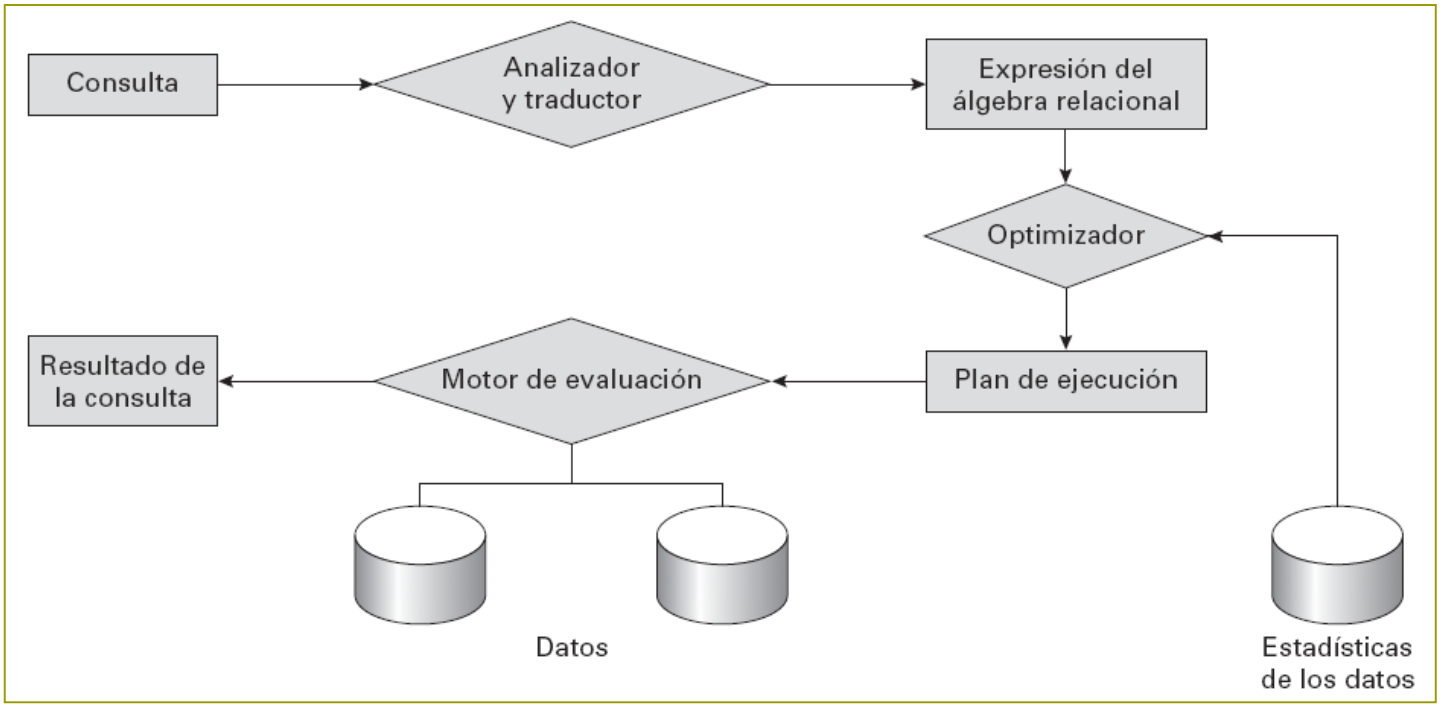
Procesamiento de Consultas

Base de Datos II

Introducción

- El procesamiento de consultas hace referencia a la serie de actividades que se realizan para recuperar datos desde una base de datos.
- Los pasos involucrados en el Procesamiento de las consultas son:
  - Análisis y Traducción
  - Optimización.
  - Evaluación.

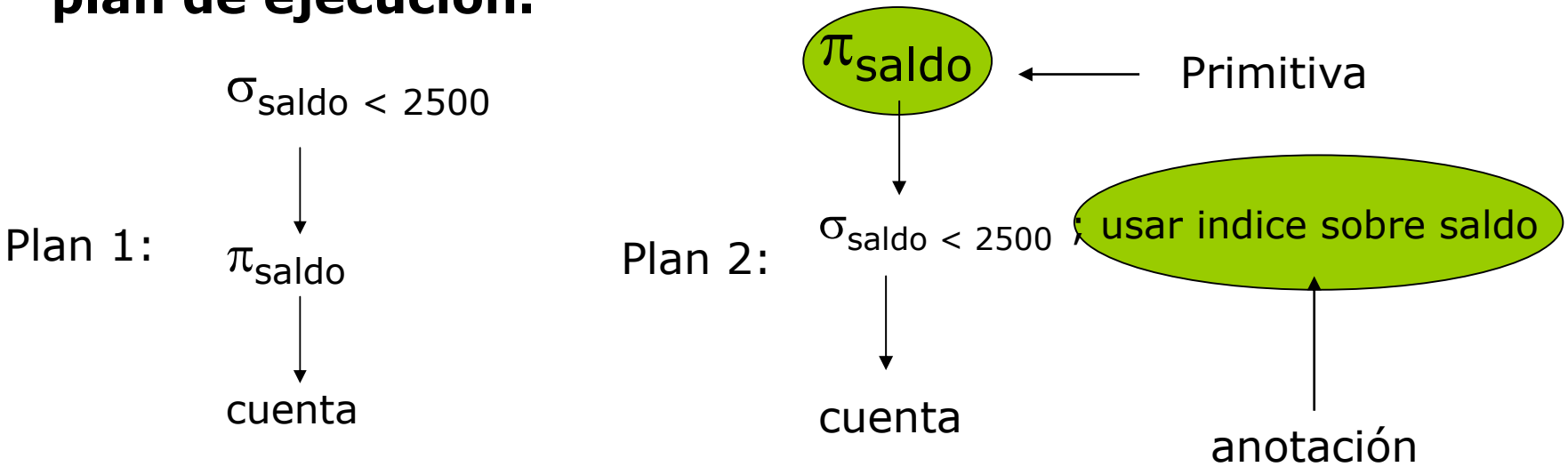


Introducción

- El Análisis y Traducción de consultas cumple la misma función que el analizador léxico y sintáctico de un compilador:
  - Revisa la sintaxis.
  - Verifica en el catálogo del sistema que todos los identificadores sean nombres de objetos de la base de datos.
- El resultado es el conjunto o serie de expresiones del álgebra relacional a evaluar para obtener el resultado de la consulta.
- Ejemplo:
    - SELECT saldo FROM cuenta WHERE saldo < 2500;
  - Se traduce en:
    - $\sigma_{saldo < 2500} (\pi_{saldo}(cuenta))$
  - O en:
    - $\pi_{saldo}(\sigma_{saldo < 2500} (cuenta))$

