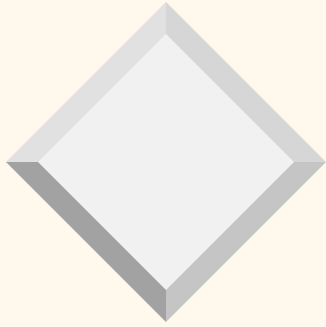


# *Evaluación de Operaciones Relacionales*

## Tema 14



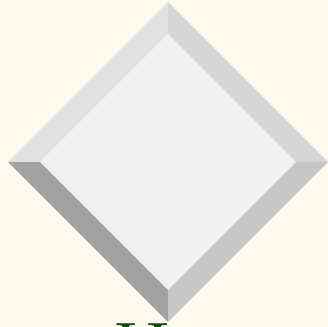
# *Introducción*

- ❖ Los operadores relacionales son las piezas principales de la evaluación de las consultas. Las consultas, escritas en SQL, se trasladan a un **optimizador de consultas**, que basado en la manera de como están almacenados los datos produce un eficiente **plan de ejecución** para la evaluación de la consulta.
- ❖ Un buen plan de ejecución no es solamente escoger la implementación de cada operador relacional que aparece en la consulta, también el orden en que se aplican afecta el costo.




## *Técnicas para desarrollar el algoritmo de cada operador*

- ❖ **Iteración:** Examina todas las tuplas de la relación de entrada. A veces, en vez de examinar las tuplas, podemos examinar las entradas de índice (que son mas pequeñas) que contienen todos los campos necesarios.
- ❖ **Indexar:** Si se especifica una condición de selección o de join, es mejor usar un índice para examinar solamente las tuplas que satisfacen la condición.
- ❖ **Particionar:** Al particionar las tuplas sobre una llave de ordenamiento, a menudo es posible descomponer una operación en una colección de operaciones menos costosas sobre las particiones. Ordenar y hashing son 2 de las técnicas mas comunes de particionamiento.



## *Rutas de Acceso*

- ❖ Hay mas de una forma de recuperar las tuplas de una relación, depende de la existencia de índices y la presencia de condiciones de selección en una consulta. Las formas alternativas para recuperar tuplas son llamadas **rutas de acceso**.
- ❖ Una ruta de acceso puede ser 1) explorar un archivo o 2) usar un índice y una condición de selección.
- ❖ La **selectividad** de una ruta de acceso es el número de páginas (tanto de índices como de datos) recuperadas.
- ❖ La ruta de acceso mas selectiva es la que recupera menos páginas.



# Operaciones relacionales


- ❖ Vamos a considerar como implementar:
  - Selección ( $\sigma$ ) Selecciona un subconjunto de tuplas de una relación.
  - Proyección ( $\pi$ ) Borra las columnas no deseadas de una relación.
  - Join ( $\bowtie$ ) Permite combinar dos relaciones.
  - Diferencia ( $-$ ) Tuplas en la relación 1 pero no en la relación 2.
  - Unión ( $\cup$ ) Tuplas en la relación 1 o en la relación 2.
  - Agregadas (SUM, MIN, etc.) y GROUP BY
- ❖ Dado que de cada operación se obtiene una relación, las operaciones pueden ser *compuestas*! Luego de estudiar las operaciones, discutiremos como *optimizar* las consultas formadas por una composición de ellas.



## Esquema de ejemplo

Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)  
Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

- ❖ Similar al esquema anterior; pero se agregó *rname*.
- ❖ Reserves:
  - Cada tupla es de 40 bytes de longitud, 100 tuplas por página, 1000 páginas.
- ❖ Sailors:
  - Cada tupla tiene 50 bytes de longitud, 80 tuplas por página, 500 páginas.



