Ayudantía 8

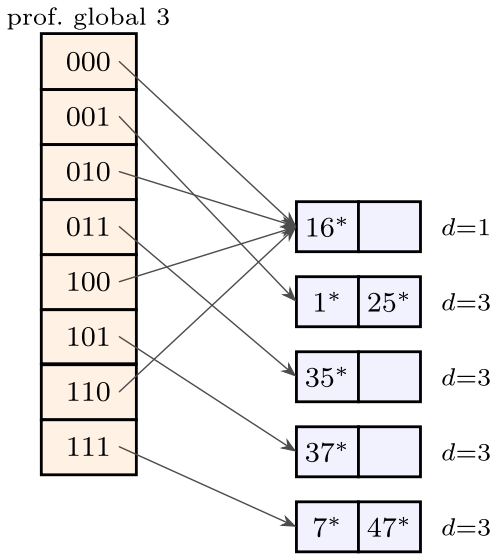
Preguntas 1, 3 y 5 · Hashing, índices y reescritura de consultas · IIC2413

Pregunta 1

La historia de un extendable hashing

Enunciado

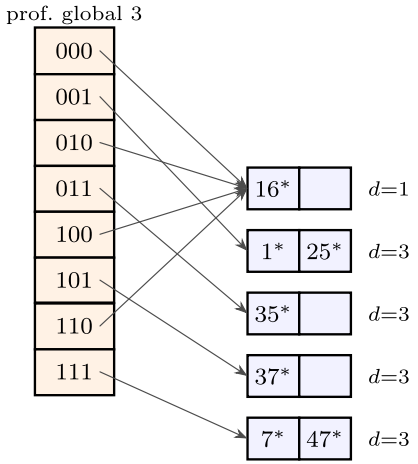
Extendable hashing con $B = 2$ que recibió **solo inserciones** de $S = \{1, 7, 16, 25, 35, 37, 47\}$ en un orden **desconocido**. Estado final:



(a) ¿Cuántos splits y doblamientos hubo? ¿Por qué no dependen del orden? · (b) Un orden donde **una** inserción provoque **todos** los doblamientos · (c) Un orden sin inserciones con más de un split · (d) ¿Existe un dato nuevo que provoque: (i) ningún split, (ii) split sin doblar, (iii) split con doblar, (iv) dos splits?

(a) Pregunta

> Miren el estado final. ¿Qué cambia en el índice **cada vez que hay un split**? ¿Y **cada vez que hay un doblamiento**?



(a) Respuesta

Evento	Qué cambia	Contador
el dato cabe	el contenido de un bucket	ninguno
split	un bucket pasa a ser dos: **buckets** $+1$	splits
doblamiento	profundidad global $n + 1$	doblamientos

> Con solo inserciones, los contadores **solo crecen**: el total lo fija el estado final.

(a) Aplicamos

Cada **split** suma exactamente **1 bucket**; cada **doblamiento** suma exactamente **1** a n. Nada más cambia esas cantidades, y con solo inserciones **nunca bajan**.

	Inicio	Final	Eventos
buckets	1	5	$5 - 1 = \mathbf{4}$ splits
profundidad global n	0	3	$3 - 0 = \mathbf{3}$ doblamientos

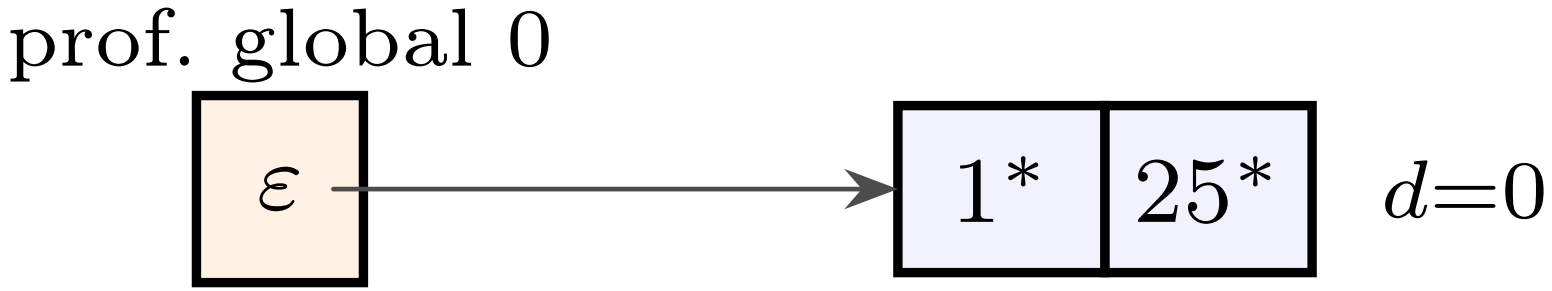
> **¿Por qué no dependen del orden?** Como cada evento suma exactamente 1, el **número** de eventos es la diferencia entre el final y el inicio. El orden solo cambia **cuándo** ocurren y **cuántos** caen en una misma inserción: de eso tratan (b) y (c).

(b) Pregunta

> Hay 3 doblamientos en total y queremos que **una sola inserción** los provoque **todos**. Justo **antes** de esa inserción, ¿cómo tiene que estar el índice? ¿Cuánto vale n y cuántos datos ya entraron?

(b) Respuesta

- n sube **de a uno**: si una inserción hace los 3, justo antes $n = 0$.
- Con $n = 0$ hay **un solo bucket**. Para que algo ocurra, debe estar **lleno**: ya entraron exactamente **2 datos** ($B = 2$).
- El **tercer** dato dispara todo.



Por ejemplo, tras insertar 1 y 25.

(b) Pregunta

> El bucket $\{a, b\}$ está lleno, con $n = d = 0$, y llega c. Hay que doblar y dividir por el **bit 1**. ¿Cuándo hay que **repetir**? ¿Cuántos doblamientos acaba provocando c?

(b) Respuesta: la cascada

- Si a, b y c tienen el **mismo bit 1**, tras dividir **siguen juntos**: el bucket sigue lleno y hay que repetir con el bit 2 (y doblar de nuevo si $d = n$).
- Seguimos dividiendo hasta el **primer bit en que los tres datos no coinciden**. Ahí se reparten en los **dos buckets nuevos**: uno con 1 dato y el otro con 2. Como $B = 2$, **ambos caben** y c queda guardada.
- Con $n = d = 0$, cada split necesita además un doblamiento. Entonces, para obtener k **doblamientos** buscamos tres datos cuyo **primer bit distinto sea el bit k**.

Vale para $B = 2$ y un dato que llega a un bucket lleno.

Y si cambiamos la función de hash, ¿se sigue cumpliendo?

(b) Aplicamos

Queremos 3 doblamientos: buscamos tres datos cuyo **primer bit distinto sea el bit 3**:

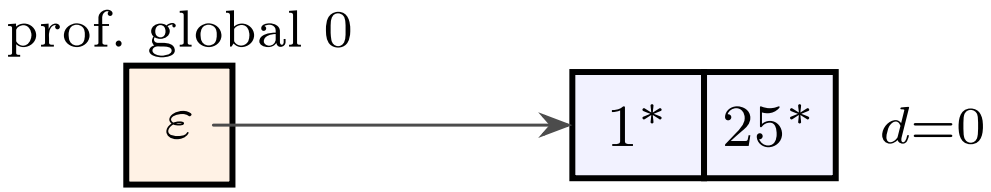
> los tres datos **coinciden en los bits 1 y 2** y **difieren en el bit 3**.