Introducción

- La fase de Optimización devuelve como resultado el plan de ejecución de la consulta
- Por lo general las operaciones del álgebra están anotadas con instrucciones de cómo evaluar, que pueden ser:
  - Algoritmo a usar
  - Índice a usar
- Las operaciones anotadas se conocen como **primitivas de evaluación**
- El conjunto de todas la primitivas a evaluar se conoce como **plan de ejecución.**



Coste de una consulta

- El coste de un consulta esta dado por la suma del coste de evaluación de las primitivas de los planes de ejecución
- Por lo general el optimizador devuelve el plan de menor coste.
- La métricas de coste pueden ser:
  - Costo de Acceso a Disco
    - Número de bloques leídos
    - Número de bloques escritos
    - Número de búsquedas realizadas
  - Utilización de CPU (Tiempo de uso del cpu)
  - Uso de Memoria Principal (Tamaño de RAM utilizada)
  - Uso de Memoria Auxiliar (Archivos auxiliares en disco)
  - Costo de comunicación (Tiempo de transmisión de datos a través de una red)
- El optimizador también hace uso de informaciones estadísticas recabas acerca de las relaciones

Coste de una consulta

- Como ejemplo supóngase que se utiliza como métrica el numero de bloque leídos/escritos
- El optimizador puede tener los siguientes datos estadísticos acerca de la tabla cuenta.
  - Número de Bloques en disco: 100.
  - Número de valores diferentes en columna “saldo”: 20
  - Número de Filas: 2000.
  - Índice secundario sobre la columna saldo.
- De los datos anteriores puede inferirse otros datos:
  - Número promedio de filas por bloques: 20
  - Número promedio de filas por valor de la columna saldo: 100.

Coste de una consulta

- El optimizador puede utilizar los datos anteriores para aproximar el coste de cada plan
- Plan 1:
  - Proyección sobre columna saldo
    - Implica leer 100 bloques de disco
  - Selección de los valores de saldo menor a 2500
    - Implica tiempo de CPU aproximado al tiempo de leer 1 bloque de disco
  - Coste del Plan 1 igual a leer 101 Bloques de disco

Coste de una consulta

- Plan 2:
  - Selección de los valores de saldo menor a 2500
    - Suponiendo que 2500 fuera el valor medio de los valores en la columna saldo significa que hay 10 valores de saldo menores a 2500.
    - En promedio hay 1000 filas con saldo menor a 2500, teniendo en cuenta que hay 100 filas en promedio por cada valor de la columna saldo.
    - En promedio hay 50 bloques con filas cuyo saldo es menor que 2500, ya que en promedio hay 20 filas por bloque
    - Con el índice se puede planificar para que solo se realicen la lectura de estos 50 bloques.
    - La selección entonces toma como coste promedio la lectura de 50 Bloques.
  - Realizar la proyección toma un tiempo de CPU igual a leer 1 bloque.
  - Coste del Plan 2 igual a 51 Bloques en promedio
- Este análisis da como resultado un tiempo promedio el cual no puede ser el caso de una consulta real.

Coste de una consulta

- El optimizador devuelve el plan 2 como Plan de ejecución de la consulta, ya que:
  - El plan 1 siempre tiene siempre un costo de 101 bloques
  - El plan 2 tiene en promedio un costo de 51 bloques.
  - El coste real del plan 2 sería siempre mayor a 51 bloques, pero gracias al índice muy difícilmente tomaría un coste de 101 bloques.
- La optimización de consultas es similar a las técnicas de optimización tradicionales:
  - Buscar una solución dentro de un espacio de soluciones, tal que dicha solución optimice (min/max) el resultado de una función que representa al problema.
- Sin embargo el optimizador no siempre escoge el plan más optimo, ya que una búsqueda exhaustiva de la estrategia óptima puede consumir mucho tiempo de proceso.
  - Se dice entonces que el optimizador escoge una estrategia "razonablemente eficiente".

Información Estadística

- El SGBD guarda información estadística de las relaciones en el catalogo, la cual es utilizada por el optimizador para evaluar el coste de los planes
- La informaciones que se deben guardar sobre las relaciones son:
  - Nro. de tuplas de la relación *r*.
    - n_r
  - Nro. de bloques que contienen tuplas de la relación *r.*
    - b_r
  - Tamaño en bytes de una tupla de *r*.
    - t_r
  - Nro. de tuplas de *r* que caben en un bloque.
    - f_r
  - Nro. de valores distintos del atributo A de la relación r.
    - V(A,*r*)
  - Nro. medio de filas de *r* que satisfacen una condición de igualdad sobre el atributo A de la relación *r*.
    - CS(A,*r*) = n_r / V(A,r)

Información Estadística

- Además de la información de las relaciones, también se utilizan información acerca de los índices:
  - Grado de salida de los nodos internos del índice *i* (para índices con estructuras de Árbol B/B+).
    - g_i
  - Nro. de bloques en el nivel hoja del índice *i*.
    - MB_i
  - Altura del índice para el atributo A de *r*.
    - $AA_i = \lceil \log_{g_i/2}(V(A,r)) \rceil$

Operación Selección

A1. Búsqueda lineal (Full Scan): Se examina cada bloque del archivo y se comprueba si los registros cumplen con la condición de selección. El costo de este algoritmo es: $C_{A1} = b_r$ ya que se deben leer todos los bloques del archivo

```
SQL> SELECT Apellido,
  2         dept
  3       FROM emp
  4       WHERE dept = 42;
```

