PROCESAMIENTO Y OPTIMIZACION DE CONSULTA

Vamos a calcular el costo de una consulta. Hay varias opciones de realizar ciertas consultas. Algunas utilizan operaciones de mayor costo que otras.

El objetivo es obtener la forma menos costosa de resolverla.

Los SGBD's guardan distinto tipo de información de catálogo que es utilizada para estimar costos y optimizar las consultas. Nosotros utilizaremos la siguiente notación:

- n(R): Cantidad de tuplas de la relación R.
- B(R): Cantidad de bloques de almacenamiento que ocupa R.
- V(A,R): Cantidad de valores distintos que adopta el atributo A en R (variabilidad).
- F(R): Cantidad de tuplas de R que entran en un bloque (factor de bloque). $F(R) = \frac{n(R)}{B(R)} \Rightarrow B(R) = \lceil \frac{n(R)}{F(R)} \rceil$.

La variabilidad seria como un select count distinct de un atributo en la tabla. Este no siempre se calula constantemente xq es más difícil de calcular. Para saber si un atributo esta o no voy a tener que ver si ya apareció y eso es re costoso.

También se almacena información sobre la cantidad de niveles que tienen los índices construídos, y la cantidad de bloques que ocupan sus hojas.

- Height(I(A, R)): Altura del índice de búsqueda I por el atributo A de la relación R.
- Length(I(A, R)): Cantidad de bloques que ocupan las hojas del índice I.

El largo en realidad no lo vamos a usar nunca. **Solo la altura**

Plan de consulta y plan de ejecución

La optimización de una consulta se inicia con una expresión en álgebra relacional.

La expresión se optimiza a través de una heurística y utilizando reglas de equivalencia, obteniendo un plan de consulta.

Luego, cada plan de consulta lógico se materializa para obtener un plan de ejecución en el que se indica el procedimiento físico: estructuras de datos a utilizar, índices, algoritmos a utilizar, etc.

Para comparar distintos planes de ejecución, necesitamos estimar su costo. Algunos de los factores que inciden en la performance son:

- El costo de acceso a disco (lectura o escritura)
- El costo de procesamiento
- El costo de uso de memoria
- El costo de uso de red

Solo estudiaremos los costos de acceso a disco

REGLAS DE EQUIVALENCIA

- Selección
  - $\sigma_{c_1 \wedge c_2 \wedge \ldots \wedge c_n}(R) = \sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R))\ldots))$ (Cascada)
  - $\sigma_{c_1 \vee c_2 \vee \ldots \vee c_n}(R) = \sigma_{c_1}(R) \cup \sigma_{c_2}(R) \cup \ldots \cup \sigma_{c_n}(R)$
  - $\sigma_{c_1}(\sigma_{c_2}(R)) = \sigma_{c_2}(\sigma_{c_1}(R))$ (Conmutatividad)

Es lo mismo un and de muchas condiciones a aplicarlas en cascada. Lo mismo con el or.

- Proyección
  - $\pi_{X_1}(\pi_{X_2}(\ldots(\pi_{X_n}(R))\ldots)) = \pi_{X_1}(R)$ (Cascada)
  - $\pi_X(\sigma_{cond}(R)) = \sigma_{cond}(\pi_X(R))$ (Conmutatividad con σ)

Si hacemos muchas proyecciones es lo mismo a que si solo hacemos la de más afuera.

La conmutatividad es SOLO cuando la condición involucra atributos de la proyección.

- Producto cartesiano y junta
  - $R \times S = S \times R$ (Conmutatividad)
  - $R * S = S * R$
  - $(R \times S) \times T = R \times (S \times T)$ (Asociatividad)
  - $(R * S) * T = R * (S * T)$

- Operaciones de conjuntos
  - $R \cup S = S \cup R$ (Conmutatividad)
  - $R \cap S = S \cap R$
  - $(R \cup S) \cup T = R \cup (S \cup T)$ (Asociatividad)
  - $(R \cap S) \cap T = R \cap (S \cap T)$

- Otras mixtas
  - Dado $\sigma_c(R * S)$, si c puede escribirse como $c_R \wedge c_S$, con c_R y c_S involucrando sólo atributos de R y de S respectivamente, entonces:

    $$\sigma_c(R * S) = \sigma_{c_R}(R) * \sigma_{c_S}(S)$$

    (Distribución de la selección en la junta)
  - Dado $\pi_X(R * S)$, si todos los atributos de junta están incluídos en X, entonces llamando X_R y X_S a los atributos de R y S que están en X respectivamente:

    $$\pi_X(R * S) = \pi_{X_R}(R) * \pi_{X_S}(S)$$

    (Distribución de la proyección en la junta)

HEURISTICA DE OPTIMIZACION

La aplicación de las reglas de equivalencia a una expresión algebraica para obtener otra de menor costo se conoce como optimización algebraica.

Las siguientes son algunas reglas generales utilizadas para optimizar algebraicamente una consulta. Se busca que los resultados intermedios sean los menores posibles:

1. Realizar las **selecciones lo más temprano posible**.
2. **Remplazar** (productos cartesianos + selección) **por juntas** siempre que sea posible.

3. Proyectar para descartar los atributos no utilizados lo antes posible. ***Entre la selección y la proyección, priorizar la selección***.
4. En caso de que hayan **varias juntas, realizar aquella más restrictiva primero**. ***Optar por árboles left-deep ó right-deep para acotar las posibilidades.***
   Esto se refiere a que si tenemos un árbol, donde cada nodo es un operador de algebra relacional, vamos a tratar de armar arboles donde solo agregamos nodos a la derecha (left Deep) o solo a la izquierda (right Deep)

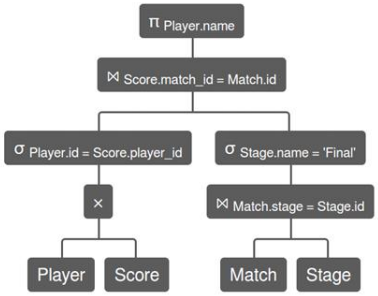
Ejemplo: 210 World Cup Dataset

Queremos calcular que jugadores convirtieron algún gol en la dinal del mundial

- Esquema de base de datos relacional:
  - Continent(id, name)
    (1, 'Africa')
  - NationalTeam(id, name, group, short_name, continent)
    (1, 'South Africa', 'A', 'RSA', 0)
  - Match(id, home, away, match_datetime_gmt, stage)
    (1, 1, 2, '2010-06-11 14:00:00', 1)
  - Player(id, name, birth_date, height, playing_position, local_club, national_team, national_team_tshirt)
    (53, 'Edinson Cavani', '1987-02-14', 188, 'FW', 'Palermo [ITA]', 3, 7)
  - Score(id, match_id, team_id, player_id, minute, score_type)
    (1, 1, 1, 8, '55', 1)
  - Stage(id, name)
    (3, 'Quarter-finals')
  - Asumiremos que "name" es siempre clave candidata.