Dato	Binario	Bit 1	Bit 2	Bit 3	Últimos 2 bits
1	`1`	1	0	0	`01`
25	`11001`	1	0	0	`01`
37	`100101`	1	0	1	`01`
7	`111`	1	1	1	`11`
35	`100011`	1	1	0	`11`
47	`101111`	1	1	1	`11`
16	`10000`	0	0	0	`00`

Terminan igual en dos bits: $\{1, 25, 37\}$ y $\{7, 35, 47\}$. En ambos el bit 3 **no** es igual para los tres.

Elegimos $1, 25, 37$ y luego el resto.

(b) Estado antes de la inserción clave

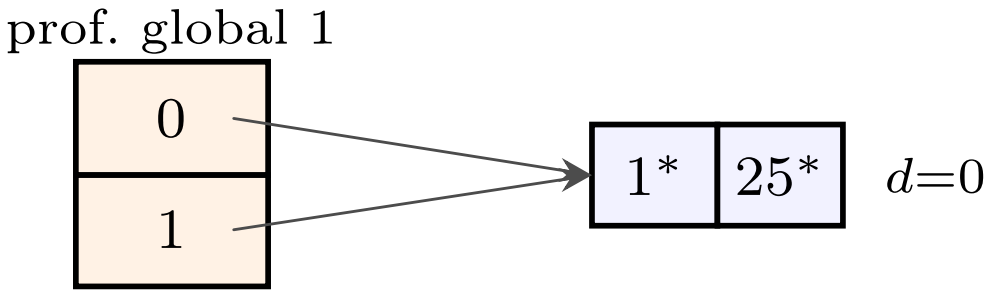


Se insertaron 1 y 25. Llega el **37** $= 100101$. Con $n = 0$, $h_0(k^*) = \varepsilon$ para todas: hay una sola entrada, que lleva al bucket $\{1, 25\}$, **lleno**. El 37 **no cabe**.

Repetimos *(doblar si $d = n$, split, ¿cabe?)*. Confirmamos paso a paso.

Dato	Binario	$h_0(k^*)$	Bucket al que lleva
1	`1`	ε	{1, 25}, $d = 0$
25	`11001`	ε	{1, 25}, $d = 0$
37	`100101`	ε	{1, 25}, $d = 0$

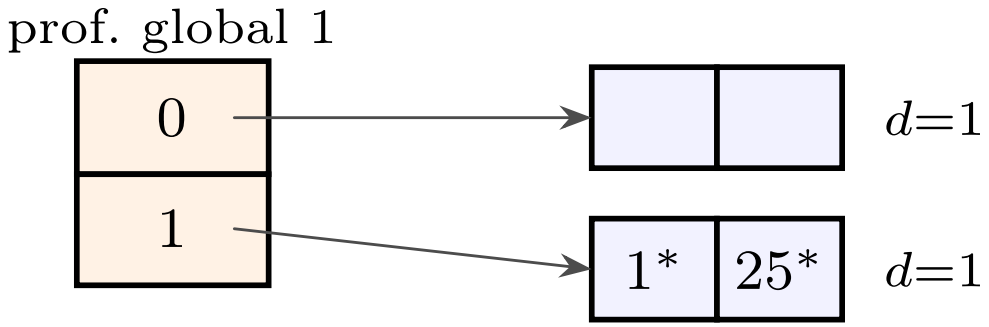
(b) Confirmamos · paso 1 · $d = n = 0$: se dobla ($n = 1$)



$d = n = 0 \Rightarrow$ **no se puede dividir sin más directorio**. Se dobla: $n = 1$. Ahora evaluamos $h_1(k^*)$, el **último bit**: los tres terminan en `1`.

Dato	Binario	$h_1(k^*)$	Bucket al que lleva
1	`1`	`1`	{1, 25}, $d = 0$
25	`11001`	`1`	{1, 25}, $d = 0$
37	`100101`	`1`	{1, 25}, $d = 0$

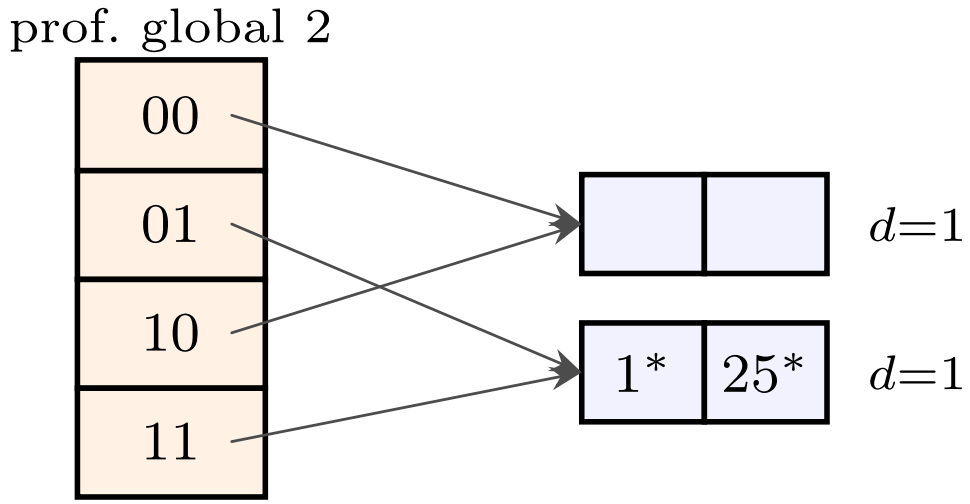
(b) Confirmamos · paso 2 · split por el bit 1



Se separa por el bit 1: 1 y 25 tienen $h_1 = 1 \Rightarrow$ **quedan juntos**; el bucket de `0` queda vacío. El 37 también tiene $h_1 = 1$: llega al bucket lleno y **sigue sin caber**.

Dato	Binario	$h_1(k^*)$	Bucket al que lleva
1	`1`	`1`	{1, 25}, $d = 1$
25	`11001`	`1`	{1, 25}, $d = 1$
37	`100101`	`1`	{1, 25}, $d = 1$

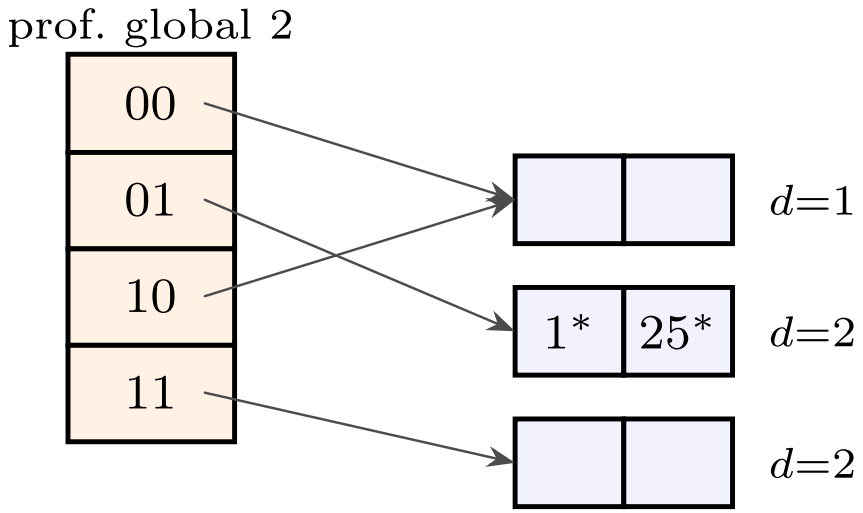
(b) Confirmamos · paso 3 · el 37 sigue sin caber: se dobla ($n = 2$)



El bucket $\{1, 25\}$ tiene $d = 1 = n \Rightarrow$ se dobla de nuevo: $n = 2$. Evaluamos $h_2(k^*)$, los **dos últimos bits**: los tres valen `01`.