## *Joins con una condición de igualdad*

```
SELECT *  
FROM   Reserves R1, Sailors S1  
WHERE  R1.sid=S1.sid
```

- ❖ En álgebra:  $R \bowtie S$ . Muy común! Debe ser cuidadosamente optimizado.  $R \times S$  es muy grande; también,  $R \times S$  seguido por una selección es muy ineficiente.
- ❖ Asumamos: Hay  $M$  páginas en  $R$ ,  $p_R$  tuplas por página,  $N$  páginas in  $S$ ,  $p_S$  tuplas por página.
  - En nuestro ejemplo,  $R$  es Reserves y  $S$  es Sailors.
- ❖ Mas adelante consideraremos condiciones de join más complejas.
- ❖ *Costo métrico*: # de operaciones de Lectura/Escritura  
Ignoraremos el costo de salida.



## Join por Ciclos anidados

for each tuple  $r$  in  $R$  do

for each tuple  $s$  in  $S$  do

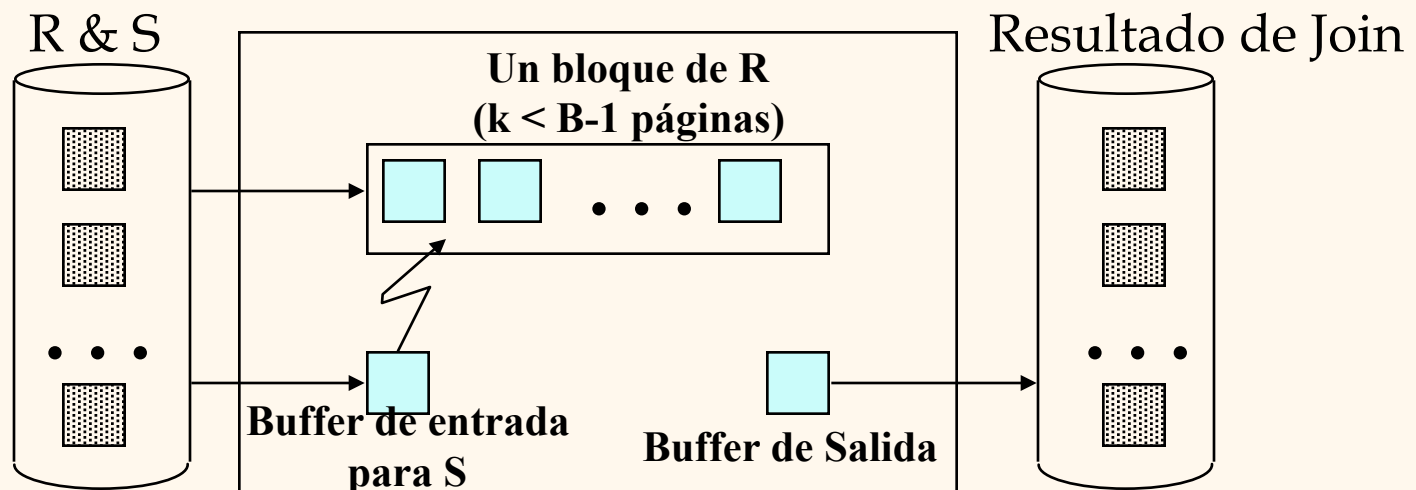
if  $r_i == s_j$  then add  $\langle r, s \rangle$  to result

- ❖ Para cada tupla de la relación externa  $R$  (*outer*), se explorará completa la relación interna  $S$  (*inner*).
  - Costo:  $M + p_R * M * N = 1000 + 100 * 1000 * 500$  I/O's
- ❖ El ciclo anidado para el join está orientado a páginas:  
Para cada *página* de  $R$ , obtener cada *página* de  $S$ , y escribir a la salida cada par de tuplas  $\langle r, s \rangle$  que cumplen la condición de join, donde  $r$  está en la página- $R$  y  $s$  está en la página- $S$ .
  - Costo:  $M + M * N = 1000 + 1000 * 500$
  - Si la relación externa es la menor ( $S$ ), costo =  $500 + 500 * 1000$