Ejemplo: 2010 World Cup Dataset

Para calcular el listado de jugadores que convirtieron algún gol en la final del mundial, un motor de bases de datos construye el siguiente plan de consulta:



Aplique las heurísticas estudiadas para optimizar el plan.

Vamos a leer de abajo para arriba

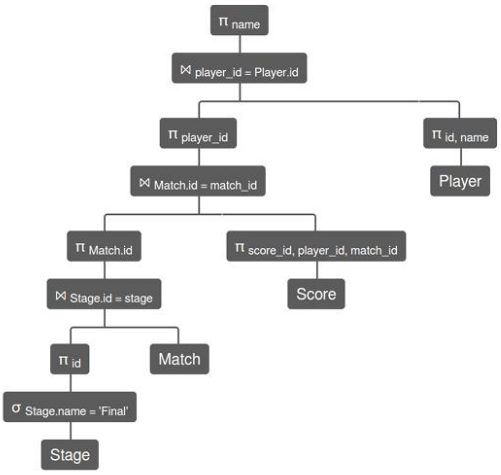
Del lado izquierdo vemos que hace un producto cartesiano con una selección, para obtener a los jugadores con sus goles. Esto lo podríamos mejorar. Del lado derecho si hace una junta, y luego lo filtra para quedarse con el Stage Final.

Hace un join para quedarse con los que hicieron goles en la final, y filtra los nombres de los jugadores.

Esto lo vamos a tener que optimizar.

Vamos a hacer un árbol intentando que sea left Deep. Podemos agarrar el Stage, y seleccionar la final (siempre hacer las selecciones lo antes posible), luego hacemos un join con match. Ahora tengo que hacer un join con score, porque con players no tengo como juntarlo sin hacer un produto cartesiano. Luego hago un join con jugador por player id.

Solución



INDICES

Los índices son estructuras de búsqueda almacenadas y actualizadas por el SGBD, que agilizan la búsqueda de registros a partir del valor de un atributo o conjunto de atributos.

Se pueden implementar con árboles, tablas de hash, etc.

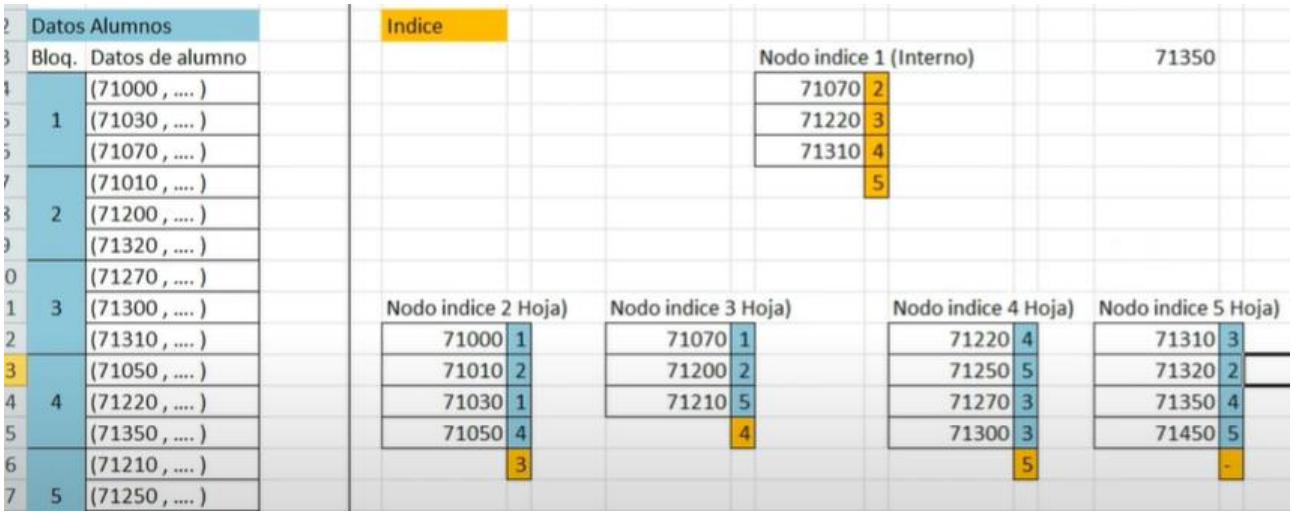
En los SGBD's los índices se clasifican en distintos tipos:

- **INDICE PRIMARIO:** Cuando el índice se construye sobre el campo de ordenamiento clave de un archivo ordenado de registros. Ejemplo: en la tabla tenemos un índice de padron y además la tabla esta ordenada por padrón.
- **INDICE DE CLUSTERING:** Cuando se construye sobre el campo de ordenamiento del archivo físico, pero este no es clave. Ejemplo: tengo alumnos y el índice es fecha de entrada. No es clave xq muchos pueden tener la misma fecha de entrada.
- **INDICE SECUNDARIOS:** Los índices que se construyen sobre campos que no son los campos de ordenamiento del archivo.

Obvs: un archivo solo puede tener un único índice primario o de clustering.

SQL no dispone de una sentencia estándar para la definición de índices, aunque la mayoría de los SGBDs tiene una sentencia del tipo CREATE INDEX.

Supongamos que tenemos los datos del alumno, y que entran 3 alumnos por bloque.



Acá el índice es SECUNDARIO, porque el índice es una clave pero los datos de alumnos NO están físicamente ordenados por el padrón.

Si no tenemos un índice, y tenemos que buscar los datos de un alumno por padron, debería levantar todos los bloques hasta encontrar el dato que me coincida con el padron. El costo de la búsqueda seria mas o menos 5 bloques, porque tengo 5 bloques.

Cuando tenemos un árbol (por ej un árbol B), tenemos otros tipos de bloque. Cada bloque es un nodo, y se dividen entre bloques iternos o bloques hojas. Los internos tienen las entradas del índice y un puntero mas que entradas que tiene. Si tiene 3 entradas, va a tener 4 punteros, y así.

El puntero me dice qué siguiente nodo tengo que ir a ver.

Por ejemplo: busco el padrón 71350. Voy al bloque interno, que me dice que un corte esta en el 71070. Los padrones que vienen antes de 71070 están en el bloque 2 del índice. Los que están entre el 71070 y el 71220, están en el bloque 3.

El 71350 es mayor al 71310 así que va a estar en el nodo hoja 5.