Dato	Binario	$h_2(k^*)$	Bucket al que lleva
1	`1`	`01`	{1, 25}, $d = 1$
25	`11001`	`01`	{1, 25}, $d = 1$
37	`100101`	`01`	{1, 25}, $d = 1$

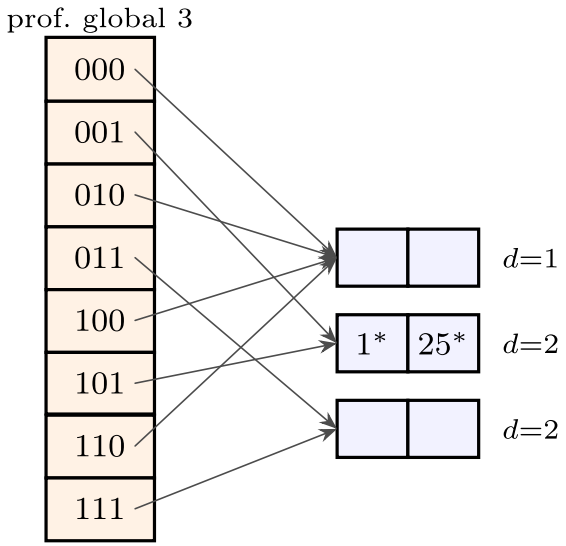
(b) Confirmamos · paso 4 · split por el bit 2



Se separa por el bit 2: 1, 25 y 37 tienen bit 2 $= 0$ (se ve en $h_2 = 01$) ⇒ **siguen juntos** y el 37 todavía no cabe.

Dato	Binario	$h_2(k^*)$	Bucket al que lleva
1	`1`	`01`	{1, 25}, $d = 2$
25	`11001`	`01`	{1, 25}, $d = 2$
37	`100101`	`01`	{1, 25}, $d = 2$

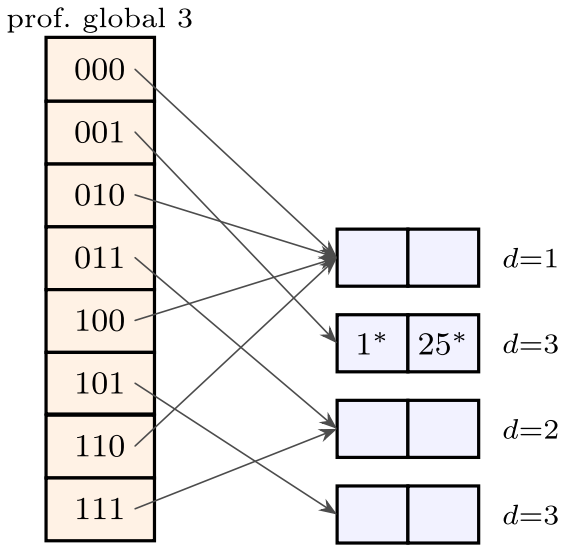
(b) Confirmamos · paso 5 · de nuevo $d = n$: se dobla ($n = 3$)



$d = n = 2 \Rightarrow$ tercer doblamiento: $n = 3$. Evaluamos $h_3(k^*)$, los **tres últimos bits**: el 1 y el 25 valen `001`; el 37 vale `101`. **Ya difieren.**

Dato	Binario	$h_3(k^*)$	Bucket al que lleva
1	`1`	`001`	{1, 25}, $d = 2$
25	`11001`	`001`	{1, 25}, $d = 2$
37	`100101`	`101`	{1, 25}, $d = 2$

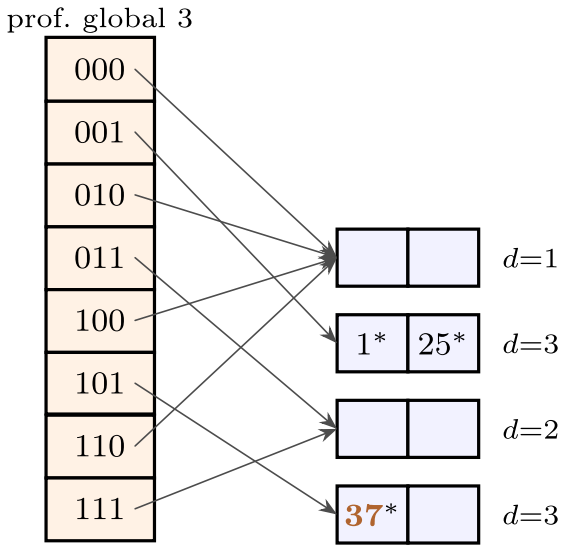
(b) Confirmamos · paso 6 · split por el bit 3



Se separa por el bit 3: 1 y 25 tienen bit 3 $= 0$ (entrada `001`), juntos; la entrada `101` lleva a un bucket **vacío**.

Dato	Binario	$h_3(k^*)$	Bucket al que lleva
1	`1`	`001`	{1, 25}, $d = 3$
25	`11001`	`001`	{1, 25}, $d = 3$
37	`100101`	`101`	{ }, $d = 3$

(b) Confirmamos · paso 7 · el 37 cabe



$h_3(37^*) = 101 \Rightarrow$ cae en el bucket vacío y cabe.

> En **una** inserción: **3 splits y 3 doblamientos**. Es el máximo posible.

Dato	Binario	$h_3(k^*)$	Bucket al que lleva
1	`1`	`001`	{1, 25}, $d = 3$
25	`11001`	`001`	{1, 25}, $d = 3$
37	`100101`	`101`	{37}, $d = 3$

(b) Con ese orden, la historia completa

Orden $1, 25, 37, 7, 16, 35, 47$:

Dato	Binario	n	Entrada	Bucket al que llega	¿Cabe?	Splits	Dobl.
1	`1`	0	ε	{ }, d=0	sí	0	0
25	`11001`	0	ε	{1}, d=0	sí	0	0
37	`100101`	0	ε	{1, 25}, d=0	**no**	3	3
7	`111`	3	`111`	{ }, d=2	sí	0	0
16	`10000`	3	`000`	{ }, d=1	sí	0	0
35	`100011`	3	`011`	{7}, d=2	sí	0	0
47	`101111`	3	`111`	{7, 35}, d=2	**no**	1	0
					total	**4**	**3**

Los totales siguen siendo 4 y 3, como en (a).

(c) Pregunta

> Ahora queremos un orden donde **ninguna** inserción provoque más de un split. Con lo que descubrimos: ¿cuándo una inserción provoca 2 o más? ¿Qué datos de S podrían hacerlo?

