

Optimización de Consultas

CDBE

Objetivo General=Minimizar el t de respuesta de una consulta

Pero también hay otras métricas y costos en DBE, objetivos mutuamente excluyentes:

Maximizar Paralelismo

Maximizar Rendimiento

Minimizar uso de Recursos (memoria,etc)

Otros objetivos

Optimización de Consultas

CDBE

Elemento central QP (query processor)=DD (data dictionary, catalogo BD)

DD=catalogo+información estadística+páginas asociadas a <> objetos del sistema

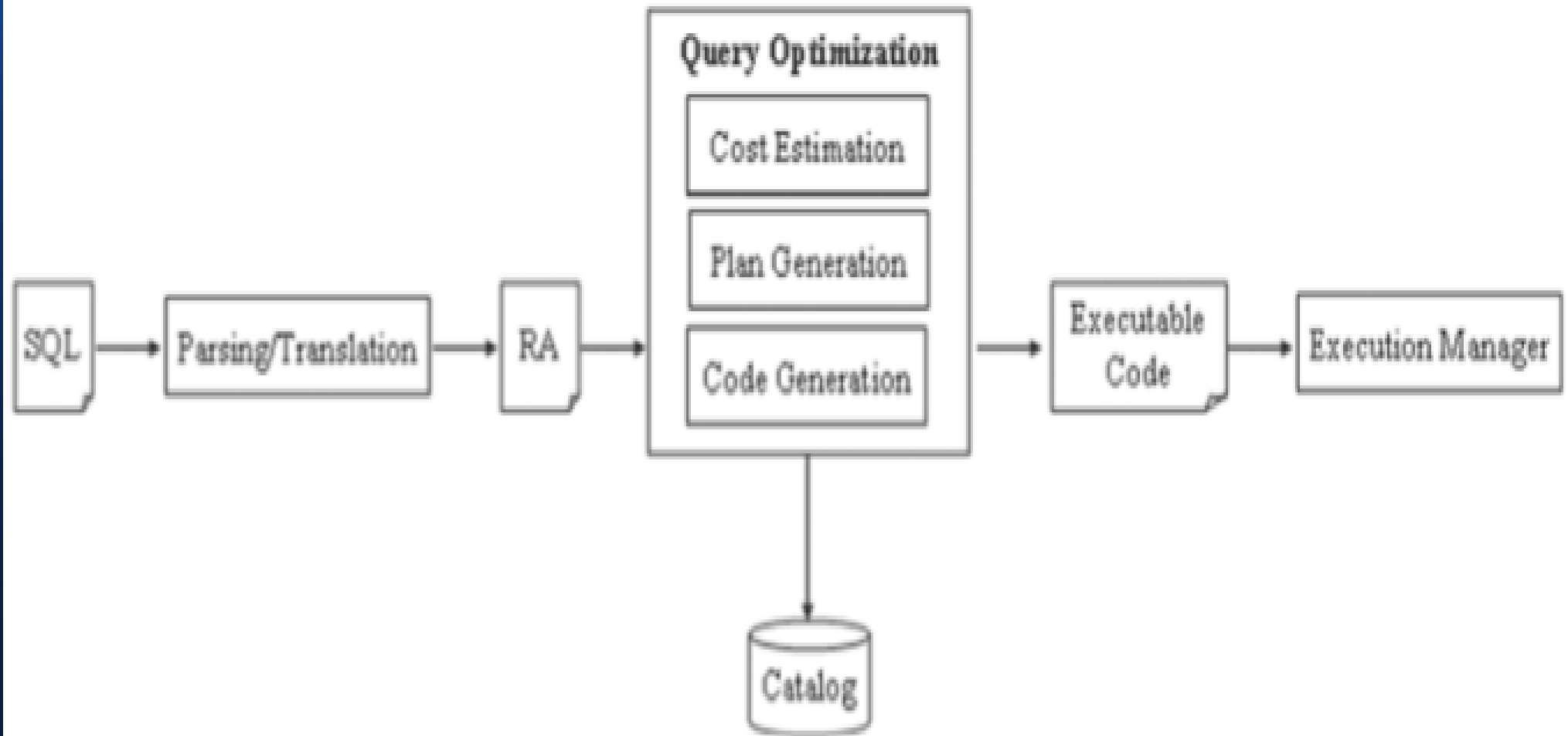
DD DDBE=DD+info de distribución y fragmentación de datos, velocidad de los enlaces de comunicación, etc.