El nodo hoja tiene para cada clave en que nodo de los datos esta ese valor. Acá dice que el 71350 esta en el nodo 4 de datos. El puntero amarillo de los nodos hoja indica cual es el siguiente nodo hoja al que tengo que ir.

Ahora tuve 3 accesos en vez de 5 como vimos antes.

COSTO DE LOS OPERADORES

Recordamos que vamos a medir el costo según cuantos bloques yo tengo que acceder para cada operador.

Selección

Existen distintas estrategias de búsqueda, según los recursos que contamos. Vamos a analizar distintas situaciones para la comparación por =

Partimos de una selección básica del tipo $\sigma_{cond}(R)$, en donde *cond* es una condición atómica.

File scan vs. Index scan

FILE SCAN: recorren el/los archivo/s en busca de los registros que cumplen con la condición. Básicamente va a leer todo el archivo. Realiza una **Búsqueda lineal**: Consiste en explorar cada registro, analizando si se verifica la condición. Luego $cost(S1) = B(R)$ porque voy a recorrer todos los bloques

INDEX SCAN: utilizan un índice de búsqueda. La clave de calcular el costo es la altura del índice y la cantidad de tuplas que me va a devolver esa selección.

Cuando el índice es de árbol, utilizamos formulas que trabajan con la ***altura del árbol*** + una estimación de cuantas filas nos va a devolver

Búsqueda con índice primario: Cuando Ai es un atributo clave del que se tiene un índice primario.

Sólo una tupla puede satisfacer la condición. Con I = indice

Si utilizamos un árbol de búsqueda: $cost(S_{3a}) = Height(I(A_i,R)) + 1$

Si utilizamos una clave de hash: $cost(S_{3b}) = 1$

Busqueda con indice de clustering: Cuando Ai no es clave, pero se tiene un índice de ordenamiento (clustering) por él.

Las tuplas se encuentran contiguas en los bloques, los cuales estarán disjuntos.

$$cost(S_5) = Height(I(A_i, R)) + \left\lceil \frac{n(R)}{V(A_i, R) \cdot F(R)} \right\rceil$$

Busqueda con indice secundario: Cuando Ai no tiene un índice de clustering, pero existe un índice secundario asociado a él.

$$cost(S_6) = Height(I(A_i, R)) + \left\lceil \frac{n(R)}{V(A_i, R)} \right\rceil$$

Siempre es la altura + una cantidad extra que depende de la variabilidad.

Los cálculos que hemos visto pueden extenderse para otros tipos de comparación (<,≤,>,≥,=).

Selecciones complejas

Si la selección involucra la conjunción (osea un and) de varias condiciones simples, pueden adoptarse distintas estrategias:

- Si uno de los atributos tiene un índice asociado, se aplica primero esta condición, y luego se selecciona del resultado a aquellas tuplas que cumplen con las demás condiciones.
- Si hay un índice compuesto que involucra a atributos de más de una condición, se utiliza este índice y luego se seleccionan las tuplas que cumplen los demás criterios.
- Si hay índices simples para varios atributos, se utilizan los índices por separado y luego se intersecan los resultados.

Si la selección involucra una disyunción (un or) de condiciones simples, debemos aplicar las mismas por separado y luego unir los resultados.

- Si uno de los atributos no dispone de índice, hay que usar fuerza bruta.

PROYECCION

Dividiremos el análisis de la proyección $\pi_X(R)$ en dos casos:

- X es ***superclave:*** No es necesario eliminar duplicados. $cost(\pi_X(R)) = B(R)$.
- X no es superclave: Debemos eliminar duplicados. Llamaremos $\hat{\pi}_X(R)$ a la proyección de multisets (i.e., sin eliminar duplicados). Podemos:
  - ✓ Ordenar la tabla: si $B(\hat{\pi}_X(R)) \leq M$ podemos ordenar en memoria. De lo contrario, el costo usando un sort externo será:
    M= bloques de memoria que tenemos

$$cost(\pi_X(R)) = cost(Ord_M(R)) = 2 \cdot B(R) \cdot \underbrace{\lceil log_{M-1}(B(R)) \rceil}_{\text{Representa cant. de etapas del sort}} - B(R)$$

Todas las etapas del sort tienen 2B(R) porque leo y escribo, excepto la ultima etapa, que solo escribo, así que le resto B(R). la cantidad de etapas depende de cuantos bloques de memoria tenga.

  - ✓ Utilizar una estructura de hash: Si $B(\hat{\pi}_X(R)) \leq M$ también podemos hacer el hashing en memoria, con costo B(R). Utilizando hashing externo el costo es de: $B(R) + 2 \cdot B(\hat{\pi}_X(R))$.

Obvs: si la consulta SQL no incluye **DISTINCT** entonces el resultado es un *multiset,* y el costo es siempre B(R)

OPERADORES DE CONJUNTOS

El costo es bastante parecido al de la proyección, salvo que acá yo debo ordenar ambos, tanto R como S, grabarlo a disco y luego leerlo ordenado, por lo que el costo es un poquito mayor (le sumo 2B(R) y 2B(S) al costo de ordenamiento

1. Primero ordenamos las tablas R y S. Si alguna no entra en memoria, usamos sort externo.

Asumimos que no se devuelven repetidos (comportamiento default en SQL). Se puede modificar sencillamente en caso de querer repetidos.

2. Procesaremos ambas tablas ordenadas haciendo un merge que avanza por las filas ri y sj ordenadas de cada tabla.

Costo: cost(R [∪| ∩ |−] S) = cost(Ord_M(R))+cost(Ord_M(S)) +2·B(R)+2·B(S)

Para la unión (∪), debemos devolver todas las filas:

1. Si $r_i = s_j$ devolver una de ellas y avanzar sobre ambas tablas hasta que cambien de valor.
2. Sino, devolver la menor y avanzar sobre su tabla hasta que cambie de valor.
3. Cuando alguna tabla se termine, devolver todo lo que queda de la otra, sin duplicados.

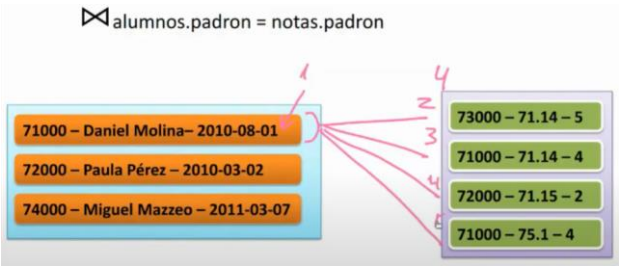
- Para la intersección (∩), devolver las tuplas que están en ambas relaciones:
  1. Si $r_i \neq s_j$ avanzar sobre la tabla de la menor de ellas un lugar, sin devolver nada.
  2. Si $r_i = s_j$ devolver una de ellas y avanzar sobre ambas tablas hasta que cambien de valor.
  3. Cuando alguna tabla se termine, finalizar.