(c) Respuesta

1. **Cascada.** Es dividir por bits en los que los tres datos coinciden: cada uno de esos splits no separa nada y hay que repetir (son 2 o más splits en una inserción). Queremos evitarla.
2. **Quién puede causarla.** Solo datos que comparten bits. En S son los grupos $\{1, 25, 37\}$ y $\{7, 35, 47\}$: comparten los bits 1 y 2, y difieren recién en el 3.
3. **De qué depende si la causan.** Un bucket con profundidad d ya "usó" los últimos d bits: todos sus datos coinciden en ellos, así que no pueden separar a nadie. Si hay que dividirlo, se parte desde el bit $d + 1$. El tercero que llega comparte esos d bits (si no, no habría llegado a ese bucket). Con el grupo $\{1, 25, 37\}$:

d del bucket	Bits que ya usó	Qué hay que dividir	Splits
0	ninguno	bit 1 (siguen juntos), bit 2 (siguen juntos), bit 3 (se separan)	3
1	el bit 1	bit 2 (siguen juntos), bit 3 (se separan)	2
2	los bits 1 y 2	bit 3 (se separan)	1

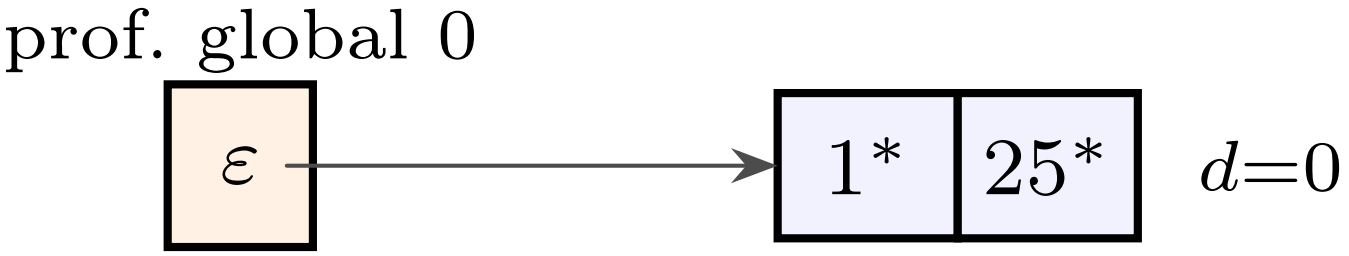
Los splits inútiles son los que caen en bits compartidos que el bucket no usó: $2 - d$. Más el que separa (bit 3): $(2 - d) + 1 = \mathbf{3 - d}$ splits.

(c) Respuesta

> **Conclusión.** Hay cascada cuando $3 - d \geq 2$, es decir, $d \leq 1$. **Se evita si, cuando llega el tercero de un grupo, el bucket que tiene a los otros dos ya tiene $d \geq 2$**: ya usó los dos bits compartidos y el siguiente bit los separa de una vez. Con $d > 2$ el tercero ya no llega a ese bucket.

(c) Pregunta

> Partimos como en (b): 1 y 25 ya están en el bucket ($n = d = 0$). ¿Cómo logramos que ese bucket ya tenga $d \geq 2$ **cuando llegue el 37**, sin provocar una cascada por el camino? ¿Qué datos podemos insertar antes?



(c) Respuesta: probamos cada dato

Estado: $\{1, 25\}$ lleno, $n = d = 0$. El bit $d + 1 = 1$ de 1 y 25 es `1`.

Dato	Binario	Bit $d + 1$ del dato	Splits	d de $\{1, 25\}$ después	¿Sirve?
7	`111`	`1` (coincide)	2	2	no: cascada
16	`10000`	`0` (difiere)	1	1	**sí**: 1 split, pero falta subir d
35	`100011`	`1` (coincide)	2	2	no: cascada
47	`101111`	`1` (coincide)	2	2	no: cascada
37	`100101`	`1` (coincide)	3	3	no: es el dato que queremos al final

> **Prueba para saber si un dato sirve:** mira el bit $d + 1$ del dato. Si **difiere** del de 1 y 25, se separa de inmediato: **1 split** y el bucket de $\{1, 25\}$ sube a $d + 1$. Si **coincide**, siguen juntos: cascada.

(c) Respuesta: segunda ronda

Tras insertar el 16: $\{1, 25\}$ con $d = 1$, $n = 1$. Ahora el bit $d + 1 = 2$ de 1 y 25 es `0`.

Dato	Binario	Bit $d + 1$ del dato	Splits	d de $\{1, 25\}$ después	¿Sirve?
7	`111`	`1` (difiere)	1	2	**sí**: 1 split y $d \geq 2$
35	`100011`	`1` (difiere)	1	2	**sí**: 1 split y $d \geq 2$
47	`101111`	`1` (difiere)	1	2	**sí**: 1 split y $d \geq 2$
37	`100101`	`0` (coincide)	2	3	no: es el dato que queremos al final

> Cualquiera de 7, 35 o 47 nos lleva a $d = 2$ con **un solo split**. El 37 coincide en el bit 2 (`0`): provocaría cascada, así que **espera** hasta que el bucket ya tenga $d = 2$.

(c) Aplicamos: un orden que evita la cascada

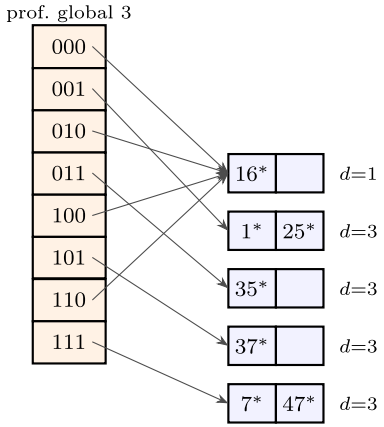
$1, 25, 16, 7, 37, 35, 47$: primero 16 (abre el bit 1), luego 7 (abre el bit 2), y recién ahí el 37. Cada vez que llega un dato a un bucket lleno, los tres datos **se separan en el bit siguiente**, el $d + 1$:

Llega	Bucket al que llega	d	n	Primer bit distinto	Splits	Dobl.
1	{ }	0	0	—	0	0
25	{1}	0	0	—	0	0
16	{1, 25} (lleno)	0	0	bit 1	1	1
7	{1, 25} (lleno)	1	1	bit 2	1	1
37	{1, 25} (lleno)	2	2	bit 3	1	1
35	{7}	2	3	—	0	0
47	{7, 35} (lleno)	2	3	bit 3	1	0

Ninguna fila pasa de 1 split. Totales 4 y 3, como en (a). El orden creciente $1, 7, 16, 25, 35, 37, 47$ también sirve; hay $1\,944$ órdenes válidos de $5\,040$.

(d) Pregunta

> Sobre el estado final: para que un dato nuevo provoque (i) ningún split, (ii) un split sin doblar, (iii) un split con doblar, (iv) dos splits, ¿qué tiene que cumplir **el bucket al que llega**?



(d) Respuesta

Efecto	Requisito sobre el bucket de destino
(i) ningún split	bucket **con espacio**
(ii) split **sin** doblar	bucket **lleno** con $d < n$
(iii) split **con** doblar	bucket **lleno** con $d = n$
(iv) dos splits	bucket lleno cuyos datos, junto con el nuevo, **coinciden también en el bit siguiente**: siguen juntos al dividir

(d) Aplicamos al estado final

Bucket	d	Entradas	Estado	¿Lleno?
$\{16\}$	1	`000 010 100 110`	$d < n$	no
$\{1, 25\}$	3	`001`	$d = n$	**sí**
$\{35\}$	3	`011`	$d = n$	no
$\{37\}$	3	`101`	$d = n$	no
$\{7, 47\}$	3	`111`	$d = n$	**sí**

> **(i) existe:** sirve cualquier $k \notin S$ que llegue a un bucket con espacio, es decir, que **no termine en** `001` **ni en** `111`. Por ejemplo, 3 llega a $\{35\}$; cualquier par llega a $\{16\}$.