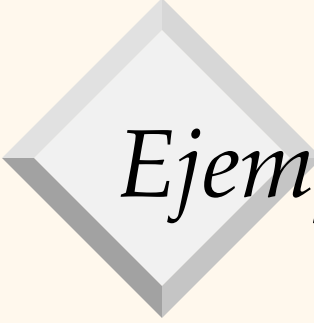
# *Join por Ciclos anidados en bloques*

- ❖ Se utiliza una página como buffer de entrada para explorar la relación interna  $S$ , una página como buffer de salida, y el resto de páginas para mantener un bloque de la relación externa  $R$ .
  - Por cada tupla  $r$  del bloque  $R$  que coincida con  $s$  en la página  $S$ , agregar  $\langle r, s \rangle$  al resultado. Luego leer el siguiente bloque de  $R$ , explorar  $S$ , etc.




## *Join por Ciclos anidados en bloques*

- ❖ Si tenemos suficiente memoria para mantener la relación menor (S) con por lo menos 2 páginas buffer extras. Podemos leer de la relación menor y usar un de los buffer extras para explorar la relación mayor R. Para cada tupla  $s$  de S, se verifica R y se coloca en la salida una tupla  $\langle r, s \rangle$  para las tuplas  $s$  que verifican la condición. El segundo buffer extra se usa como buffer de salida
- ❖ Cada relación se explora una sola vez, para un costo total en operaciones I/O de  $M+N$ , que es óptimo.
- ❖ Si hay memoria suficiente disponible, un refinamiento importante es El costo es  $M$  por las lecturas en R. S se explora  $\lceil M/(B-2) \rceil$  veces y cada exploración cuesta  $N$ .
- ❖ Así se realizan 10 exploraciones en S, cada una en 500 IO. El Total es:  $1000 + 10*500 = 6000$  IO



## *Ejemplo de Ciclos anidados en bloques*

- ❖ **Costo: Explorar relación externa + #bloques de relacion extena \* explorar relación interna**
  - #bloques de la relación externa = [#de páginas de la relación externa / tamaño de bloque]
- ❖ Con Reserves (R) como la relación externa, y 100 páginas de R:
  - El costo de explorar R es 1000 I/Os; un total de 10 *bloques*.
  - Por cada bloque de R, se explora Sailors (S); 10\*500 I/Os.
  - Si hay espacio solo para 90 páginas de R, es posible explorar S 12 veces.
- ❖ Con un bloque de 100-páginas de Sailors como relación externa:
  - Costo de explorar S es 500 I/Os; con un total de 5 bloques.
  - Por cada bloque S, se explora Reserves; 5\*1000 I/Os.



## *Join por Ciclos anidados con índice*

for each tuple  $r$  in  $R$  do  
    for each tuple  $s$  in  $S$  where  $r_i == s_j$  do  
        add  $\langle r, s \rangle$  to result

- ❖ Si hay un índice sobre la columna de la condición del join de una de las relaciones (sea  $S$ ), podemos hacer de esta la interna y usar el índice.
  - Costo:  $M + (M * p_R) * \text{costo de buscar tuplas de } S \text{ que cumplan}$
- ❖ Por cada tupla  $R$ , el costo de probar el índice  $S$  es alrededor de 1 o 2 para un índice tipo hash, 2 a 4 para un índice tipo árbol B+. El costo de buscar las tuplas de  $S$  (si asumimos la Alt. (2) o (3) para entradas de datos) depende de si es o no cluster.
  - Índice clustered : 1 I/O (normalmente), unclustered: hasta 1 I/O por cada tupla  $S$  que coincida.