- Para la diferencia (−), devolver las tuplas que están en R pero no es S:
  1. Si $r_i > s_j$ avanzar sobre la tabla S hasta que cambie de valor.
  2. Si $r_i < s_j$ devolver r_i y avanzar sobre la tabla R hasta que cambie de valor.
  3. Si $r_i = s_j$ avanzar sobre ambas tablas hasta que cambien de valor.
  4. Cuando R se termine, finalizar. Cuando S se termine, devolver todo lo que queda de R, sin duplicados.

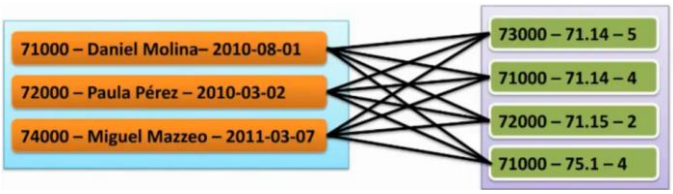
JUNTA

La operación de junta es una de las más frecuentes y demandantes.

En el peor de los casos, voy a tener que, por cada fila de una tabla, compararla con todas las filas de la otra tabla.

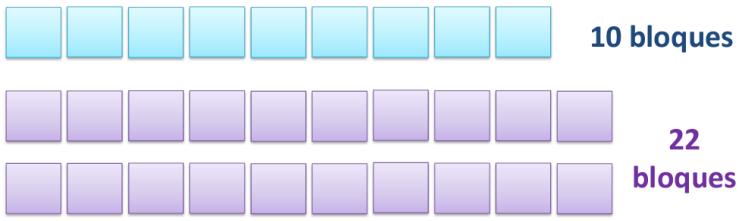


Y lo mismo para todas las filas.



Pero voy a tener distintas situaciones

Que pasa si la celeste es la tabla de alumnos (A), y la violeta la de notas (N), con esa cantidad de bloques por tabla:



- Con memoria suficiente
- Con 2 bloques para lectura y 1 para resultados
- Generalizar a M bloques (1 para resultados)

Si yo puedo almacenar todo en memoria, leo una vez cada tabla y después en memoria comparo la cantidad de veces que quiera. Para eso B(A) + B(N) <= M.

Ahora, que pasa si tengo un bloque, otro bloque para leer las dos tablas, y otro bloque para los resultados. Tengo que empezar que es lo que voy a guardar y que sacar.

Existen distintos métodos para calcular la junta:

- ✓ Método de loops anidados por bloque
- ✓ Método de único loop
- ✓ Método sort-merge
- ✓ Método de junta hash (variante GRACE)

Observación: A continuación, presentaremos los métodos, y sólo indicaremos el costo de lectura de datos y cálculo del resultado. Para calcular el costo de almacenamiento (que no siempre se realiza para las operaciones intermedias) es necesario estimar la cardinalidad del resultado.

1. *Método de loops anidados por bloque*

Dadas dos relaciones R y S, el método de loops anidados por bloque consiste en tomar cada par de bloques de ambas relaciones, y comparar todas sus tuplas entre sí.

Básicamente este algoritmo dice:

- ✓ Levanto un bloque de una de las tablas (la mas chica me conviene) y ese bloque lo comparo con todos los bloques de la segunda
- ✓ Levanto un segundo bloque, y comparo con todos los bloques de la otra tabla.
- ✓ El 3ro contra todos de la 2da.

El “comparo” significa: tengo dos bloques en memoria, me fijo que combinación ed filas cumple la condición de join, y cada una que cumpla, la devuelvo.

Si porcada bloque de R se leen todos los bloques de S, el costo de procesar dicho bloque es: 1+B(S); 1 porque lo tengo que leer, y B(S) porque lo tengo que juntar con el bloque S. El total es de B(R) · (1+B(S)).

Utilizando las tuplas de S como pivotes, el costo total sería de B(S) · (1 + B(R)).

El costo del método es entonces: cost(R ∗ S) = min(B(R)+B(R)·B(S), B(S)+B(R)·B(S))

Esta estimación es un peor caso, cuando solo tengo 3 bloques en memoria disponibles., suponiendo que sólo podemos tener un bloque de cada tabla simultáneamente en memoria (Mi = 2) y un bloque para los resultados.

Si pudiéramos cargar una de las tablas completa en memoria y sobrara un bloque, tendríamos **el mejor caso**: cost(R ∗ S) = B(R)+B(S). levanto toda una tabla en memoria (leo una sola vez R) y la comparo con todos los bloques de la otra.

Si no me entra toda la tabla en memoria, tengo que ver de a cuantos bloques la puedo leer la tabla mas pequeña y la cantidad total / la cantidad de bloques que puedo leer es cuantas veces voy a tener que acceder a S para comparar. Es decir: si me entran 3 bloques de memo, voy a tener que leer S tantas veces como bloques de R/3 tenga.

2. *Método de único loop*

Funciona cuando tenemos un índice que podemos aprovechar de alguna de las tablas, que es con la condición de join, y la condición de join es por igualdad.

Arrancamos leyendo la tabla que NO tiene índice, lo cual tiene un costo de B(R) y para cada fila de esa tabla, hago un index scan con la otra tabla, donde el costo del index scan es el mismo que para la selección. Lo tengo que hacer tantas veces como filas tenga la otra tabla (la tabla sin índice) n(S).

Si el atributo de junta tiene un índice asociado en R, por ejemplo, podemos recorrer las tuplas de S y para cada una de ellas buscar en el índice la/s tupla/s de R en que el atributo coincide.

- ✓ Si el índice es primario:

$$cost(R * S) = B(S) + n(S) \cdot (Height(I(A,R)) + 1)$$

- ✓ Si el índice es de clustering, puede haber más de una coincidencia:

$$cost(R * S) = B(S) + n(S) \cdot \left(Height\big(I(A,R)\big) + \left\lceil \frac{n(R)}{V(A,R)\ .\ F(R)} \right\rceil \right)$$

- ✓ Si el índice es secundario:

$$cost(R * S) = B(S) + n(S) \cdot \left(Height\big(I(A,R)\big) + \left\lceil \frac{n(R)}{V(A,R)} \right\rceil \right)$$

Ejemplo

Estimar el costo de la junta
Clientes(*nroCliente*, *nombre*) ∗ *Ordenes*(*nroCliente*, *nroOrden*),
utilizando un índice de clustering por nroCliente en la tabla Ordenes.