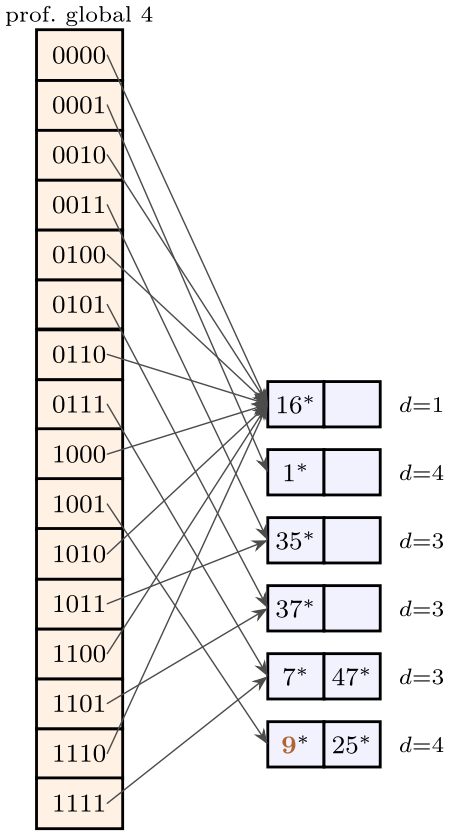
(d)(iii) Split con doblamiento: existe

Un dato provoca un split **con** doblamiento si llega a un bucket **lleno** con $d = n$:

Bucket	Datos que terminan en	¿Lleno?	d vs n	¿Split + doblar?
$\{16\}$	`0` (pares)	no	$d < n$	no: cabe
$\{35\}$	`011`	no	$d = n$	no: cabe
$\{37\}$	`101`	no	$d = n$	no: cabe
$\{1, 25\}$	`001`	**sí**	$d = n$	**sí**
$\{7, 47\}$	`111`	**sí**	$d = n$	**sí**

Ejemplo: el $9 = 1001$ termina en `001` y llega a $\{1, 25\}$ (lleno, $d = n = 3$).

1. se dobla: $n = 4$
2. split por el bit 4: $1 \to 0$, $25 \to 1$
3. el 9 (bit $4 = 1$) queda con el 25



(d)(iv) Dos splits: no existe

Los únicos buckets llenos se separan con **un** split (difieren en el bit 4):

Bucket	Dato	Bit 4	Resultado
$\{1, 25\}$	$1 = 0001$	`0`	separados
	$25 = 11001$	`1`	
$\{7, 47\}$	$7 = 0111$	`0`	separados
	$47 = 101111$	`1`	

El dato nuevo cae en un bucket con a lo más un dato: **cabe**.

(d)(ii) Propuesto

> Para (ii) necesitamos un bucket **lleno** con $d < n$. El único con $d < n$ es $\{16\}$ y **no está lleno**. ¿Qué hacemos? Si preparamos el estado con un dato, ¿cuál conviene elegir para que el siguiente sí lo divida **sin cascada**?

Pregunta 3

Diseño de índices para una carga de trabajo

Enunciado

$\mathbf{Pedidos}(id, cliente, fecha, monto, estado)$: $2\,000\,000$ de tuplas, 50 por página $\Rightarrow$ $P = 40\,000$ páginas. A lo más **3 índices**, a lo más **1 clustered**.

Método	Costo
Full scan	$P = 40\,000$
Hash (igualdad)	$\lceil t/200 \rceil + t$
B+tree unclustered	$2 + \lceil t/200 \rceil + t$
B+tree clustered	$2 + \lceil t/50 \rceil$

	Consulta	t	Frec./h
Q1	`id = …`	1	20 000
Q2	`cliente = …`	20	5 000
Q3	`fecha BETWEEN …`	50 000	50
Q4	`estado = …`	40 000	20

El optimizador toma siempre el método más barato, incluido el full scan. **Hash:** los buckets guardan entradas (llave, puntero), 200 por página, solo de la llave buscada, y **el directorio se asume en memoria** (su lectura no cuenta). Costo total por hora $= \sum_i f_i \cdot \text{costo}(Q_i)$.

Enunciado: qué se pide

> **(a)** Calcule, para cada consulta, el costo con un índice sobre su atributo en cada una de las tres variantes (hash, B+tree unclustered, B+tree clustered) y con full scan.
>
> **(b)** Elija los índices (a lo más 3, a lo más uno clustered) que minimizan el costo total por hora. Dé ese costo y diga qué método usa cada consulta.
>
> **(c)** Justifique por qué el índice clustered conviene sobre `fecha` y no sobre `cliente`, `estado` o `id`.
>
> **(d)** Si solo se permitiera **un** índice, ¿cuál sería? Y si Q_4 devolviera solo 200 tuplas, ¿valdría la pena un índice sobre `estado`?

De dónde salen el 200 y el 50

Cada divisor es la **capacidad de una página de hoja**, y las hojas de los dos índices guardan cosas distintas:

	Qué guarda la hoja	Cabe por página	Costo
Clustered	las **tuplas completas** (las hojas *son* las páginas de datos)	50	$2 + \lceil t/50 \rceil$: las tuplas del rango están **contiguas**
Hash	*data entries* (llave + puntero) en el bucket	200	$\lceil t/200 \rceil + t$: bucket (si $t \leq 200$, 1 página) y **luego** una lectura por tupla
Unclustered	*data entries*: llave + puntero (mucho más chicas)	200	$2 + \lceil t/200 \rceil + t$: hojas, y **luego** una lectura por tupla (están desparramadas)

> El peso del unclustered está en el término t; el $\lceil t/200 \rceil$ casi no cuenta. Por eso deja de convenir cuando $t \approx P$.

El 200 y el 50 son datos del enunciado; lo general es que una entrada de índice pesa menos que una tupla.

Cómo se resuelve (el plan)

1. **Costo de cada consulta con cada índice** (reemplazar t) y comparar con el full scan.
2. **Ahorro por hora** de cada índice posible: $f \cdot (40\,000 - \text{costo con índice})$.
3. **Hash y unclustered no compiten** por el clustered; el clustered va donde más ahorra.
4. **Respetar** el tope (3 índices, 1 clustered) y **sumar** el total para comprobar.

> Un índice solo sirve si **mejora** el full scan. Con $t \approx P$ ningún unclustered ayuda.

(a) Pregunta

> Calcule, para cada consulta, el costo con un índice sobre su atributo en cada una de las tres variantes (hash, B+tree unclustered, B+tree clustered) y con full scan.

(a) Tabla de costos: reemplazar t

	Full scan	Hash $\lceil t/200 \rceil + t$	Unclustered $2 + \lceil t/200 \rceil + t$	Clustered $2 + \lceil t/50 \rceil$
Q1 ($t = 1$)	40 000	$1 + 1 = 2$	$2 + 1 + 1 = 4$	$2 + 1 = 3$
Q2 ($t = 20$)	40 000	$1 + 20 = 21$	$2 + 1 + 20 = 23$	$2 + 1 = 3$
Q3 ($t = 50\ \ 000$)	40 000	no aplica (rango)	$2 + 250 + 50\ \ 000 = 50\ \ 252$	$2 + 1\ \ 000 = 1\ \ 002$
Q4 ($t = 40\ \ 000$)	40 000	$200 + 40\ \ 000 = 40\ \ 200$	$2 + 200 + 40\ \ 000 = 40\ \ 202$	$2 + 800 = 802$

El hash no sirve para el rango de Q3. Los costos **sobre** $40\,000$ equivalen a no tener índice.