## *Ejemplo de Join por Ciclos anidados con índice*

- ❖ Índice tipo hash (Alt. 2) sobre *sid* de Sailors (como interna):
  - Explorar Reserves: 1000 páginas I/Os,  $100 \times 1000$  tuplas.
  - Por cada tupla de Reserves: 1.2 I/Os para obtener la entrada de datos en el índice, mas 1 I/O para obtener la tupla de Sailors que coincida.  
Total: 220,000 I/Os.
- ❖ Índice tipo hash (Alt. 2) sobre *sid* de Reservas (como interna):
  - Explorar Sailors: 500 pagina I/Os,  $80 \times 500$  (40,000) tuplas.
  - Por cada tupla de Sailors: 1.2 I/Os para buscar la página de índices con las entradas de datos ( $1.2 \times 40,000$  I/O), mas el costo de recuperar las tuplas de Reserves que cumplan. Si se asume que estan distribuidas uniformemente, 2.5 reservaciones por marinero ( $100,000 / 40,000$ ). El costo de recuperalas es 1 a 2.5 I/Os dependiendo si el índice es clustered. (40,000 a 100,000 I/O)



## Sort-Merge Join ( $R \bowtie_{i=j} S$ )

- ❖ Ordenar R y S sobre la columna de join, se exploran para hacer una unión, y se muestra el resultado.
  - Se explora R hasta que la R-tupla actual  $\geq$  S-tupla actual, luego se explora S hasta que la S-tupla actual  $\geq$  R-tupla actual; se hace hasta que R-tupla actual = S-tupla actual.
  - En este punto, todas las tuplas de R con el mismo valor en  $R_i$  (*grupo actual de R*) y todas las tuplas de s con el mismo valor en  $S_j$  (*grupo actual de S*) coinciden; Por cada tupla r en la partición actual de R, debemos explorar todas las tuplas s en la actual partición de S, y mostrar la tupla unida  $\langle r, s \rangle$ . Luego termina explorando R y S.
- ❖ R es explorado una vez; cada grupo S se explora una vez por tupla de R que coincida.

## *Ejemplo de Sort-Merge Join*

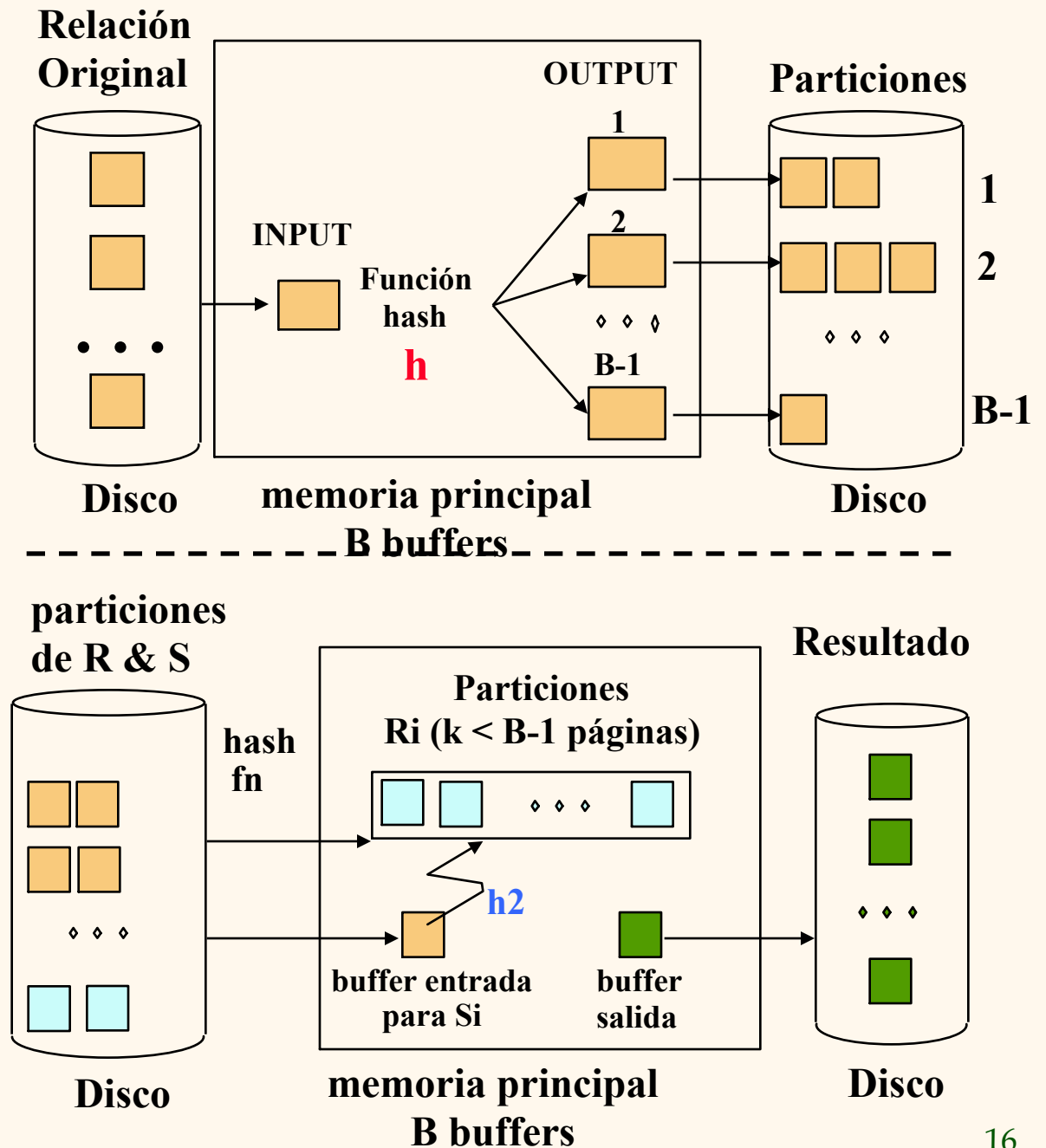
<u>sid</u>	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