Clientes	Ordenes
n(Clientes) = 5000	n(Ordenes)=10000
F(Clientes) = 20	F(Ordenes)=25
	Height(I(nroCLiente, Ordenes)) = 4
V(nroCliente, Clientes) = 5000	V(nroCliente, Ordenes)=2500

Es un índice de clustering, así que deberemos usar esa formula.

No tenemos B(S) con S = Clientes, pero si tenemos n(S) y F(S) así que podemos hacer n(S)/F(S) = 5000/20.

Luego, sumamos n(S) así que sumamos 5000 clientes. Multiplicamos por la altura de I(A, R), que es 4 (dato) y por ultimo debemos calcular $\left\lfloor \frac{n(R)}{V(A,R)\ .\ F(R)} \right\rfloor$. Vamos a ver cúantas ordenes tiene cada cliente (n(R)) y cuantos bloques ocupan esas órdenes, sabiendo que están ordenadas. Hay 10.000 dividido la variablidad, que es 2500 y la cantidad de ordenes por bloque que es 25.

Esto es $\frac{10.000}{2500\ .\ 25} = 0.16$ y como no podemos acceder a 0.16 bloques, rendondeamos para arriba, nos queda 1. Finalmente, obtenemos:

Respuesta

$cost(Clientes * Ordenes) = \frac{5000}{20} + 5000 \cdot (4 + 1) = 25250$

3. *Método de sort merge*

Consiste en ordenar los archivos de cada tabla por el/los atributo/s de junta. Por ejemplo: si hago la junta de las tablas Alumnos y Notas por *padron,* ordeno ambas tablas por padron. Luego, voy leyendo en orden en memoria, y ahí los datos van a estar "mas cerca". Esto asume de alguna forma de que tengo la memoria necesaria.

- ✓ Si entran en memoria, el ordenamiento puede hacerse con quicksort, y el costo de acceso a disco es sólo **B(R) + B(S).**
- ✓ Si los archivos no caben en memoria debe utilizarse un algoritmo de sort externo. El costo de ordenar R y volverlo a guardar en disco ordenado es de aproximadamente ***2·B(R)·⌈logM−1(B(R))⌉*** → ¡El log estima la cantidad de etapas del sort externo!

Una vez ordenados, se hace un merge de ambos archivos que sólo selecciona aquellos pares de tuplas en que coinciden los atributos de junta. *El merge recorre una única vez cada archivo*, por lo que tiene un costo de B(R)+B(S).

El costo total es entonces: $cost(R*S) = B(R)+B(S)+2\cdot B(R)\cdot\lceil \log_{M-1}(B(R))\rceil+2\cdot B(S)\cdot\lceil \log_{M-1}(B(S))\rceil$

Es el costo de ordenar y grabar a disco para cada tabla y el costo de leer R y leer S.

En general se suele utilizar este método **cuando ya tenemos algún orden previo.** Si no, es bastante costoso.

4. *Método de junta hash (variante Grace)*

¿Que pasa si ninguna de las tablas entra en memoria? Yo puedo intentar dividir las tablas en grupos de tal manera que estos sí entren en memoria.

Ej: tenemos tabla de alumnos con 1,000 bloques y notas con 10,000. En memoria tengo 102 bloques. Si la tabla de alumnos fuera de solo 100 bloques podría levantar todo alumnos, levantar de a un bloque la de notas, y en el otro bloque acumular resultados. Pero alumnos no entra en memo.

¿Qué pasaría si pudiera dividir a la tabla de alumnos en 10 conjuntos de 100 bloques cada uno? Levanto de a conjunto y voy procesando con Notas. Por lo que también querría tener 10 bloques (en este caso de 1000) de notas.

La idea es particionar los datos para que la tabla más chica tenga una cantidad de particiones que entre completamente en memoria, y ahí recién usar el método de loops anidados.

Se van a particionar las tablas R y S en m grupos utilizando una función de hash h(X) aplicada sobre los atributos de junta X, porque necesito que las notas que estén en el conjunto 1 sean las de los alumnos del conjunto 1. Lo que hace esto es que no me queden notas en un conjunto cuando el alumno este en otro conjunto. **Para que pase eso, aplicamos el hash sobre el atributo de la junta,** para que caigan en el mismo conjunto.

Atención: Que dos tuplas r ∈ R y s ∈ S cumplan que h(r.X) = h(s.X) no implica que r.X = s.X!

Costo del particionado: 2 · (B(R)+B(S)), porque es necesario leer todos los bloques y reescribir sus datos en otro orden.

El particionado es básicamente:

- ✓ Leer la tabla R bloque a bloque
- ✓ A cada fila aplicarle la función de hash
- ✓ Ir acumulándola en un buffer de memoria. Cuando se llena lo paso a disco
- ✓ Las particiones van a tener tantos bloques como bloques tenia la tabla
- ✓ Hago lo mismo con la tabla S.
- ✓ Osea tengo una entrada R y una salida R_0, R_1,…, R_{m-1},y una entrada S y una S_0, S_1,…, S_{m-1}

Luego, cada par de grupos Ri y Si se combina verificando si se cumple la condición de junta con un enfoque de fuerza bruta. Cada Ri y Si se combina haciendo el método de loops anidados. Se levanta la mas pequeña completamente en memoria y se procesa contra la mas grande.

Obvs: No es necesario combinar Ri y Sj para i= j poque r.X =s.X →h(r.X) = h(s.X) osea xq seguro las notas de un alumno fueron al mismo conjunto.

Hipótesis: m fue escogido de manera que para cada par de grupos (Ri,Si) al menos uno entre en memoria, y sobre un bloque de memoria para hacer desfilar al otro grupo.

Es decir, dado el m que elija, me tiene que entrar una partición entera de la tabla mas chica + un bloque para levantar de a bloques la otra tabla + un bloque para volcar los resultados. Suponiendo que R es la de menor tamaño… cuánto me ocupa Ri en disco: B(R)/m.

Hay que elegir un m (m=particiones del hash y M=memoria disponible) tal que

- m debe ser <= a M-1 (por la etapa de particionamiento
- m debe ser <= a M-2 (siendo R la relación mas pequeña) por la etapa de loops anidados
- m <= V(A,R), m <= V(A,S) asumiendo A el atributo de junta

Costo de la combinación de R_i y S_i: $B(R_i) + B(S_i)$. Esto se deduce de:

- Observación 1: $F(R_i) = F(R)$ y $F(S_i) = F(S)$
- Observación 2: $\sum_{i=1}^{m} n(R_i) = n(R)$ y $\sum_{i=1}^{m} n(S_i) = n(S)$

El costo total es: $cost(R * S) = 3 \cdot (B(R)+B(S))$

Son 2 (B(R)+B(S)) de la etapa del hash y uno mas de la etapa de loops anidados.