$n_r = 25$
$b_r = 9$
$t_r = 60$
$f_r = 3$
$V(A,r) = 12$
$CS(A,r) = n_r/V(A,r) = 2,5$

apellido	dept	
Velasquez	50	
Ngao	41	
Nagayama	31	
Quick-To-See	10	
Ropeburn	50	
Urguhart	41	
Menchu	42	←
Biri	43	
Catchpole	44	
Havel	45	
Magee	31	
Giljum	32	
Sedeghi	33	
Nguyen	34	
Dumas	35	
Smith	41	
Nozaki	42	←
Patel	42	←

Operación Selección

A2. Búsqueda binaria: Si la tabla esta ordenada físicamente por el atributo A y la condición es una igualdad sobre el atributo, entonces se puede utilizar búsqueda binaria. El costo estimado será: $C_{A2} = \lceil \log_2(b_r) \rceil + \lceil CS(A,r)/f_r \rceil - 1$

Costo de localizar el 1er. Bloque

Nro. de bloques subsecuentes ya que la clave es no candidata y ordena la tabla

```
SQL> SELECT id,
  2         apellido
  3    FROM emp
  4    WHERE id = 10;
```

$n_r = 25$
$b_r = 9$
$t_r = 60$
$f_r = 3$
$V(A,r) = 12$
$CS(A,r) = n_r/V(A,r) = 2,5$

ID	Apellido
1	Velasquez
2	Ngao
3	Nagayama
4	Quick-To-See
5	Ropeburn
6	Urguhart
7	Menchu
8	Biri
9	Catchpole
10	Havel ←
11	Magee
12	Giljum
13	Sedeghi

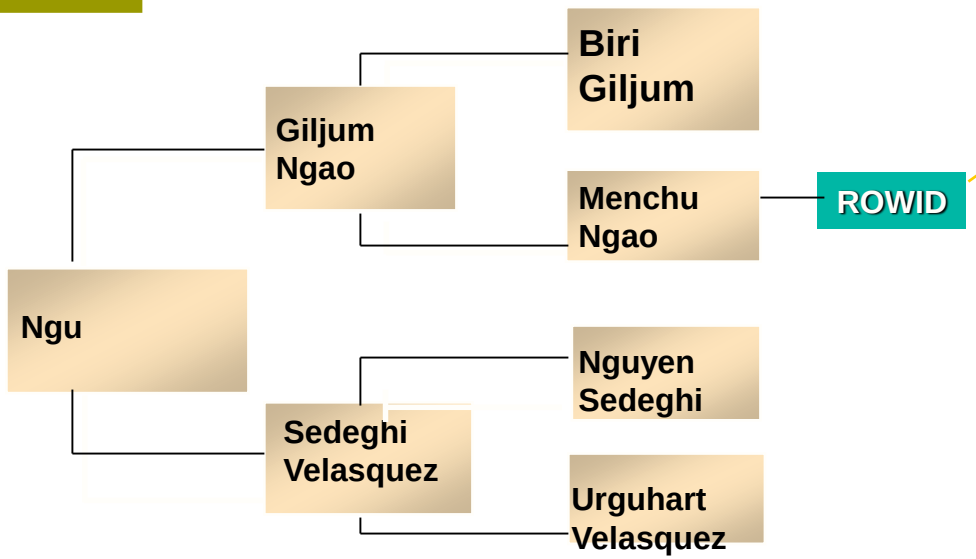
Operación Selección

- **A3. Índice primario, igualdad en clave candidata:** Ya que el índice está construido sobre el atributo clave, la búsqueda puede utilizar este índice para rescatar los datos. Por lo tanto para recuperar el único registro que cumple con la condición se necesita leer sólo un bloque.

El costo estimado será: $C_{A3} = AA_i + 1$ ← ya que a clave es candidata

$AA_i = 3$

$MB_i = 4$



ID	LAST_NAME
1	Velasquez
2	Ngao
3	Nagayama
4	Quick-To-See
5	Ropeburn
6	Urguhart
7	Menchu
8	Biri
9	Catchpole
10	Havel
11	Magee
12	Giljum
13	Sedeghi
14	Nguyen
15	Dumas
16	Maduro
17	Smith
18	Nozaki
19	Patel
20	Newman
21	Markarian
22	Chang
23	Patel
24	Dancs
25	Schwartz

Operación Selección

- **A4. Índice primario, igualdad basada en un atributo no clave:** El costo está determinado por la cantidad de bloques que contienen registros que cumplen con la condición de igualdad más la altura del índice.

El costo estimado será: $C_{A4} = AA_i + [CS(A,r) / f_r]$

```
SQL> SELECT det_asi_nro,
  2         det_item
  3    FROM detalle
  4    WHERE det_asi_nro = 2;
```

$n_r = 13$
$b_r = 7$
$t_r = 1024$
$f_r = 2$
$V(A,r) = 3$
$CS(A,r) = n_r/V(A,r) = 4$
$AA_i = 2$
$MB_i = 2$

Asiento	Item
1	1
1	2
1	3
1	4
2	1 ←
2	2
2	3
2	4
3	3
3	2
3	3
3	4
3	5

Operación Selección

- **A5. Índice secundario, igualdad:** En el mejor de los casos el campo indexado es una clave candidata, entonces de la exploración del índice resulta un puntero al registro deseado, derivando en un coste igual a la aplicación del Algoritmo **A3.**

  Si el campo indexado no es una clave, de la exploración del índice se debe acceder a un cajón con CS(A,*r*) punteros, como el índice es secundario se estudia el peor caso, en el cual cada uno de los registros se encuentra en un bloque diferente, por lo tanto el costo es:

$$\mathbf{C_{A5} = AA_i + 1 + CS(A,}\mathit{r}\mathbf{)}$$

Operación Selección

- **A6. Índice primario, comparación:** Donde A <= *v* ó A >= *v*. Dado que el índice es primario se sabe que la tabla está ordenada por el valor de la clave, por lo tanto, si el operador es > ó >= se realiza una búsqueda por el índice primario para encontrar a v en A y luego se realiza un recorrido lineal desde esa tupla hasta el final de la tabla. Para la desigualdad < ó <= no es necesario utilizar el índice, simplemente se inicia el recorrido que terminará cuando A = v. Sea **c** el número estimado de registros que satisfacen la condición.