<u>sid</u>	<u>bid</u>	<u>day</u>	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin


- ❖ Costo:  $M \log M + N \log N + (M+N)$ 
  - El costo de explorar,  $M+N$

# Hash-Join

- ❖ Particionar cada relación usando una función hash **h**: Las tuplas R en la partición i coinciden con tuplas S en la partición i.
- ❖ Se explora la partición de S que coincide, se buscan coincidencias.







## *Costo de Hash-Join*

- ❖ En la fase de partición, lectura+escritura de ambas relaciones;  $2(M+N)$ . En la fase de comparación, las lecturas de las relaciones;  $M+N$  I/Os.
- ❖ Para el ejemplo, esto es un total de 4500 I/Os.




# Condiciones Generales de Join

- ❖ Igualdades sobre varios atributos (Ejemplo, *R.sid=S.sid AND R.rname=S.sname*):
  - Para ciclos anidados con índice, construir un índice sobre *<sid, sname>* (Si S es la interna); o usar los índices existentes sobre *sid* o *sname*.
  - Para Sort-Merge Join y Hash Join, ordenar/particionar sobre la combinación de columnas de join.
- ❖ Condiciones de desigualdad (Ej., *R.rname < S.sname*):
  - Para ciclos anidados con índice, necesario índice tipo árbol B+ clustered. El rango de pruebas sobre la relación interna; el número de coincidencias va a ser probablemente mucho mayor que para un join con igualdades.
  - Hash Join, Sort Merge Join no son aplicables.
  - Ciclos anidados por bloques podría ser el mejor método de join en este caso.

## Selección Simple

```
SELECT *  
FROM   Reserves R  
WHERE  R.rname < 'C%'
```

- ❖ De la forma  $\sigma_{R.\text{atributo } op \text{ valor}}(R)$
- ❖ El tamaño del resultado es aproximadamente *el tamaño de R \* un factor de reducción*; mas adelante se considerará como calcular dicho factor de reducción.
- ❖ **Sin índice, no ordenado:** Se debe explorar toda la relación **el costo es M I/O's** (# páginas en R).
- ❖ **Sin índice y datos ordenados:** Se usa el ordenamiento para hacer una búsqueda binaria para localizar la primera tupla que satisface la condición, luego iniciando en esta posición explora la relación R para recuperar todas la tuplas que satisfacen la condición hasta que ésta no se satisface. El método de acceso es una búsqueda en archivo ordenado con una condición de selección. Costo: Costa de la búsqueda binaria  $O(\log_2 M)$  + Costo de recuperar las tuplas.