En el caso de que sea una junta de una tabla consigo misma, el costo es 3B(R) (ver explicación de la practica)

Ejemplo

Ejemplo: Obra social

La base de datos de una obra social cuenta con las siguientes tablas:

- Afiliaciones(nro_socio, cod_plan)
- Planes(cod_plan, servicio)

Existe también un índice de clustering por "cod_plan" para Planes. Se requiere hacer la junta natural de ambas tablas.

Afiliaciones	Planes
n(Afiliaciones) = 3000000	n(Planes)=10000
B(Afiliaciones) = 300000	B(Planes)=500
	Height(I(cod_plan, Planes)) = 2
	V(cod_plan, Planes)=40

Indique cuál de los siguientes métodos de junta es más conveniente:

- Junta hash GRACE
- Único loop con índice

Vamos a calcular el costo a ambos métodos. No me dice cuanta memoria hay disponible así que vamos a asumir un M que cumple con los requisitos que enunciamos previamente.

Costo junta hash GRACE: = 3(B(R) + B(S)) = 3(300.000 + 500) = 900.000 aproximadamente

Costo unico loop con indice =

$$cost(R * S) = B(S) + n(S) \cdot Height\big(I(A,R)\big) + \left\lfloor \frac{n(R)}{V(A,R)\ .\ F(R)} \right\rfloor$$

Ya que tengo un índice de clustering. En este caso, la tabla que NO tiene índice (osea la tabla S) es la de Afiliados.

Luego:

$$cost(Planes * Afiliados) = B(Afil) + n(Afil) \cdot Height\big(I(cod\_plan, Planes)\big) + \left\lfloor \frac{n(Planes)}{V(A,Pl)\ .\ F(Pl)} \right\rfloor$$

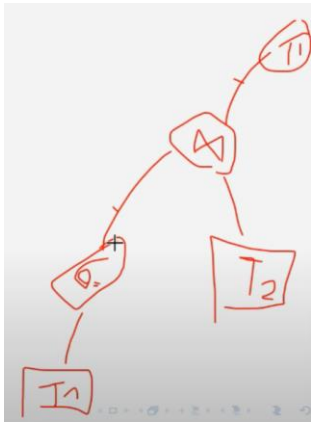
Calculamos el F(planes) con F(R) = n(R)/B(R) = 10.000/500 = 20

$$cost(Planes * Afiliados) = 300.000 + 3.000.000 * (2 + \frac{10.000}{40\ .\ 20}) = 45.000.000$$

Vemos entonces que la mas eficiente es la junta hash Grace

PIPELINING

Acá vamos a empezar a acoplar operadores. El problema es que si yo aplico por ejemplo lo siguiente:



cuando quiero calcular el costo de la selección, no hay problema, porque tengo todos los datos de B, f y n de la tabla 1. Ahora, cuando llego al join, solo tengo los datos de la tabla 2, ya no tengo los de la uno.

Podría guardarme el resultado intermedio a disco y luego guardarlo luego volver a levantarlo cuando hago el join, pero eso no es lo que conviene.

Lo que se suele aplicar, que es ***pipelining*** es que cuando apenas la selección termina de armar un bloque de las filas que cumplen la selección, ese bloque se manda al join y ya está en memoria, no lo tuve que pasar a disco. Básicamente suspendo el 1er operador para que comience a actuar el segundo, y lo mismo hago cuando le pase del join a la proyección.

Entonces ponele que en el join yo estaba usando loops anidados, cuya formula de costo es: $B(R)+B(R)\cdot B(S)$. Y si hago esto del pipelining, como me quedó el bloque en memoria, ese B(R) te lo ahorras porque no tengo que ir a buscar el bloque a disco. Entonces los costos empiezan a disminuir.

Entonces, formalizando:

En muchos casos, el resultado de un operador puede ser procesado por el operador siguiente en forma parcial (es decir sin necesidad de que el operador anterior haya terminado de generar todas las tuplas). Forma parcial = termino un bloque y se lo va pasando por mas que no haya procesado todos los bloques

Esta estrategia se denomina ***pipelining***, y los SGBD suelen utilizarla en los planes de ejecución siempre que sea posible.

Al calcular el costo de dos operadores O anidados $O_2(O_1(R))$ debemos considerar que en caso de utilizar pipelining no será necesario tener todos los bloques de la salida de O_1 para comenzar a calcular O2. En particular, no tendremos que materializar toda la salida de O_1 por falta de espacio en memoria.

Entonces como no tengo que grabar a disco después de cada operador y volver a levantar todo para el próximo operador, me ahorro mucho costo a disco.

1. Si la selección **recibe** bloques en memoria generados por otro operador, no agrega costo extra (se evalúa la condición en memoria): Si recibe de otro operador la selección aporta un **costo 0** extra. Es simplemente un if que se hace en memoria, por lo que no hay cosoto extra de acceso a disco.
2. Si la proyección recibe bloques en memoria generados por otro operador: **se le resta B(R)** al costo de la proyección.

- ✓ Si NO se usa DISTINCT o si se proyecta por una SUPERCLAVE de R, el **costo es B(R)**
- ✓ si SI hay que evitar duplicados, hay que mantener en memoria el resultado
  - sí entra en memo, **el costo es costo 0**.
  - Si NO entra en memoria, **se usa la fórmula de proyección con el método utilizado restándole un B(R)**

3. Si la Junta recibe bloques en memoria generados por otro operador, se usan las fórmulas que ya vimos, pero quitando el costo B(R), porque la primera tabla lo recibe del operador anterior, como vimos en el árbol de ejemplo al principio de esta sección (Sacar todos los B(Tabla) de los métodos de Junta que vimos.

*Por ejemplo, en el caso de **hash Grace es 2 * B(R) + 3 * B(S)*** porque me ahorro 1 lectura

No se puede hacer doble pipeline con el join, osea que venga de las dos ramas del árbol. Es decir, el join puede tener pipeline de una rama del árbol, pero de la otra no.

AL FINAL SUMAMOS LOS COSTOS DE CADA OPERACIÓN QUE SE ANIDO EN EL PIPELINE PARA OBTENER EL COSTO TOTAL DE LA CONSULTA

Lo que cambia es que si me llega algo anidado, cambian los n, y los B por ejemplo si tengo una selección o lo que sea. Para estimar esos nuevos valores de n y B, realizamos una estimación de cardinalidad.

TALLER DE PIPELINING

Vamos a ver como maneja PostgreSQL el tema de índices.

