por los parámetros, temperatura inicial, reducción de temperatura y condición de parada.



Simulated-Annealing

Entrada: state "Random starting point"

Salida: OrdenRel

Minstate: =state;

minCost: =Cost(state);

t:="Starting temperature" (numero de transformaciones a aplicar)

Repeat

Repeat

newstate: = "state after random move" (escoger aleatoriamente una transformacion y aplicarla)

newcost: = Cost(newstate)

if newcost < cost then

state: = newstate;

cost: = newcost;

else "with probability $e^{(newcost - cost)/t}$ "

%t representa al numero de transformaciones que han sido aplicadas en la etapa

% al principio de la etapa, existe un mayor probabbilidad de escoger una peor solucion

state: =newstate;

cost: =newcost;

end

if cost < mincost then

minstate: = state;

mincost: = cost

end

until "equilibrium reached" (alcanzar el numero de transformaciones)

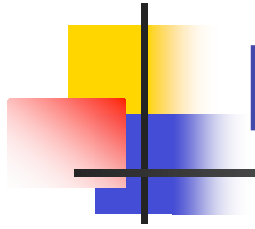
"Reduce temperature"

Until "frozen" (alcanzar el numero de etapas)



Combinación Mejoras Iterativas-Simulated Annealing

- El algoritmo de Mejoras Iterativas es muy bueno cubriendo grandes porciones del espacio de soluciones, mientras que el Simulated Annealing es muy bueno recorriendo el vecindario de un punto dado.
- Algoritmo aleatorio de dos fases:
 - Por un número aleatorio de puntos de inicio, identificar mínimos locales haciendo uso de mejoras iterativas.
 - Para el mínimo local con menor costos, utilizar el algoritmo de simulated annealing para identificar una mejor solución.



Referencias

- Selinger P., et al. "Access Path Selection in a Relational Database Management System". ACM SIGMOD 1976.
- Ioanidis Y., Kang Y. , "Randomized Algorithms for Optimizing Large Join Queries". ACM SIGMOD 1990.
- Ono K., Lohman G., "Measuring the complexity of join enumeration in query optimization". VLDB 1990.
- Steinbrum M., Moerkotte G. Kemper A. "Heuristic and Randomized optimization for the join ordering problem". VLDB Journal 1997.