Query parsing & translation

parsing=chequeo de sintaxis, eliminación de comentarios, etc.

translation=traducción del SQL a un lenguaje interno del sistema, por lo general, basado en algebra relacional.

Query parsing & translation



Ejemplo parsing & translation

Dada la siguiente base de datos:

CUSTOMER (CID, CNAME, STREET, CCITY);
BRANCH (BNAME, ASSETS, BCITY);
ACCOUNT (A#, CID, BNAME, BAL);
LOAN (L#, CID, BNAME, AMT);
TRANSACTION (TID, CID, A#, Date, AMOUNT);

Ejemplo parsing & translation

Select c.Cname

From Customer c, Branch b, Account a

Where c.CID = a.CID AND a.Bname = b.Bname

AND b.Bcity = 'Edina';

==>

PJ cname (SL Bcity = 'Edina' (Customer CP (Account
CP Branch)))

debe mejorarse la expresión! el producto cartesiano de 3 relaciones generará una enorme relación intermedia para resolver el query!

Ejemplo parsing & translation

Select c.Cname

From Customer c, Branch b, Account a

Where c.CID = a.CID AND a.Bname = b.Bname

AND b.Bcity = 'Edina';

==>

PJ cname (SL Bcity = 'Edina' (Customer CP (Account CP Branch)))

debe mejorarse la expresión! el producto cartesiano de 3 relaciones generará una enorme relación intermedia para resolver el query!

Ejemplo parsing & translation

PJ cname (SL Bcity = 'Edina' (Customer CP
(Account CP Branch)))

==>

PJ cname (Customer NJN Account) NJN (Account
NJNI (SL Bcity = 'Edina' (Branch)))

==>

PJ cname (Customer NJN (Account NJN (SL
Bcity='Edina' (Branch))))

Estimación de costo

query $\rightarrow$ N operadores RA $\rightarrow$ Z queries equivalentes

Obtengo Z alternativas aplicando las propiedades: asociativa, conmutativa, idempotent, distributiva, factorización sobre operadores RA

Propiedades Operadores RA

Conmutativa (operador unario)

$$\text{Uop1}(\text{Uop2}(R)) \equiv \text{Uop2}(\text{Uop1}(R))$$

Ejemplo:

SL Bname = 'Main' (SL Assets > 12000000
(Branch) $\equiv$ SL Assets > 12000000 (SL Bname =
'Main' ((Branch))

Propiedades Operadores RA

Idempotent

$$\text{Uop}((R)) \equiv \text{Uop1}(\text{Uop2}((R)))$$

Ejemplo:

SL Bname = 'Main' AND Assets > 12000000
(Branch) $\equiv$ SL Bname = 'Main' (SL Assets > 12000000 (Branch))

Propiedades Operadores RA

Conmutativa (operador binario)

excepto para la resta:

$$R \text{ Bop1 } S \equiv S \text{ Bop1 } R$$

Ejemplo:

$$\text{Customer NJN Account} \equiv \text{Account NJN Customer}$$

Propiedades Operadores RA Asociativa

$$R \text{ Bop1 } (S \text{ Bop2 } T) \equiv (R \text{ Bop1 } S) \text{ Bop2 } T$$

Ejemplo:

$$\text{Customer NJN (Account NJN Branch)} \equiv (\text{Customer NJN Account}) \text{ NJN Branch}$$

Propiedades Operadores RA Distributiva

$$\text{Uop}(R \text{ Bop } S) \equiv (\text{Uop}(R)) \text{ Bop } (\text{Uop}(S))$$

Ejemplo:

SL sal > 50000 (PJ Cname, sal (Customer)
UN PJ Ename, sal (Employee)) $\equiv$
(SL sal > 50000 (PJ Cname, sal (Customer))
UN ((SL sal > 50000 (PJ Ename, sal (Employee))))

Propiedades Operadores RA

Factorización

$$(Uop(R)) \text{ Bop } (Uop(S)) \equiv Uop(R \text{ Bop } S)$$

Ejemplo:

SL sal > 50000 (PJ Cname, sal (Customer))
UN (SL sal > 50000 (PJ Ename, sal (Employee))) $\equiv$
SL sal > 50000 (PJ Cname, sal (Customer)) UN (PJ
Ename, sal (Employee))

Queries Equivalentes Alternativas

Volviendo al query original:

```
Select c.Cname  
From Customer c, Branch b, Account a  
Where c.CID = a.CID AND a.Bname = b.Bname  
AND b.Bcity = 'Edina';
```

y aplicando las propiedades anteriores, tenemos 8 alternativas de queries equivalentes. El optimizador de query debe elegir la mas optima.