(a) Tabla de costos: reemplazar t

	Full scan	Hash $\lceil t/200 \rceil + t$	Unclustered $2 + \lceil t/200 \rceil + t$	Clustered $2 + \lceil t/50 \rceil$
Q1 ($t = 1$)	40 000	2	4	3
Q2 ($t = 20$)	40 000	21	23	3
Q3 ($t = 50\ \ 000$)	40 000	no aplica (rango)	50 252	1 002
Q4 ($t = 40\ \ 000$)	40 000	40 200	40 202	802

El hash no sirve para el rango de Q3. Los costos **sobre** $40\,000$ equivalen a no tener índice.

(a) ¿Qué gana en cada consulta?

	Mejor método posible	Costo	¿Gana al full scan?
Q1	hash	2	sí
Q2	clustered	3	sí
Q3	clustered	1 002	sí
Q4	clustered	802	sí

- **Q3** y **Q4**: solo el clustered mejora el full scan. En Q3 y Q4 el unclustered **pierde** ($50\,252$ y $40\,202$) y el hash de Q4 también ($40\,200$).
- Un unclustered solo gana si $t \leq 39\,798$ (hash: $t \leq 39\,800$).

(b) Pregunta

> Elija los índices (a lo más 3, a lo más uno clustered) que minimizan el costo total por hora. Dé ese costo y diga qué método usa cada consulta.

(b) Ahorro por hora de cada índice

Índice	Cálculo $f \cdot (40\,000 - \text{costo})$	Ahorro por hora
hash(id)	$20\ \ 000 \cdot (40\ \ 000 - 2)$	**799 960 000**
clustered(id)	$20\ \ 000 \cdot (40\ \ 000 - 3)$	**799 940 000**
hash(cliente)	$5\ \ 000 \cdot (40\ \ 000 - 21)$	**199 895 000**
clustered(cliente)	$5\ \ 000 \cdot (40\ \ 000 - 3)$	**199 985 000**
clustered(fecha)	$50 \cdot (40\ \ 000 - 1\ \ 002)$	**1 949 900**
clustered(estado)	$20 \cdot (40\ \ 000 - 802)$	**783 960**
hash o B+tree **unclustered** para Q3, Q4	pierden contra el full scan (el clustered sí mejora: filas de arriba)	0

- Q_1 y Q_2 se resuelven igual de bien con un **hash**, que no gasta el clustered.
- El clustered se reserva para donde más ahorra de lo que **no** tiene hash: Q3 ($1\,949\,900$) frente a Q4 ($783\,960$).

(b) Comparar configuraciones

Configuración	Total por hora
hash(id) + hash(cliente) + **clustered(fecha)**	**995 100**
hash(id) + hash(cliente) + clustered(estado)	2 161 040
hash(id) + clustered(cliente)	2 855 000
hash(id) + hash(cliente) (sin clustered)	2 945 000
clustered(id) + hash(cliente)	2 965 000
sin índices	1 002 800 000

> **Elegir:** hash(`id`) + hash(`cliente`) + clustered(`fecha`) → los **3 índices**, 1 clustered.

(b) Detalle de la mejor configuración

Consulta	Método	Costo	Costo × frecuencia
Q1	hash(id)	2	40 000
Q2	hash(cliente)	21	105 000
Q3	clustered(fecha)	1 002	50 100
Q4	full scan	40 000	800 000
		Total por hora	**995 100**

Sin índices: $40\,000 \cdot (20\,000 + 5\,000 + 50 + 20) = 40\,000 \cdot 25\,070 = \mathbf{1\,002\,800\,000}$.

> Q_4 es el 80% del costo que queda ($800\,000$ de 995 100) y **ningún** unclustered la ayuda. Se queda en full scan.

(c) Pregunta

> Justifique por qué el índice clustered conviene sobre `fecha` y no sobre `cliente`, `estado` o `id`.

(c) ¿Por qué el clustered va sobre `fecha`?

Clustered sobre	Ahorro por hora	Cálculo
fecha	**1 949 900**	$50 \cdot (40\ \ 000 - 1\,002)$
estado	783 960	$20 \cdot (40\ \ 000 - 802)$
cliente	90 000	$5\,000 \cdot (21 - 3)$ (el hash ya cubría Q2)
id	−20 000	$20\,000 \cdot (2 - 3)$: **empeora**

El clustered conviene donde la consulta devuelve **muchas tuplas contiguas** (rango) y se hace con **frecuencia**: Q3 gana por ambas razones. Sobre `id` incluso **empeora** (el hash cuesta 2, el clustered 3).

(d) Pregunta

> Si solo se permitiera **un** índice, ¿cuál sería? Y si Q_4 devolviera solo 200 tuplas, ¿valdría la pena un índice sobre `estado`?

(d) Si solo se permitiera un índice

Un solo índice	Total por hora
hash(id)	**202 840 000**
clustered(id)	202 860 000
clustered(cliente)	802 815 000
hash(cliente)	802 905 000
clustered(fecha)	1 000 850 100
clustered(estado)	1 002 016 040
sin índices	1 002 800 000

El mejor es **hash(`id`)** con **202 840 000** por hora: Q_1 es cuatro veces más frecuente que Q_2 y sin índice cada ejecución cuesta $40\,000$.

(d) Si Q_4 devolviera solo 200 tuplas

	Costo por consulta	Por hora ($f = 20$)
Full scan	$40\,000$	$800\,000$
hash(`estado`)	$\lceil 200/200 \rceil + 200 = 201$	$20 \cdot 201 = 4\,020$

> **Sí vale la pena**: ahora $t = 200$ está muy por debajo del umbral ($\approx 39\,800$).

Lo que cambió no es el índice, sino **cuántas tuplas devuelve** la consulta.

Pregunta 5

Reescritura y optimización con estadísticas

Enunciado

```sql
SELECT DISTINCT P.name, M.stadium
FROM Teams T, Players P, PlayersMatches PM, Matches M
WHERE T.id = P.tid AND P.id = PM.pid AND PM.mid = M.id
  AND T.country = 'Chile' AND PM.goals >= 2 AND M.date >= '2020-01-01'
```

$\mathbf{Players}(\underline{id}, name, year, tid) \cdot \mathbf{Teams}(\underline{id}, name, country) \cdot \mathbf{Matches}(\underline{id}, stadium, date) \cdot \mathbf{PlayersMatches}(\underline{pid}, \underline{mid}, goals)$

$|T| = 300$ (12 chilenos, 10 págs.) · $|P| = 9\,000$ · $|PM| = 150\,000$ (3 000 págs.; $15\,000$ con $goals \geq 2$) · $|M| = 5\,000$ (100 págs.; $1\,000$ con $date \geq 2020$)

(a) plan directo y reescrito · (b) tamaño de 3 planes de join · (c) ¿hace falta δ? · (d) camino de acceso

(a) Pregunta

> Dibuje el plan lógico directo y un plan reescrito completo (selecciones y proyecciones lo más abajo posible, productos convertidos en equijoins).