## Selección Simple (cont.)

- ❖ **Con índice sobre el atributo de la selección:** Se usa el índice para localizar las entradas de datos que califican, luego se recuperan los registros de datos correspondientes. (Índice tipo Hash útil únicamente en selecciones con igualdad)
- ❖ **Índice tipo árbol B+:** Si se posee un índice tipo árbol B+ de tipo clustered sobre R.atributo, y en la condición de selección *op* no es una igualdad entonces la mejor estrategia es usar el índice. En igualdades es bueno pero Hash es mejor. Si el índice no es cluster, el costo de usar el índice depende del número de tuplas que satisfacen la condición. El costo típico de identificar la hoja con la página inicial es 2 o 3 I/O's. El costo de explorar las hojas para localizar las entradas de datos que califican depende de la cantidad de dichas entradas.

## *Usando un índice para selecciones*

- ❖ El costo depende de # tuplas que califican, y si el índice es o no clustered.
  - El costo de buscar las entradas de datos que califican (normalmente pequeño) mas el costo de recuperar los registros (puede ser muy grande si no es clustered).
  - Ejemplo: Asumiendo una distribución uniforme de nombres, si alrededor de 10% de las tuplas cumplen (100 páginas, 10000 tuplas). Con un índice clustered, costo menor de 100 I/Os; si no es clustered, hasta 10000 I/Os!
- ❖ *Un refinamiento para un índice no clustered:*
  1. Buscar las entradas de datos que califican.
  2. Ordenar los rid's de los registros de datos a recuperar.
  3. Recuperar los rids en orden. Esto asegura que cada página de datos será bloqueada una sola vez.

## Continuación

- ❖ **Índice tipo Hash, y selección con igualdad:** Si se posee un índice tipo hash sobre R.atributo y op es una igualdad, la mejor forma de implementar la selección es utilizando el índice para recuperar las tuplas que satisfacen.

El costo incluye 1 o 2 I/O para recuperar el cubo en el índice + el costo de recuperar las tuplas de R que califiquen. El costo de recuperar las tuplas depende de la cantidad de ellas y si el índice es o no clustered.

Ejemplo: WHERE R.rname='Joe'. Hay un índice tipo hash y no clustered sobre rname. Supongase que hay hechas 100 reservaciones por la persona llamada Joe. El costo típico de recuperar la página del índice que contiene los rids de tales reservaciones 1 o 2 I/O. El costo de recuperar las 100 tuplas de reservaciones puede variar de 1 a 100, dependiendo de cómo estén distribuidos los registros en las páginas de R y el orden en que se recuperan.

# Condiciones Generales de Selección

➡  $(day < 8/9/94 \text{ AND } rname = 'Paul') \text{ OR } bid = 5 \text{ OR } sid = 3$

- ❖ En general la condición de selección es una combinación “booleana” de términos de la forma: *atributo op constante* o *atributo1 op atributo2*.
- ❖ Estas condiciones se convierten primero a la forma normal conjuntiva (CNF) (colección de expresiones conectadas con el operador  $\wedge$ ). En el ejemplo, cada expresión posee uno o más términos conectados por  $\vee$  (disyunción):  $(day < 8/9/94 \text{ OR } bid = 5 \text{ OR } sid = 3) \text{ AND } (rname = 'Paul' \text{ OR } bid = 5 \text{ OR } sid = 3)$
- ❖ Solo analizaremos el caso sin conectivo OR (una conjunción de términos de la forma *atributo op valor*).
- ❖ Un índice puede reúne (en una conjunción) términos que posean atributos que sean *prefijo* de la llave de búsqueda.
  - Índice sobre  $\langle a, b, c \rangle$  Podría ser  $a = 5 \text{ AND } b = 3$ , pero no  $b = 3$ .