  El costo estimado será: $\mathbf{C_{A6} = AA_i + [c / f_r]}$

- **A7. Índice secundario, comparación:** Se consideran las condiciones de comparación <, <=, > o >=. Si se asume que al menos la mitad de los registros satisfacen la condición , entonces hay que acceder a la mitad de los bloques del índice del nivel más bajo, y mediante el índice, acceder a la mitad de los bloques del archivo. Además hay que recorrer un camino en el índice desde la raíz hasta el bloque de la primera hoja que se va a usar.

  El costo medio estimado es el siguiente: $\mathbf{C_{A7} = AA_i + [MB / 2] + [b_r / 2]}$

  En el peor de los casos debe tener que acceder a todos los bloques del arvchivo.

Operaciones de Selección Complejas

- **Conjunción:** Una selección conjuntiva es de la forma $\sigma_{\theta_1 \wedge \theta_2 \wedge \ldots \theta_n}(r)$

```
SQL> SELECT last_name
  2       FROM s_emp
  3       WHERE dept_id = 50
  4       AND salary > 1000;
```

- Suponiendo que las condiciones θ_i son independientes unas de otras, se puede estimar el tamaño de cada condición como:

$$t_i = \sigma_{\theta_i}(r)$$

- Así, La probabilidad de que una tupla satisfaga la condición de selección es t_i / n_r se llama ***selectividad*** de la selección.
- La probabilidad de que una tupla cumpla con todas las condiciones de la conjunción es el producto de todas las selectividades, que deriva en:

$$n_r * \frac{t_1 * t_2 * \ldots t_n}{n_r^n}$$

Operaciones de Selección Complejas

- **Disyunción**: Una selección disyuntiva es de la forma $\sigma_{\theta_1 \vee \theta_2 \vee \ldots \theta_n}(r)$

```
SQL> SELECT last_name
  2       FROM s_emp
  3       WHERE title = 'President'
  4       OR title = 'Stock Clerck';
```

- Una condición disyuntiva se cumple mediante la unión de todos los registros que cumplen las condiciones individuales θ_i.
- Como $\mathbf{t_i/n_r}$ es la probabilidad de que una tupla cumpla θ_i (osea la selectividad), entonces la probabilidad de que una tupla cumpla la disyunción es 1 menos la probabilidad que no se cumpla ninguna de las condiciones.

$$1 - \left[1 - \frac{t_1}{n_r}\right] * \left[1 - \frac{t_2}{n_r}\right] * \ldots * \left[1 - \frac{t_n}{n_r}\right]$$

Operaciones de Selección Complejas

- **Negación**: como el resultado de una negación son simplemente las tuplas de r que no están en $\sigma_\theta(r)$, entonces el número de tuplas que cumplen con la condición de negación es:

$$tam(r) - tam(\sigma_\theta(r)) =$$
$$tam(r) * (1 - tam(\sigma_\theta(r))/ tam(\sigma(r))) =$$
$$tam(r) * (1 - S_\theta)$$

```
SQL> SELECT last_name
  2       FROM s_emp
  3       WHERE NOT (title = 'Stock Clerk');
```

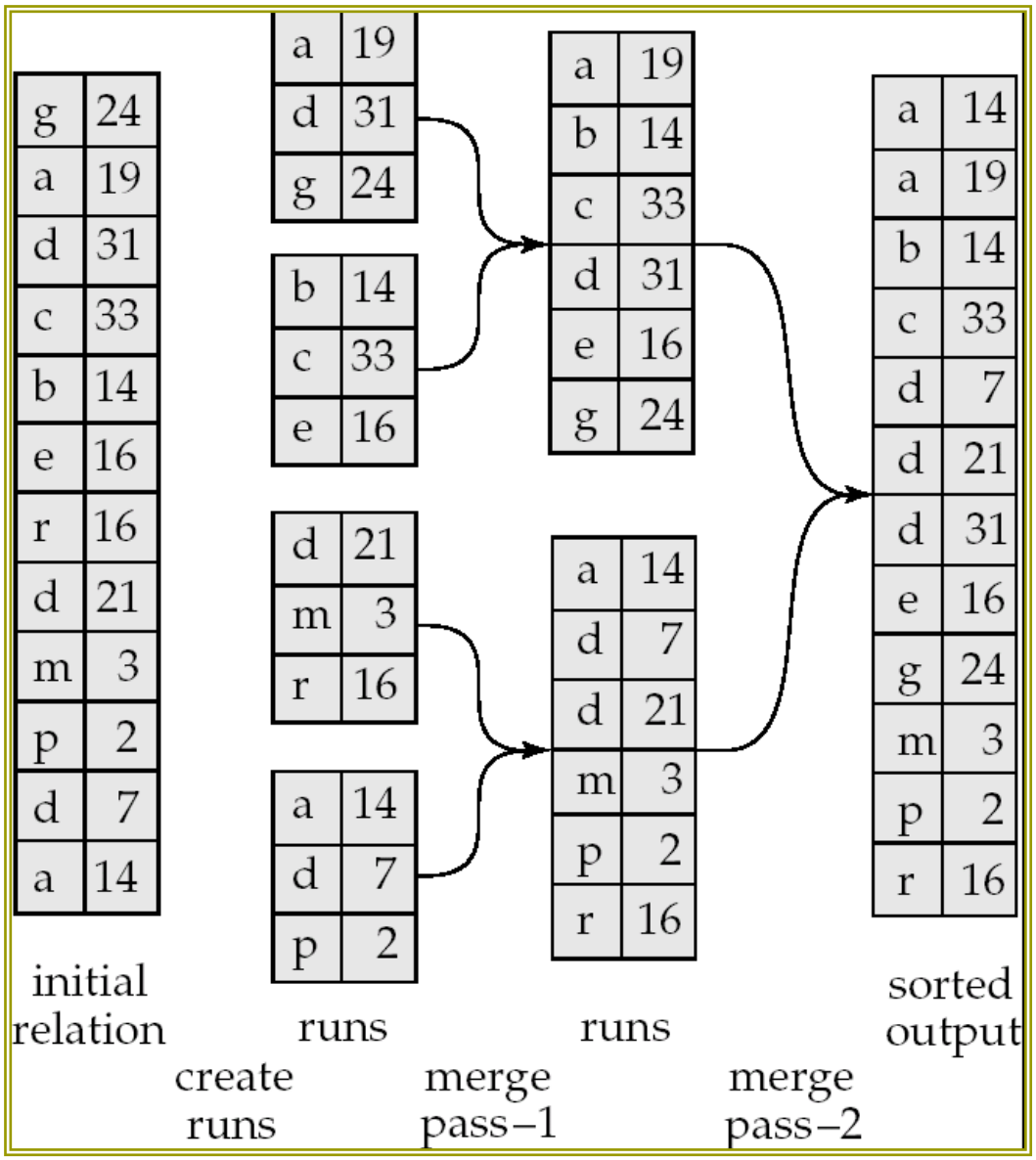
Algoritmos para la conjunción y disyunción