El utilizar el índice de una tabla me reduce bastante el costo a disco de la busqueda

ESTIMACION DE LA CARDINALIDAD

Si bien para cada tabla tenemos los valores como B(R), n(R), f(R), no los tenemos para lo que sale de cada operador, porque eso depende mucho del operador y su parametria (una selección no es lo mismo la cantidad de filas que va a salir si le pongo una condición u otra)

Entonces, lo que buscamos es estimar de cada forma, estimar los resultados que salen de los operadores anteriores.

Como parte de la estimación del costo de una consulta, es necesario a veces estimar el tamaño de las relaciones intermedias (la cardinalidad) antes de calcularlas. Osea, ***cuantas filas y cuantos bloques salen del operador y, ligado a esto, cuál es el factor del operador.***

Se espera que una estimación de cardinalidad cumpla con los siguientes requisitos:

- ✓ Sea precisa.
- ✓ Sea fácil de calcular.
- ✓ No dependa de la forma en que esa relación intermedia se calculó.

Veremos reglas de estimación de la cardinalidad a través de ejemplos para los siguientes operadores:

- ✓ Proyección
- ✓ Selección
- ✓ Junta

Proyección

Lo primero que tengo que ver es si evita duplicados o no (osea si tiene un distinct). En caso de que sí la estimacion depende mucho de la variabilidad del atributo. **En caso de que no tiene distinct, la cantidad de filas que salen es la misma de las que entran. Osea en n(R) se mantiene**

Ahora, lo que hace la proyeccion es que hace que la fila me ocupe mucho menos porque me quedo con menos columnas. Eso puede hacer que el **factor de bloque sea mayor**, osea que entren mas filas por bloque y por lo tanto tenga menos bloques.

Esta es la estimacion que tenemos que hacer. Para ver eso, debo saber cuanto ocupa cada columna.

- Ejemplo: $Persona(DNI, nombre, f\_nacimiento, género)$
  - 40 millones de tuplas
  - El DNI es un entero de 4 bytes
  - El nombre es un string variable de tamaño promedio 15 bytes
  - La fecha de nacimiento es un timestamp de 4 bytes
  - El género es un caracter

  Supongamos que los bloques son de 1024 bytes con un header de 24 bytes.
- La estimación de la cantidad de bloques que ocupa la relación es:

$$B(Persona) = \frac{40 \cdot 10^6 \cdot (4 + 15 + 4 + 1)}{10^3} = 960000$$

- Ahora queremos estimar $B(\pi_{DNI}(Persona))$. La cantidad de tuplas no se modifica, por lo tanto:

$$B(\pi_{DNI}(Persona)) = \frac{40 \cdot 10^6 \cdot 4}{10^3} = 160000$$

10^3 es porque tenemos 1000 bytes disponibles (1042 – los 24 del header).

Si hay 1000 bytes disponibles, y cada registro ocupa 24 bytes, hago el calculo de B como muestra. Luego puedo obtener el f

Ahora con la proyeccion, me quedo solo con la columna de la proyección y solo uso ese tamaño como tamaño total de la fila de la proyección.

En este caso la cantidad de filas no disminuyo, pero lo que bajo que la cantidad de bloques B.

Nos pueden dar los datos así o decir que el DNI ocupa un 10% de la fila

Selección

La selección reduce el número de tuplas en el resultado, aunque mantiene el tamaño de cada tupla. **siempre mantiene el factor de bloque, pero sí me puede cambiar el n(R), con lo cual también puede cambiar el B de la selección.**

Para estimar el tamaño (el n(R)) de una selección de la forma $\sigma_{Ai=c}(R)$, utilizaremos la variabilidad de Ai en R (V(Ai,R)), que es la cantidad de valores distintos que puede tomar el atributo Ai en dicha relación.

Puedo pensar que el atributo se distribuye equitativamente y decir, por ejemplo: si un atributo tiene 10 valores distintos, 1/10 de la tabla va a tener un valor, otro 1/10 otro y así…

Esto solo aplica para igualdad.

Si fuera por rango nos van a decir: buscamos un rango de fecha y sabemos que el 30% de los alumnos ingresaron en 2010, y la búsqueda fuera por un rango de fechas del 2010, me va a quedar un 10% del n de los atributos.

Realizaremos la siguiente estimación:

$$n(\sigma_{A_i=c}(R)) = \frac{n(R)}{V(A_i, R)}$$

La fracción $\frac{1}{V(A_i,R)}$ se denomina selectividad de A_i en R.

- Ejemplo: $Persona(DNI, nombre, f\_nacimiento, gnero)$.
- Para estimar $n(\sigma_{genero='F'}(Persona))$, consideremos que hay dos géneros posibles. Luego:

$$n(\sigma_{genero='F'}(Persona)) = \frac{n(Persona)}{V(genero, Persona)} = \frac{40 \cdot 10^6}{2} = 20 \cdot 10^6$$

$$B(\sigma_{genero='F'}(Persona)) = \frac{n(\sigma_{genero='F'}(Persona)) \cdot (4 + 15 + 4 + 1)}{10^3} = 480000$$

Dificultades:

- ✓ No nos permite estimar selecciones con otros operadores (≤,≥,=).
- ✓ La estimación asume que el valor c se toma al azar. Si no es así, entonces es sesgada.

El problema que tenemos con esta estimación es que asumimos que es equitativa, y eso no siempre es representativo. Un método más avanzado consiste en utilizar un **histograma para la distribución de Ai**, y en general se guardan los N valores mas frecuentes. Esto nos sirve para estimar mucho mejor la cardinalidad.

Selección – Estimacion con histograma

El histograma nos resume la distribución de los valores que toma un atributo en una instancia de relación dada.

Es útil cuando un atributo toma valores discretos.

- Ejemplo: Película(id, nombre, género)
  - n(Película) = 728
  - V(género, Película) = 9

	drama	comedia	suspenso	otros
Película.género	150	140	128	310

- El histograma nos dice que $n(\sigma_{genero='comedia'}(Peliculas)) = 140$
- ¿Podemos estimar mejor $n(\sigma_{genero='terror'}(Peliculas))$ utilizando el histograma?

$$n(\sigma_{genero='terror'}(Peliculas)) = \frac{n(Pelicula) - (418)}{V(genero, Pelicula) - 3} = \frac{310}{6} = 52$$

Con 418 = la estimación de todos los otros géneros que no son terror. 728 – (150 + 140 + 128). Ahora… esos 310 son de *otros,* no de terror. Para estimar terror divido por la variabilidad y le resto los géneros que ya considere, que en este caso son 3 (drama, comedia y suspenso)

El histograma también lo uso en el costo del index scan. Uso los valores del histograma o usando las formula que vimos recién.