Queries Equivalentes Alternativas

De las 8 alternativas para el query:

```
Select c.Cname  
From Customer c, Branch b, Account a  
Where c.CID = a.CID AND a.Bname = b.Bname  
AND b.Bcity = 'Edina';
```

eliminamos las alternativas que proponen productos cartesianos, el espacio de solución queda reducido a 5 alternativas

Queries Equivalentes

Arbol de Alternativas

Las alternativas pueden representarse en forma de arbol, donde las hojas representan RELACIONES y los nodos representan OPERADORES RA.

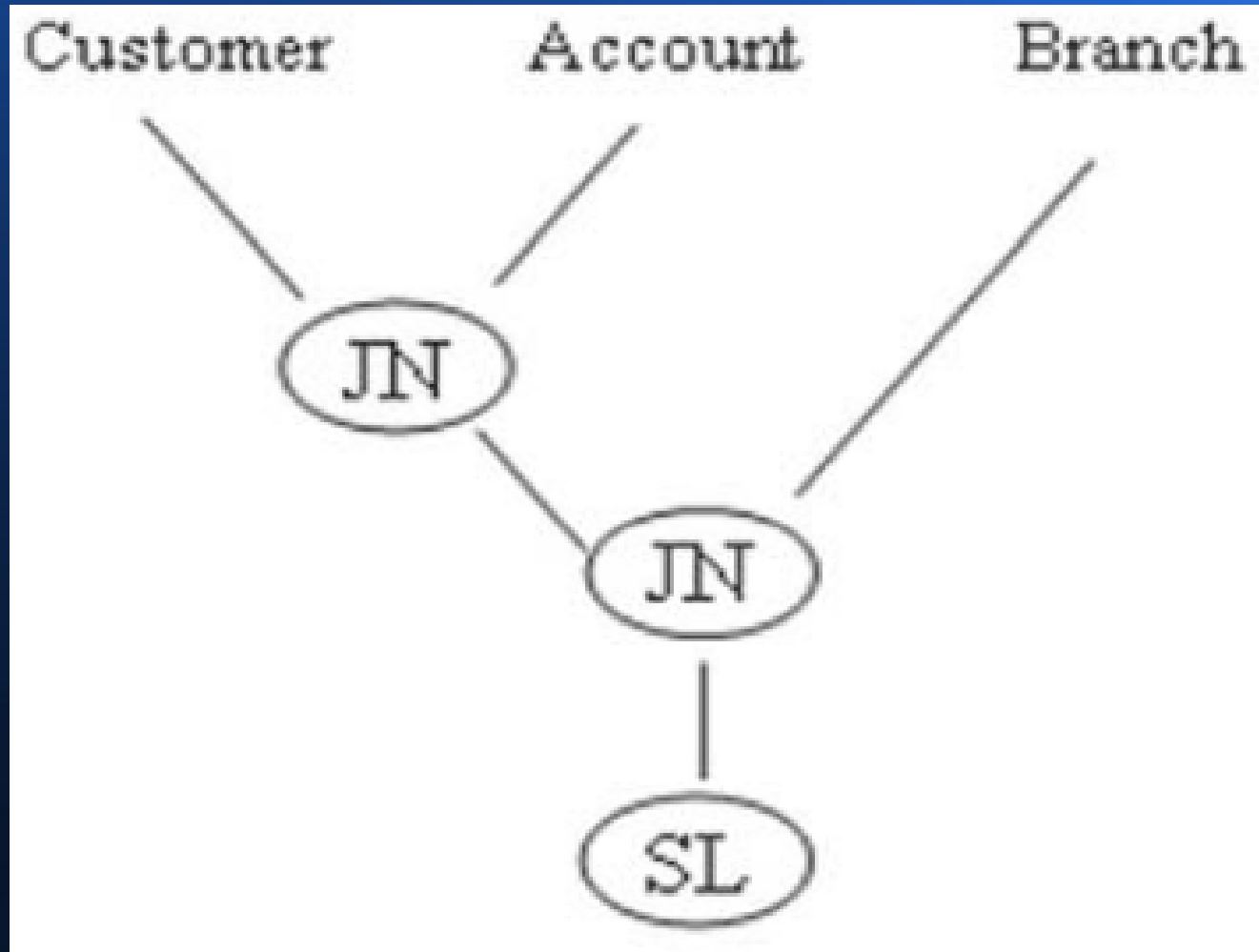
Los operadores **unarios** tienen como input una relación y devuelven una relación.