- **A8. Selección conjuntiva utilizando un índice**
  - Solo hay índices en algunas de las condiciones. Se elije a uno de ellos.
  - Se usas cualquier algoritmo del A2 al A7 puede recuperar los registros según el índice elegido.
  - En memoria intermedia se verifica que los registros cumplan el resto de las condiciones.
  - Se puede utilizar la información de selectividad (si se cuenta con la misma) para determinar que índice aplicar. El índice para la condición menos selectiva recupera menor cantidad de registros.
- **A9. Selección conjuntiva utilizando un índice compuesto:** Si la selección especifica una condición de igualdad en dos o más atributos y existe un índice compuesto en estos campos, se podría buscar en el índice directamente. El tipo de índice determina cuál de los algoritmos A3, A4 o A5 se utilizará.
- **A10. Selección conjuntiva, intersección de identificadores:** este algoritmo se utiliza cuando existen índices para cada condición individual. Se examina cada índice en busca de punteros. La intersección de todos los punteros recuperados forma el conjunto de punteros a las tuplas a leer.
- **A11. Selección disyuntiva mediante la unión de identificadores:** al igual que en el caso anterior se utilizan caminos de acceso en cada una de las condiciones, la unión de punteros forma el conjunto de punteros a tuplas que satisfacen la condición disyuntiva. Sin embargo se deberá efectuar siempre una búsqueda lineal si una de las condiciones no tiene caminos de acceso.

Ordenación

- **Ordenación es importante en BD:**
  - Se solicitan consultas ordenadas
  - Hace que algunas operaciones (como el join) sean más eficientes
- **La ordenación se puede lograr de dos maneras:**
  - Construyendo un índice por la clave de ordenación y utilizando luego ese índice. Este proceso ordena la relación sólo *lógicamente*, no en forma física, lo que podría significar acceder al disco por cada tupla.
  - Ordenando los datos en memoria principal.
- El problema es cuando el conjunto de datos no cabe en la memoria principal, por lo cual se debe realizar una **ordenación externa:**
  - Primero se crean varias secuencias ordenadas de tamaño igual al tamaño de la memoria principal (**M**), grabando el resultado en un archivo temporal.
  - Las secuencias ordenadas se usan como entrada de un proceso de mezcla (igual a la fase de mezcla del **Merge Sort**), en la cual se mezclan hasta **M-1** secuencias ordenadas por vez.
  - Durante la mezcla las secuencias ordenadas se leen por pedazos de tamaño igual a un bloque de disco (o página de memoria)
  - El proceso termina cuando se obtiene una única secuencia ordenada.
  - El coste de una ordenación externa esta dado por $\mathbf{b_r * (2 * \lceil \log_{M-1} (b_r / M) \rceil + 1)}$

Ordenación


g 24
a 19
d 31
c 33
b 14
e 16
r 16
d 21
m 3
p 2
d 7
a 14
a 19
d 31
g 24
b 14
c 33
e 16
d 21
m 3
r 16
a 14
d 7
p 2
a 19
b 14
c 33
d 31
e 16
g 24
a 14
d 7
d 21
m 3
p 2
r 16
a 14
a 19
b 14
c 33
d 7
d 21
d 31
e 16
g 24
m 3
p 2
r 16
initial relation
runs
runs
sorted output
create runs
merge pass–1
merge pass–2

Operación Join – Estimación del tamaño

- El producto cartesiano r x s contiene $n_r * n_s$ tuplas, y el tamaño de cada tupla es $t_r + t_s$, por lo que se puede calcular su tamaño.
- Sean r y s dos relaciones y R y S los atributos seleccionados de r y s respectivamente, si:
  - Si $R \cap S = \emptyset$, entonces $r \otimes s$ (r join s) es lo mismo que r x s.
  - Si $R \cap S$ es una clave externa de r, entonces el número de tuplas en $r \otimes s$ es exactamente el número de tuplas de s y viceversa.
  - Si $R \cap S$ no es una clave de R ni de S y $R \cap S = \{A\}$ entonces se seleccionan de s todas las tuplas que cumplen con A:

$$CS(A , s) = n_s / V(A, s)$$

Que se unen a todas las tuplas en r que cumplen con A, entonces se estima que hay tantas tuplas como:

$$(n_r * n_s) / V(A, s)$$

Algoritmos de Join

- **Join en bucles anidados**
  - Si z = r ⊗ s, *r* recibe el nombre de relación externa, y *s* se llama relación interna. El algoritmo de bucles anidados se puede representar como:

```
para cada tupla t_r en r
    para cada tupla en t_s en s
        si ( t_r, t_s ) satisface la condición de join entonces añadir
        ( t_r. t_s )al resultado
```

  - Para cada registro de r se tiene que realizar una exploración completa de s, entonces el número de bloques a acceder dependerá de la memoria
  - Si sólo puede contener un bloque de cada relación, la cantidad de accesos a bloques está dado por: $n_r * b_s + b_r$.
  - Si las dos relaciones caben en memoria, la cantidad de accesos a bloques está dado por: $b_s + b_r$.