## Condiciones Generales de Selección

- ❖ Si se tiene un índice hash sobre la llave de búsqueda (rname, bid, sid), podemos usar el índice para recuperar tuplas que satisfacen la condición  $\text{rname} = \text{'Joe'} \wedge \text{bid} = 5 \wedge \text{sid} = 3$ . Pero no con  $\text{rname} = \text{'Joe'} \wedge \text{bid} = 5$ , o alguna condición sobre date.
- ❖ Sí se tiene un índice árbol B+, se puede usar con  $\text{rname} = \text{'Joe'} \wedge \text{bid} = 5 \wedge \text{sid} = 3$  y también con  $\text{rname} = \text{'Joe'} \wedge \text{bid} = 5$ , pero no con  $\text{bid} = 5 \wedge \text{sid} = 3$
- ❖ Si se tiene un índice (hash o árbol) sobre la llave de búsqueda (bid, sid) y la condición de selección  $\text{rname} = \text{'Joe'} \wedge \text{bid} = 5 \wedge \text{sid} = 3$ , es posible usar el índice para recuperar las tuplas que satisfacen  $\text{bid} = 5 \wedge \text{sid} = 3$ , pero se debe aplicar la condición adicional sobre rname.






# *Condiciones Generales de Selección*

- ❖ En general, las siguientes reglas definen si se utiliza un índice cuando la condición de la selección esta en CNF:
  - Un índice tipo hash verifica la condición de la selección sin disyunciones si hay términos de la forma atributo=valor para cada atributo de la llave de búsqueda del índice.
  - Un índice tipo árbol verifica una condición de selección sin disyunciones si hay términos de la forma atributo op valor para cada atributo en un prefijo de la llave de búsqueda del índice.

## Dos aproximaciones de Selecciones Generales sin disyunción

- ❖ Primera aproximación: Buscar la *ruta de acceso mas selectiva* (explorar el archivo o usar un índice simple que coincida con algunas expresiones de la conjunción), se recuperan las tuplas y se aplican los términos restantes que no coincidan con el índice:
  - *La ruta de acceso mas selectiva:* Uso de un índice o exploración del archivo que estimamos requerirá la menor cantidad de I/Os.
  - Los términos que coincidan con el índice reducen el número de tuplas recuperadas; los otros términos se usan para descartar algunas de las tuplas, pero no afecta en número de tuplas/página recuperadas.



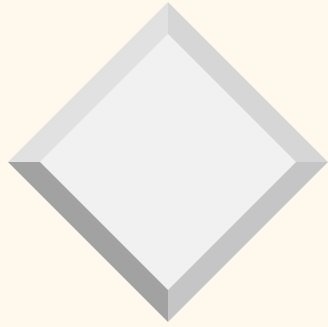
## *Intersección de Rids*

- ❖ Segunda Aproximación (Si se poseen 2 o más índices que coinciden que usan Alternativa 2 o 3 para las entradas de datos):
  - Obtener los conjuntos de rids de registros de datos usando las índices que coinciden.
  - Luego *interceptar* estos conjuntos de de rids
  - Recuperar los registros y aplicar los restantes términos de la condición.



## *Selecciones con disyunción*

- ❖ Si uno de los términos requiere que se explore el archivo porque no se posee índices ni ordenamiento, verificar estas expresiones por si mismas requiere que se explore el archivo.
- ❖ Si cada término dentro de la disyunción tiene un índice que coincide, es posible recuperar las tuplas candidatas usando los índices y luego hacer una unión. Y si la Alternativa es 2 o 3, una mejor aproximación es tomar la unión de los rid y ordenarlos antes de recuperar los registros de datos que califican.
- ❖ La mayoría de los sistemas actuales no manejan eficientemente condiciones con disyunciones, y se concentran en la optimización de las selecciones sin disyunción.