Los operadores **binarios** tienen como input dos relaciones y devuelven una relación.

Los **resultados fluyen** de un nivel a otro del arbol hasta llegar a la **raiz** del mismo.

Queries Equivalentes

Ejemplo Arbol

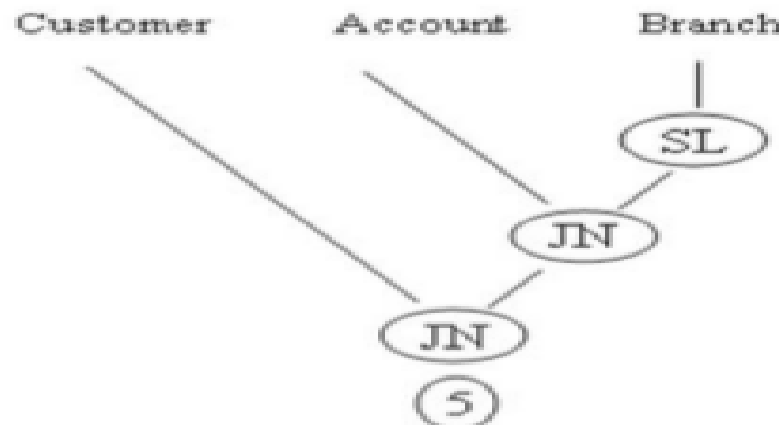
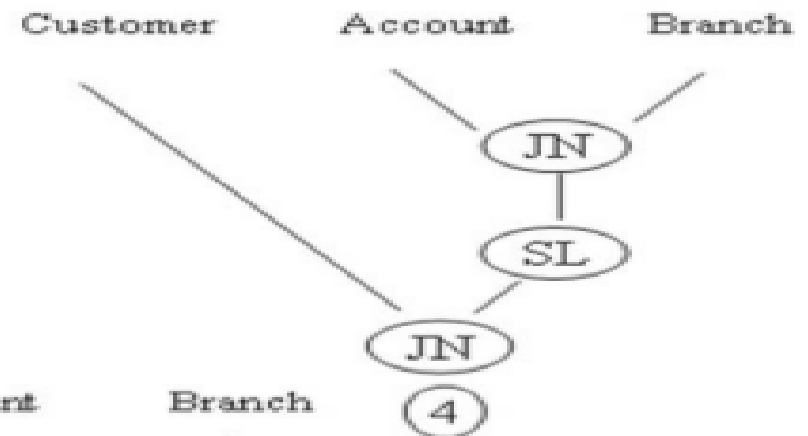
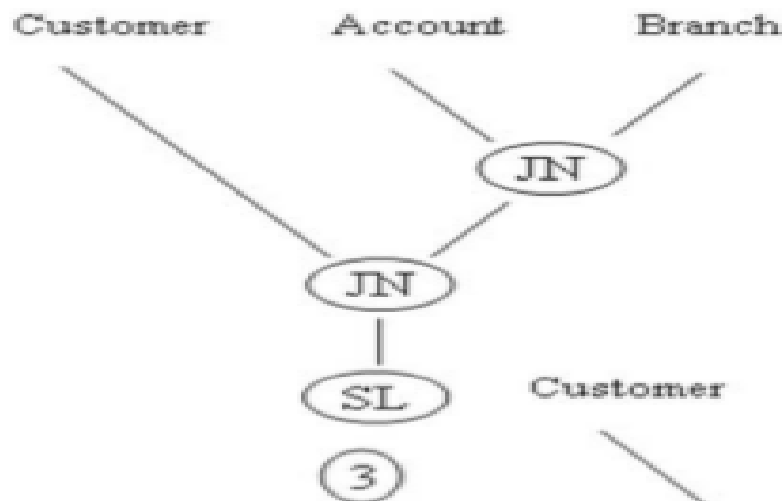
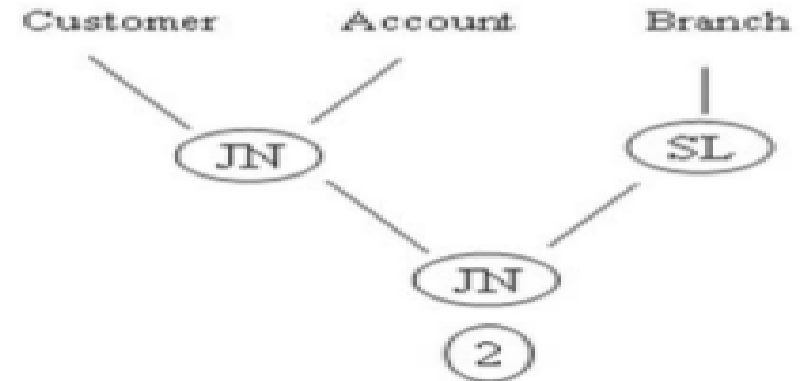
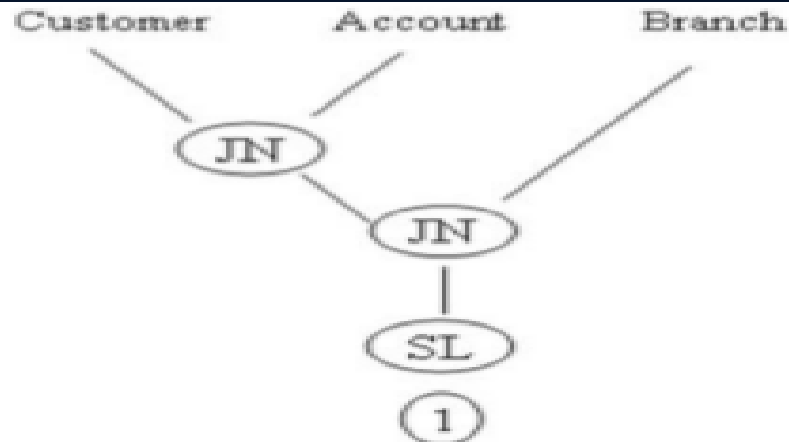