Algoritmos de Join

- **Join en bucles anidados por bloque**
  - Si z = r ⊗ s, *r* recibe el nombre de relación externa, y *s* se llama relación interna. El algoritmo de bucles anidados por bloques se puede representar como:

```
Para cada bloque b_r de r
    Para cada bloque b_s de s
        para cada tupla t_r en en b_r
            para cada tupla en t_s en b_s
                si (t_r,t_s) satisface la condición de  join entonces
                añadir (t_r.t_s) al resultado
```

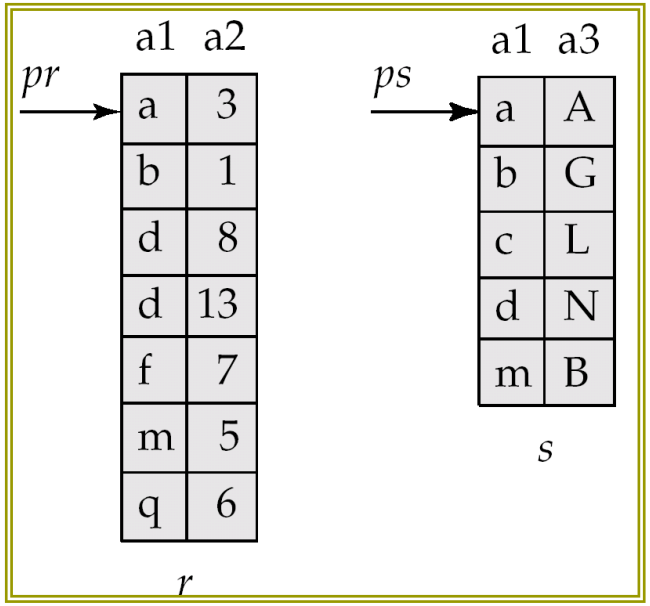
  - Si sólo puede contener un bloque de cada relación, la cantidad de accesos a bloques está dado por: $b_r * b_s + b_r$.
  - Si las dos relaciones caben en memoria, la cantidad de accesos a bloques está dado por: $b_s + b_r$.

Algoritmos de Join

- **Join en bucles anidados por índices**
  - Este algoritmo se puede usar si existe un índice en el atributo de join de la relación interna.
  - El costo del algoritmo se puede calcular como sigue:
    - Para cada tupla de la relación externa *r* se realiza una búsqueda en el índice de s.
    - Sea c = costo de la búsqueda en el índice, el cual se puede calcular con cualquiera de los algoritmos de selección con índices.
    - El costo del join es: $b_r + n_r * c$

Algoritmos de Join

- **Join por mezcla**
  - Las relaciones deben estar o ser ordenadas por las claves de join.
  - Como las relaciones están ordenadas, las tuplas son consecutivas y sólo se hace un ciclo en ambas relaciones, este método resulta eficiente.
  - El número de acceso a bloques de disco es: $b_r + b_s$.
  - Si las relaciones fueron primeramente ordenadas se añade el costo de cada ordenación.



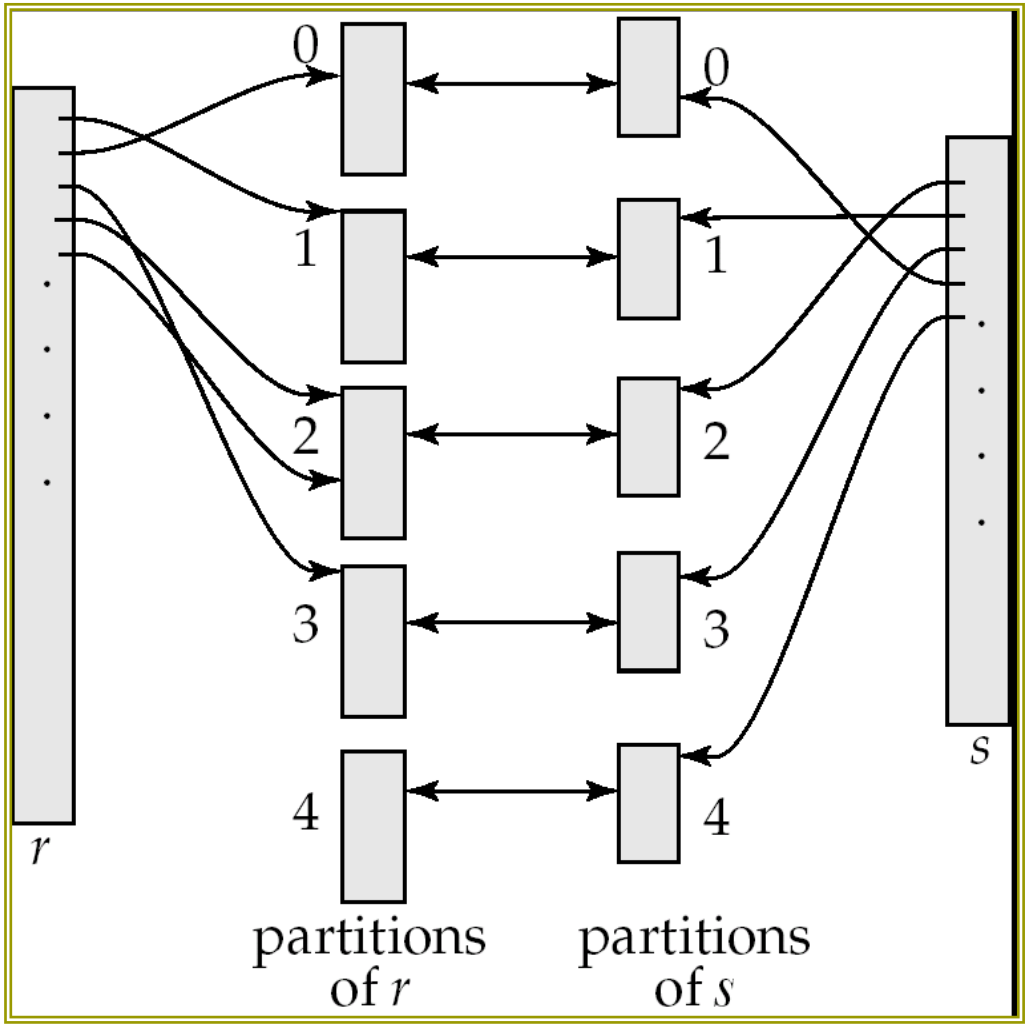
- **Join por mezcla híbrida.**
  - Aprovecha el hecho de una de las relaciones esté ordenada y sobre la otra existe un índice **i** de tipo B+ secundario.
  - Se realiza la reunión de los registros de una relación con los registro índices y luego se resuelven los punteros.
  - Coste = br + MBi + bs