## *Operación de Proyección*

- ❖ Para implementar la proyección se hace lo siguiente:
    - Remover los atributos no deseados.
    - Eliminar las tuplas duplicadas que se producen.
- El segundo paso es el difícil. Hay 2 algoritmos básicos, uno basado en ordenamiento y otro en hashing. (pero ambos son casos de particionar)



# *Proyección basada en ordenamiento*

- ❖ El algoritmo basado en ordenamiento tiene los siguientes pasos (conceptualmente):
  - Explorar R y producir un conjunto de tuplas que contengan únicamente los atributos deseados. Costo (  $M + T$  )
  - Ordenar este conjunto de tuplas usando la combinación de todos sus atributos como llave de búsqueda (Costo  $O(T \log T)$ )
  - Explorar el resultado ordenado, comparar las tuplas adyacentes y eliminar los duplicados. (Costo  $T$ )
  - Para Reserves ( $M=1000$ , 10 bytes de longitud de cada tupla en la relación temporal (40 bytes la original)), y 20 páginas de buffers para ordenar (2 pasadas) :  $1000 + 250 + 2*2*250 + 250 = 25000$ .

# *La operación de Proyección*

```
SELECT  DISTINCT  
        R.sid, R.bid  
FROM    Reserves R
```

- ❖ Una aproximación basada en ordenamiento:
  - **Modificar el Paso 0 del ordenamiento externo para eliminar los campos no deseados.** Se obtienen corridas de 2B páginas, pero las tuplas de las corridas son mas pequeñas que las tuplas de entrada. (La fracción del tamaño depende de el número y tamaño de los campos eliminados)
  - **Modificar los pasos de unión para eliminar duplicados.** La cantidad de tuplas del resultado es menor que en la entrada. (La diferencia depende de la cantidad de duplicados)

## Proyección basada en hash

- ❖ *Fase de partición ( $M + T IO$ )*: Se lee  $R$  usando un buffer de entrada. Para cada tupla se descartan los campos no necesarios, se aplica una función hash  $h1$  para escoger uno de  $B-1$  buffers de salida.
  - El resultado son  $B-1$  particiones (de tuplas sin los campos no necesarios), garantizando que 2 tuplas de diferentes particiones son distintas.
- ❖ *Fase de eliminación de duplicados ( $T IO$ )* : Para cada partición se lee y se construye en memoria una tabla hash, luego se descartan los duplicados.
  - Si la partición no cabe en la memoria, se puede aplicar el algoritmo de proyección basado en hash recursivamente para esta partición.





# Operaciones con conjuntos

- ❖ La intersección y el producto cartesiano son casos especiales de el join.
- ❖ La Unión (Distinct) y Diferencia son similares.
- ❖ Aproximación de unión basada en ordenamiento:
  - Se ordenan ambas relaciones (sobre todos los campos).
  - Se exploran las relaciones y se unen.
  - *Alternativa*: Unir las corridas del Paso 0 de *ambas* relaciones
- ❖ Aproximación de unión basada en hash:
  - Particionar R y S usando una función hash  $h$ .
  - Para cada partición de S, construir en memoria una tabla hash (usando  $h_2$ ), explorar la correspondiente partición de R y agregar las tuplas al resultado descartando duplicados.

# Operaciones Agregadas (*AVG, MIN, etc.*)

## ❖ Sin agrupación:

- En general, requiere explorar la relación.
- Si se posee un índice cuya llave de búsqueda incluye todos los atributos en las cláusulas `SELECT` o `WHERE`, es posible explorar únicamente el índice.

## ❖ Con agrupación:

- Ordenar sobre los atributos del ordenamiento, luego explorar la relación y computar las funciones agregadas para cada grupo.
- Se puede hacer una aproximación similar basada en hashing sobre los atributos de agrupación.
- Si se posee un índice tipo árbol cuya llave de búsqueda incluye todos los atributos de las cláusulas `SELECT`, `WHERE` y `GROUP BY`, es posible explorar únicamente el índice. Si los atributos de agrupación forman un prefijo de la llave de búsqueda, es posible recuperar las entradas de datos y luego las tuplas.