Implementación Arbol

Cada operador RA se implementa como un **iterador** que funciona como **productor** para el iterador del siguiente nivel y como **consumidor** del nivel previo.

Ejemplo: para resolver SL iterador.**open()** accede a la relación del SL, asigna los buffers de memoria requeridos para procesar la tabla; iterador.**get_next()** es llamado repetidas veces para procesar las tuplas del buffer de input y las que califican las pone en el buffer de output; cuando ya no hay mas tuplas a procesar, se ejecuta iterador.**close()** para cerrar la relación y liberar los buffers de memoria.

Arboles de las 5 alternativas de Solución



Espacio de Solución

Optimizador aplica reglas tales como:
“antes de hacer join de dos relaciones, las relaciones deben reducirse en tamaño aplicando operaciones SL,PJ”
aplicando esto, elije Alt-2 y Alt-5 y corre un análisis de costo para ambos.

Para el análisis se requiere de información estadística:

- * Hay 500 clientes (CUSTOMER)
- *En promedio cada cliente tiene 2 cuentas (ACCOUNTS)
- *Hay 100 sucursales (BRANCH) en el Banco
- *Hay 10 sucursales en la ciudad de 'Edina'
- *El 10% de los clientes tiene cuenta en las sucursales de 'Edina'

Esto permite analizar costo en función de la cantidad de tuplas que cada alternativa debe procesar.

Análisis Alt-2

Operación	Costo (acceso a tuplas)	tuplas resultantes
CUSTOMER NJN ACCOUNT → R1	500 * 1000 tuplas	1000 tuplas
SL bcity='Edina' (BRANCH) → R2	100 tuplas	10 tuplas
R1 NJN R2 → RESULTADO	1000 * 10 tuplas	100 tuplas
Costo Total	510.100 tuplas	

Análisis Alt-5

Operación	Costo (acceso a tuplas)	tuplas resultantes
SL bcity='Edina' (BRANCH) → R1	100 tuplas	10 tuplas
R1 NJN ACCOUNT → R2	1000 * 10 tuplas	100 tuplas
R2 NJN CUSTOMER → RESULTADO	500 * 100 tuplas	100 tuplas
Costo Total	60.100 tuplas	