Algoritmos de Join

1. **Reunión por Asociación (Hash)**
   1. Se utiliza un función hash ***H*** para crear ***N*** particiones de cada relación.
      1. Toma como entrada los valores de los atributos de join de cada fila
      2. Retorna como resultado el identificador h_i del cajón donde debe ir la fila
      3. La tabla de menor tamaño es considerada la tabla de **construcción** o interna, la tabla de mayor tamaño es llamada externa o de **prueba**
   2. Para realizar el join se procesan a la vez las particiones de cada relación que tienen el mismo identificador.
      1. Para la partición de la relación interna se construye un índice asociativo según el valor de los atributos de join.
      2. Por cada fila de la relación externa se utiliza los valores de sus atributos de join para, a través del índice asociativo, determinar (**probar**) las filas de la relación interna con la cual debe reunirse (**construirse**).

Algoritmos de Join

- **Reunión por Asociación (Hash)**



Algoritmos de Join

- **Reunión por Asociación**
  - El valor de ***N*** y la función hash ***H*** se deben escoger de manera que cada partición de la relación interna junto con su índice quepan en memoria.
    - Típicamente N es escogida como $\lceil b_s/M \rceil * 1.2$
    - Las particiones de la relación externa no necesitan caber en memoria
  - **Particionamiento Recursivo o División Recursiva**
    - Se aplica cuando **N** > **M**
    - Primero se parten ambas relaciones en M-1 particiones de acuerdo a alguna función H_0
    - Luego se vuelven partir las particiones usando una nueva fúncion H_1
    - Se aplica la división recursiva hasta que las particiones de la relación interna quepan en memoria.
    - Raramente utilizada. Ej. Una relación de 1GB con bloques de 4KB necesitaría de la partición recursiva sólo si se tiene menos de 2MB de memoria

Algoritmos de Join

- **Reunión por Asociación**
  - Un particionamiento se dice que es **sesgado** si alguna de sus particiones tienen más filas fila que otras.
  - El **Desbordamiento** de las particiones de la relación interna puede ser **resuelto** volviendo a dividir dicha partición. También se vuelve a dividir la partición de la relación externa correspondiente.
  - El **Desbordamiento** puede ser **evitado**, si se consideran muchas particiones inicialmente.
  - Un **desbordamiento** no podrá ser **resuelto** ni **evitado** cuando existen muchos valores duplicados en los atributos de reunión.
    - En estos casos se puede utilizar la estrategia de reunión por bloques, en lugar de anidado indexada.

Algoritmos de Join

- **Reunión por Asociación**
  - Supóngase que se quiere calcular ***r join s,*** con Br = 100 bloques y Bs = 400 bloques. M = 20.
  - Se pueden crear particiones para ***r*** *de tamaño 5* y de tamaño 80 para ***s***. Entonces se tiene que **N = 5**
  - La primera parte requiere la lectura de (Br + Bs) bloques para la lectura de los datos y la escritura de (Br + Bs) bloques para las particiones.
  - La reunión requerirá la lectura de (Br + Bs) bloques de la particiones.
    - Puede requerirse adicionalmente la lectura de 2N bloques ya que pueden existir bloques a medio llenar.
  - Entonces el costo es total es de :
    - 3(Br + Bs) + 2N lect/esc de bloques.
    - 3(400 + 100) + 2 * 5 = 1510.
  - En este caso N no supera a M, por lo que no se producen lecturas/escrituras adicionales debido a operaciones de división recursiva

Algoritmos de Join

- **Reunión por Asociación**
  - En el caso de división recursiva, se necesita ciclos en los cuales las particiones son dividas en un factor de M-1.
  - El numero de ciclos de división recursiva esta dado por $\lceil log_{M-1}(B_s) - 1 \rceil$.
  - Cada divide ambas relaciones por lo que el costo de inicial de particionamiento es:
    - 2* (Br + Bs) * $\lceil log_{M-1}(B_s) - 1 \rceil$ + (Br + Bs) = (Br + Bs) * (2 * $\lceil log_{M-1}(B_s) - 1 \rceil$ + 1)
    - Suponiendo Br = 4000 y Bs = 1000, entonces el costo es: (4000 + 1000) * (2 * $\lceil log_{19}(1000) - 1 \rceil$ + 1) = 25000
  - En el caso de que ambas relaciones quepan en memoria el costo del algoritmo deriva en (Br + Bs).

Algoritmos de Join

- **Reunión por Asociación Híbrida**
  - Se aplica en el caso de que la memoria sea grande pero que ninguna de las relaciones pueda caber en memoria.
  - **La idea es mantener las primeras particiones de la relación interna en memoria sin escribirlas a disco.**
  - **Luego al particionar la relación externa se puede aprovechar los datos de la particiones en memoria para escribir el resultado, pudiéndose obviar las lecturas/escrituras de las primeras particiones de la relación externa.**

Algoritmos de Join

- **Reunión por Asociación Híbrida**
  - Suponga que Br = 400, Bs = 100 y M = 26.
  - Se particiona s en 5 particiones de 20 bloques, la primera queda en memoria consumiendo 20 bloques y ahorrándose la lectura y escritura de los mismos.
  - R es particionada en 5 particiones de 80 bloques, para las tuplas que deben ir a la primera partición se realiza en cálculo de la reunión en vez de escribirlas y leerlas más tarde, ahorrándose la lectura/escritura de 80 bloques.
  - El costo esta dado por:
    - 3 * (80 + 320), correspondientes al costo de procesar de la manera normal las 4 particiones de cada relación, más
    - 80 + 20 lecturas de bloques, correspondientes las primeras particiones de cada relación procesadas directamente.
    - Dando un total de 1300 operaciones contra un total de 1500 al aplicar el algoritmo normal.