(a) Las reglas que se usan

Regla	Qué dice
A partir conjunción	$\sigma_{\theta_1 \wedge \theta_2}(R) = \sigma_{\theta_1}(\sigma_{\theta_2}(R))$
B bajar selección	si $\mathrm{att}(\theta) \subseteq \mathrm{att}(R)$: $\sigma_\theta(R \times S) = \sigma_\theta(R) \times S$ (y simétrica)
D producto + igualdad	$R \bowtie_\theta S = \sigma_\theta(R \times S)$ (evita formar el producto)
PP bajar proyecciones	$\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(S))$ si M, N no interfieren con el join ni con L

M y N no interfieren: $(\mathrm{att}(R) \cap \mathrm{att}(S)) \cup (L \cap \mathrm{att}(R)) \subseteq M \subseteq \mathrm{att}(R)$, y análogo para N. Para $\pi_L(\sigma_\theta(R))$ se exige además $\mathrm{att}(\theta) \subseteq M$.

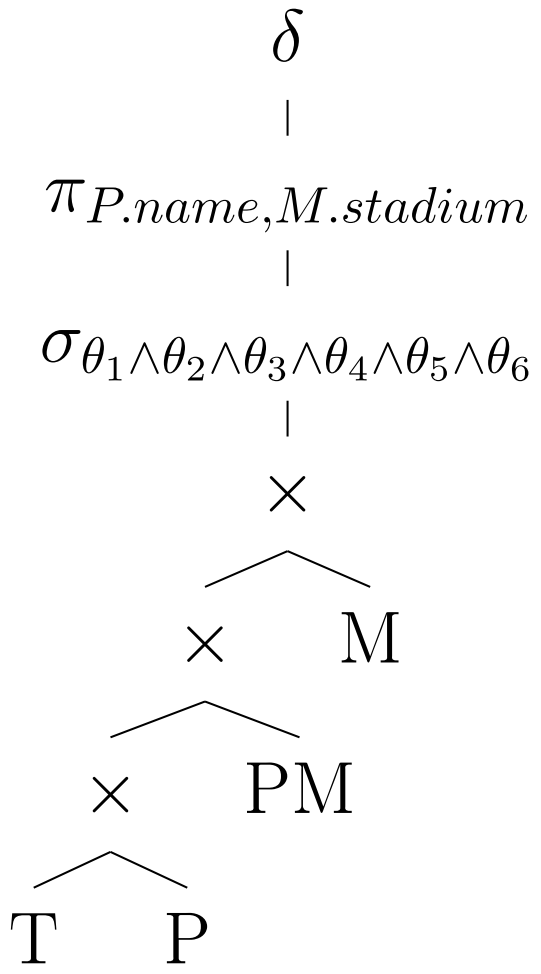
Push-selection: Garcia-Molina, Ullman y Widom, *Database Systems: The Complete Book*, §16.2.2–16.2.3, pp. 772–774.

(a) Paso 0 · el plan directo (traducción de SQL)

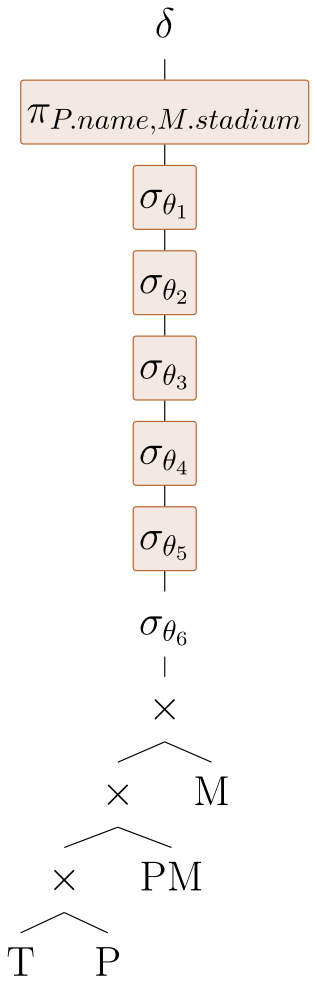
$\pi_{\text{SELECT}}\big(\sigma_{\text{WHERE}}(R_1 \times \cdots \times R_n)\big)$, con δ en la raíz por el `DISTINCT`. Asociación elegida: $((T \times P) \times PM) \times M$

.

	Condición
θ_1	$T.id = P.tid$
θ_2	$P.id = PM.pid$
θ_3	$PM.mid = M.id$
θ_4	$T.country = \text{Chile}$
θ_5	$PM.goals \geq 2$
θ_6	$M.date \geq \text{2020-01-01}$



(a) Paso 1 · partir la conjunción (Regla A)



Seis selecciones encadenadas; **todavía están todas en la raíz**.

> **Regla A** (partir una conjunción):
> $\sigma_{\theta_1 \wedge \theta_2}(R) = \sigma_{\theta_1}(\sigma_{\theta_2}(R))$. El orden de las selecciones no importa.

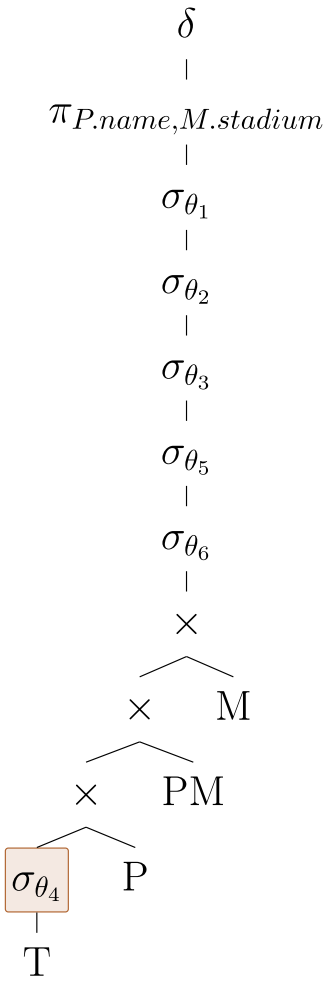
(a) Paso 2 · ¿hasta dónde baja cada condición?

Una selección baja hasta **el nodo más bajo cuyo sub-árbol contiene todos sus atributos**:

	Atributos	Primer nodo que los contiene	Baja hasta
θ_4	$T.country$	la hoja T	encima de T
θ_5	$PM.goals$	la hoja PM	encima de PM
θ_6	$M.date$	la hoja M	encima de M
θ_1	$T.id, P.tid$	$T \times P$	encima de $T \times P$
θ_2	$P.id, PM.pid$	$(T \times P) \times PM$	encima de ese producto
θ_3	$PM.mid, M.id$	el producto con M	**no baja**

> **Regla B** (bajar una selección por un producto): si $\mathrm{att}(\theta) \subseteq \mathrm{att}(R)$, entonces $\sigma_\theta(R \times S) = \sigma_\theta(R) \times S$. Si $\mathrm{att}(\theta) \subseteq \mathrm{att}(S)$: $\sigma_\theta(R \times S) = R \times \sigma_\theta(S)$ (versión simétrica). Vale con duplicados.