Junta

Cuando la condición de join es la clave primaria en una tabla y clave foránea en la otra, la cardinalidad es la de la tabla donde estaba la foránea.

Consideremos la junta de R(A,B) y S(B,C).

En principio, 0 ≤ n(R ∗S) ≤ n(R)·n(S), dependiendo de como estén distribuídos los valores de B en una y otra relación.

Dadas las variabilidades V(B,R) y V(B,S), asumiremos que los valores de B en la relación con menor variabilidad están incluídos dentro de los valores de B en la otra relación. En el caso en que el atributo de junta es clave primaria en una relación y clave foránea en la otra, esta hipótesis se cumple.

Supongamos que V(R,B) ≥ V(S,B) y tomemos una tupla en $t_R \in R$ y una tupla en $t_S \in S$. Sabemos que t_S.B está incluído dentro de los valores que toma B en R. Luego:

$$cost(t_s.B = t_R.B) = \left\lfloor \frac{1}{V(R,B)} \right\rfloor$$

De manera análoga, si V(R,B) ≤ V(S,B) entonces que tR.B está incluído dentro de los valores que toma B en S. Luego:

$$P(t_R.B = t_S.B) = \left\lfloor \frac{1}{V(S,B)} \right\rfloor$$

Estimacion de la cardinalidad (n(R*S))

En general, $P(t_R.B = t_S.B) = \frac{1}{max(V(R,B),V(S,B))}$, que es la selectividad de la junta (*js*). Luego:

$$n(R * S) = js \cdot n(R) \cdot n(S) = \frac{n(R) \cdot n(S)}{max(V(R,B), V(S,B))}$$

Ejemplo

Estimar la cardinalidad de $R(A,B) * S(B,C) * T(C,D)$, siendo:

R(A,B)	S(B,C)	T(C,D)
n(R) = 1000	n(S)=2000	n(T)=5000
V(R,B) = 20	V(S,B) = 50	
	V(S,C)=100	V(T,C)=500

Respuesta

$n(R * S * T) = 400000$

Se hacen de a una las juntas

1. Saco la cardinalidad n1 de R(A,B) * S(B, C) por R.b = S.b
   $$n1 = \frac{n(R).n(S)}{\max(V(R,B),V(S,B))} = \frac{1000\ .\ 2000}{50} = 40.000$$
2. El n final uso el n1 con el n de T, y la mayor variabilidad de C, que es el atributo en común por el que puedo hacer la junta acá.
   $$n = \frac{n1\ .n(T)}{\max(V(S,C),V(T,C))} = \frac{40000\ .\ 5000}{500} = 400.000$$

Para estimar el factor de bloque del resultado, asumiremos que si una tupla de R ocupa $\left\lfloor \frac{1}{F(R)} \right\rfloor$ bloques y una tupla de S ocupa $\left\lfloor \frac{1}{F(S)} \right\rfloor$ bloques, entonces una tupla del resultado ocupa menos de: $\left\lfloor \frac{1}{F(R)} \right\rfloor + \left\lfloor \frac{1}{F(S)} \right\rfloor$ y por lo tanto el factor de bloque es al menos:

Estimacion del factor de bloque

$$F(R * S) = \left(\frac{1}{F(R)} + \frac{1}{F(S)} \right)^{-1}$$

La fórmula subestima el factor de bloque, porque no tiene en cuenta que los atributos de junta se repiten en ambas tablas

Estimacion de cantidad de bloqus

La cantidad de bloques será (sobreestimación):

$$B(R * S) = \frac{js \cdot n(R) \cdot n(S)}{F(R * S)} = js \cdot B(R) \cdot B(S) \cdot (F(R) + F(S))$$

Con js siendo la selectividad de la junta

Junta – estimación con histograma

Brinda una mejor estimacion de la junta, especialmente cuando la distribucion NO es equitativa.

- Ejemplo:
  - $R(A, B)$, con $V(B, R) = 18$
  - $S(B, C)$, con $V(B, S) = 15$
- Supongamos que disponemos de un histograma que nos muestra los k valores más frecuentes de B en cada una de las relaciones. En este caso, $k = 5$.

	4	12	14	20	22	30	otros
R.B	200		320	120	150	65	550
S.B	150	100		180	210	85	410

- Para cada valor x_i del que conocemos $f_R(x_i)$ [1] y $f_S(x_i)$, sabemos que la cantidad de tuplas en el resultado será: $f_R(x_i) \cdot f_S(x_i)$.

	4	12	14	20	22	30	otros
R.B	200		320	120	150	65	550
S.B	150	100		180	210	85	410
$R * S$	30000			21600	31500	5525	

[1] $f_R(x_i) = n(\sigma_{B=x_i}(R))$.

Osea el 30.000 sale de hacer 200 . 150

1. Cuando ambas tienen valores, hacemos la multiplicación
2. Cuando conocemos uno solo de los valores, ***estimamos la que esta vacia.***
   Para el de 12, hacemos 550/18 – 5 con 550 siendo el valor de otros.
   Cuando usamos ese 550, le vamos a después restar al ***otros*** el valor que otuve de la cuenta de recién que estime.

- Para aquellos x_i de los que sólo conocemos $f_R(x_i)$ ó $f_S(x_i)$, estimaremos el faltante a partir de la columna "otros" y de la variabilidad.
- Por ejemplo, si conocemos sólo $f_R(x_i)$, entonces:

$$f_S(x_i) = \frac{f_S(otros)}{V(B,S) - k}$$

	4	12	14	20	22	30	otros
R.B	200	43	320	120	150	65	550 507
S.B	150	100	41	180	210	85	410 369
$R * S$	30000	4300	13120	21600	31500	5525	

- Actualizamos también las frecuencias de "otros", y el valor de k, que se convierte en $k' = 6$.

- Finalmente estimamos las tuplas correspondientes a "otros" en el resultado utilizando la estimación simple (equiprobable):

$$f_{R*S}(otros) = \frac{f_R(otros) \cdot f_S(otros)}{max(V(R,B) - k', V(S,B) - k')}$$

	4	12	14	20	22	30	otros
R.B	200	43	320	120	150	65	550 507
S.B	150	100	41	180	210	85	410 369
$R * S$	30000	4300	13120	21600	31500	5525	15590

- La estimación final es:

$$n(R * S) = \sum_i f_{R*S}(x_i) = 121635$$

- La simple estimación (sin histograma) nos hubiera dado como resultado $n(R * S) = 88594$ (verificar).

Osea la estimación final es la suma de la fila de debajo de todo que esta en azul.