Algoritmos de Join con operaciones complejas

- **Condición Conjuntiva:**

$$r \ \ join_{\theta 1 \wedge \theta 2 \wedge \ldots \wedge \theta n} s$$

  - Se aplica uno de los algoritmos de bucles anidados, ó
  - Computar el resultado para una condición ***r join*** $_{\theta i}$ ***s***
    - Sobre el resultado se seleccionan las filas que cumplen con las demás condiciones

$$\theta_1 \wedge \ldots \wedge \theta_{i-1} \wedge \theta_{i+1} \wedge \ldots \wedge \theta_n$$

- **Condición disyuntiva:**

$$r \ \ join_{\theta 1 \vee \theta 2 \vee \ldots \vee \theta n} s$$

  - Se aplica uno de los algoritmos de bucles anidados, ó
  - Calcular cada operación individualmente y unir los resultados:

$$(r \ \ join_{\theta 1} s) \cup (r \ join_{\theta 2} s) \cup \ldots \cup (r \ join_{\theta n} s)$$

Join Externos

- Se calculan:
  - Realizando una operación de Join interno con los algoritmos descritos y luego agregando las filas no reunidas de una o ambas relaciones complementadas con valores nulos.
  - Modificando los algoritmos de Join interno.
- Para el join por mezcla, **r left outer join s** puede calcularse como:
  1. Caculando **r join s** y unir su resultado al resultado de la operación Π_R **(r join s) – r** complementadas con valores nulos, ó
  2. Modificando el algoritmo de reunión por mezcla: Durante la mezcla para las filas de **r** no reunidas añadirlas al resultado complementadas con valores nulos.
- Para el join por asociación, **r left outer join s** puede calcularse como:
  1. Si **r** es la relación de **prueba**, añadirla al resultado sus filas complementadas al no encontrarse filas de **s** concordantes a través del índice de la partición.
  2. Si **r** es la relación de **construcción**, se marca las entradas del índice no utilizadas para luego agregarlas complementadas al resultado.

Evaluación de expresiones

- Hasta ahora se ha visto como se realizan individualmente las operaciones de un plan de ejecución.
- Para procesar un plan de ejecución se deben evaluar todas las operaciones en el mismo. Existen dos estrategias:
- **Materialización**
  - Se evalúa una operación a la vez, empezando por las expresiones de nivel más bajo del *árbol de operadores*
  - El resultado se **materializa** en un archivo temporal
  - Coste = coste operaciones + escritura de tablas temporales
  - **Double buffering**, para solapar computo y escrituras.
- **Encauzamiento**
  - Se evalúan varias operaciones simultáneamente
  - Los resultados de una operación se pasan a la siguiente utilizando una memoria intermedia, sin utilizar archivos temporales.
  - PERO... Entradas/Salidas de las operaciones no están disponibles todas a la vez.
  - No es siempre aplicables. Casos de ordenamiento y reunión por mezcla
  - Puede realizarse de dos formas: **Bajo Demanda** o **Dirigida por Productores**

Reglas del optimizador de Oracle

- Si la consulta NO tiene **where**, se realiza una *exploración completa*
- Cuando se consultan filas concretas, la BD puede utilizar un índice siguiendo las reglas:

REGLA 1: El índice no es utilizado cuando la **columna indexada** está **dentro de una expresión o en una función SQL**

```
SELECT * FROM pedido
WHERE TO_CHAR (dateped, 'DD/MM/YYYY') = '02/02/2001';
```

pero sí en:

```
SELECT * FROM pedido
WHERE dateped= TO_DATE ('02/02/2001', 'DD/MM/YYYY');
```

REGLA 2: El índice no es utilizado cuando la **columna indexada** es **comparada con nulo**

```
SELECT * FROM cliente
WHERE numcli IS NULL;
```

pero sí en:

```
SELECT * FROM cliente
WHERE numcli > -1;
```

Reglas del optimizador de Oracle

REGLA 3: El índice no es utilizado cuando la **columna indexada** es **comparada por diferencia**(!= ó NOT)

```
SELECT * FROM cliente
WHERE numcli != 1000;
```

pero sí en:

```
SELECT * FROM cliente
WHERE numcli < 1000 OR numcli > 1000;
```

REGLA 4: El índice sólo es utilizado cuando el 1º carácter de la izquierda es distinto de %

```
SELECT * FROM cliente
WHERE nomcli LIKE '%DUPONT';
```

pero sí en:

```
SELECT * FROM cliente
WHERE nomcli LIKE 'DUPONT%';
```

REGLA 5: Un **índice compuesto** es utilizado si la primera columna del índice está declarada en la consulta

Reglas del optimizador de Oracle

REGLA 6: Un índice puede ser utilizado en una **operación de ordenación** si:

- La columna indexada es obligatoria (declarada como NOT NULL)
- No hay otros índices utilizables, de GROUP BY, de DISTINCT, etc. en la sentencia.

REGLA 7: En el caso de **AND**, Oracle utiliza hasta 5 índices concurrentes. Si entre ellos hay un índice único, sólo este será utilizado.

REGLA 8: En el caso de **OR**, el optimizador descompone la sentencia en tantas partes como condiciones existan separadas por este operador. Es suficiente que una condición no disponga de índice para que el optimizador no utilice ningún índice

Reglas del optimizador de Oracle

REGLA 9: JOIN

- Si las columnas de join NO están indexadas => se ordena cada tabla secuencialmente y se fusionan verificando la condición de join.
- Si una de las columnas del join está indexada => se elige como *tabla directriz* aquella que no disponga de índice. Esta tabla será recorrida secuencialmente.
- Si las dos columnas del join están indexadas => se selecciona como tabla directriz aquella que aparece en último lugar en la cláusula FROM.
- En caso de un join con condiciones de selección => se asigna una nota por cada condición en base a una *tabla de prioridades* . La tabla con menor nota será tomada como tabla directriz.

REGLA 10:

- En caso de **subconsultas,** la tabla directriz es la de la subsentencia