Alt-5 es la mejor opción, tiene el menor numero de tuplas para acceder

Algunas Conclusiones

Factores que intervienen en la Optimización de queries

CDBE

numero de selecciones

numero de predicados

numero de operaciones

tamaño de cada relación

orden de las operaciones

existencia de índices

nro de alternativas para cada operación individual

DDBE

detalles de fragmentación

localización de la fragmentación

tablas / fragmentos en el sistema

velocidad de comunicación entre sitios

Tiempos de ejecución de queries

Tiempos de Ejecución de un query en DDBE

t para comunicar query local a cada DBE local +

t para correr query local +

t para ensamblar datos y generar resultado final +

t para mostrar resultado final al usuario

Optimización local (basada en
análisis de RA + estadística)

Optimización Global

Tiempos de ejecución de queries

Dada una operación

N Alternativas $\rightarrow$ N trayectorias a datos o caminos de acceso

$T = t$ de acceder cada tupla en memoria + t de traer tuplas de disco a memoria

SELECCION

JOIN

Selección S predicado (R)

Sin Indices, sin orden

Scan de todas las páginas de R en disco. Trae pagina a memoria y se examinan las tuplas en memoria vs predicado (p)

Selección S predicado (R)

B+Arbol

sobre atributo usando en predicado (p) → el uso o no del índice depende de la complejidad del predicado (p)

predicado simple → Atributo Op Value → atravesar índice desde raíz hasta 1 hoja → guarda rowid → se lee bloque que contiene la tupla → tupla objetivo

El costo depende del tipo de índice: unique o not unique, clustered no denso (las tuplas se recuperan en grupo y pueden estar contiguas en disco → 1 solo acceso I/O para recuperar bloque en disco con tuplas objetivo) o no clustered denso (las tuplas están desparramadas en disco y requiere N accesos I/O para recuperar N bloques)

predicado complejo → conjunción (AND) → uso índice si: índice por 1 atributo de predicado o ambos o todos. En el caso de la disyunción (OR) se suele utilizar scan de R

hashing en R → sobre atributo usado en predicado. Índice de un solo nivel, función hash(atributo) → posición → rowid → bloque I/O → tupla . Manejo de colisiones puede requerir de accesos adicionales.

Join

La operación mas costosa en RA

$R \text{ join } S \rightarrow$ comparar cada tupla de R con S , si condición Ok $\rightarrow$ concatenar tuplas

Distintos enfoques para resolver join $\rightarrow$ loops anidados $\rightarrow$ basados en tuplas

Ejemplo: $R \text{ join } S$ donde $R.a = S.b$

```
for each r in R do
```

```
  for each s in S do
```

```
    if  $r.a = s.b$  then add  $r || s$ 
```

```
  next
```

```
next
```

Join

M = 1000 paginas en disco para R

N = 500 paginas en disco para S

R = 100000 tuplas → cardinalidad K = 100000

S = 40000 tuplas → cardinalidad L = 40000

Usamos 3 buffers para procesar tuplas en memoria, en cada buffer entra 1 pagina:

buffer1 → procesa paginas de R

buffer2 → procesa paginas de S

buffer3 → procesa paginas resultado

Cuando el buffer3 se llena, se graba en disco

Todas las tuplas ocupan menos que una pagina

Cantidad I/O en disco = $M + K * N = 1000 + (100000 * 500) = 50.001.000$ accesos a disco → si cada acceso tarda 9 ms → 125 horas de proceso para join !!!

Se podria hacer un join anidado orientado a bloques (en vez de tuplas) y el costo sería: $1000 + (1000 * 500) = 501.000$ → 75 minutos

Si aumentamos el numero de buffers para R a 3, el costo sería $1000 + 500 = 1500$ accesos a disco → 0.225 minutos

Si usamos 202 buffers: 200 para R, 1 para S, 1 para resultado → $1000 + (1000/200) * 500 = 3500$ = menos de 1 minuto

Los DBMS actuales utilizan gran cantidad de buffers para almacenar relaciones

Bibliografía

- [1] Saeed K. Rahimi, Frank S. Haug, "Distributed Database Managment Systems: A Practical Approach", Wiley-IEEE Computer Society Press., 2010, Chapter 4 Query Optimization. Se omitió la generación del plan de ejecución y la generación de código (luego de haber realizado la optimización). Se omitieron detalles sobre RA que el alumno ya conoce de 11077 – BD I.