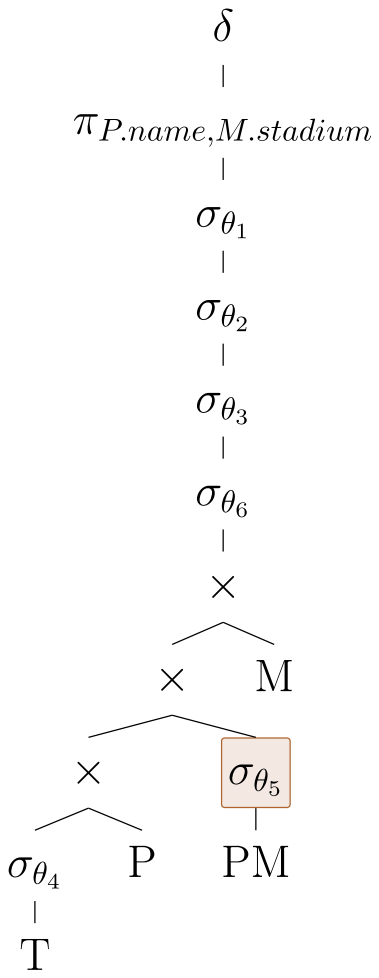
(a) Paso 3 · bajar θ_4 hasta T (Regla B, 3 veces)



$\mathrm{att}(\theta_4) = \{T.country\} \subseteq \mathrm{att}(T)$. Baja por la rama izquierda: sobre el producto con M, sobre $(T \times P) \times PM$, sobre $T \times P$.

> **Regla B** (bajar una selección por un producto): si $\mathrm{att}(\theta) \subseteq \mathrm{att}(R)$, entonces $\sigma_\theta(R \times S) = \sigma_\theta(R) \times S$. Si $\mathrm{att}(\theta) \subseteq \mathrm{att}(S)$: $\sigma_\theta(R \times S) = R \times \sigma_\theta(S)$ (versión simétrica). Vale con duplicados.

(a) Paso 4 · bajar θ_5 hasta PM (Regla B, 2 veces)

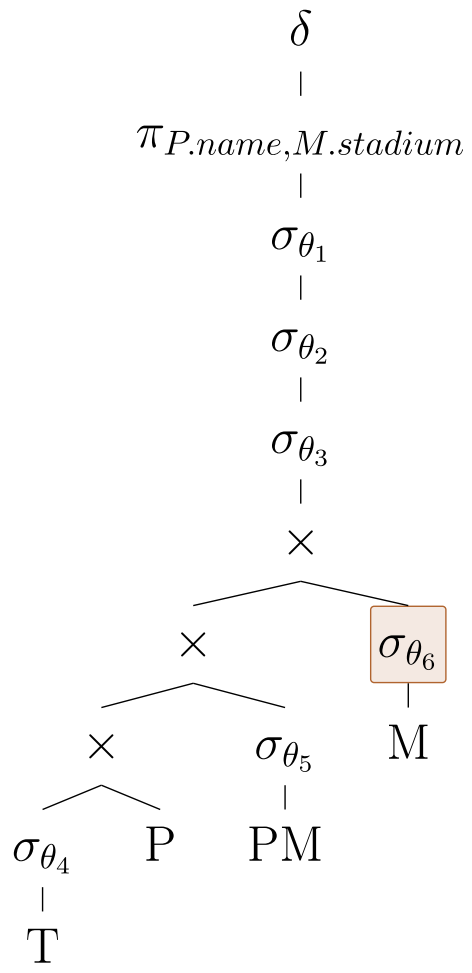


$\mathrm{att}(\theta_5) = \{PM.goals\} \subseteq \mathrm{att}(PM)$.

Primero por la rama izquierda de la raíz; luego, como PM es el hijo **derecho** de $(T \times P) \times PM$, se usa la versión **simétrica**.

> **Regla B** (bajar una selección por un producto): si $\mathrm{att}(\theta) \subseteq \mathrm{att}(R)$, entonces $\sigma_\theta(R \times S) = \sigma_\theta(R) \times S$. Si $\mathrm{att}(\theta) \subseteq \mathrm{att}(S)$: $\sigma_\theta(R \times S) = R \times \sigma_\theta(S)$ (versión simétrica). Vale con duplicados.

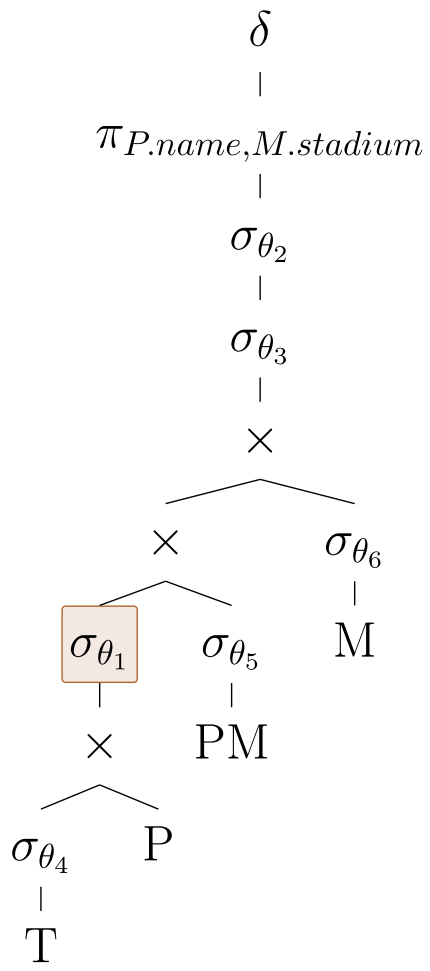
(a) Paso 5 · bajar θ_6 hasta M (Regla B, 1 vez)



$\mathrm{att}(\theta_6) = \{M.date\} \subseteq \mathrm{att}(M)$ y M es el hijo derecho de la raíz: versión simétrica, una sola aplicación.

> **Regla B** (bajar una selección por un producto): si $\mathrm{att}(\theta) \subseteq \mathrm{att}(R)$, entonces $\sigma_\theta(R \times S) = \sigma_\theta(R) \times S$. Si $\mathrm{att}(\theta) \subseteq \mathrm{att}(S)$: $\sigma_\theta(R \times S) = R \times \sigma_\theta(S)$ (versión simétrica). Vale con duplicados.

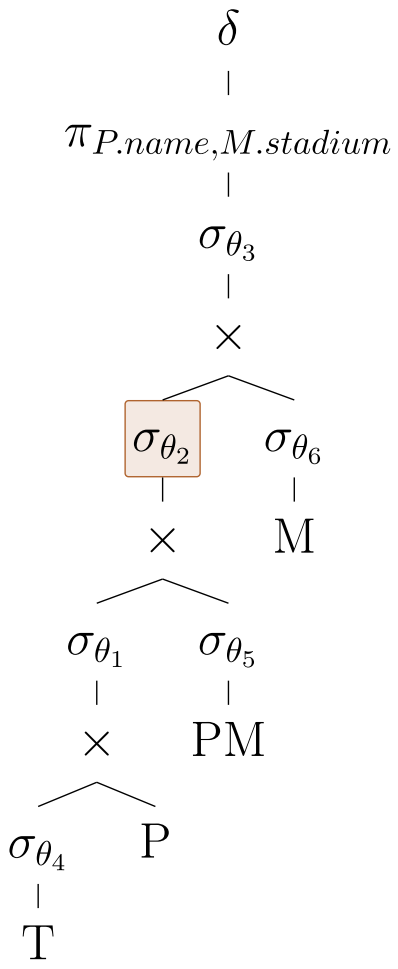
(a) Paso 6 · bajar θ_1 hasta $T \times P$ (Regla B, 2 veces)



$\mathrm{att}(\theta_1) = \{T.id, P.tid\}$ no está en T ni en P por separado, pero sí en $T \times P$. Baja dos veces y **se detiene** encima de $T' \times P$ (más abajo ya no estarían ambos atributos).

> **Regla B** (bajar una selección por un producto): si $\mathrm{att}(\theta) \subseteq \mathrm{att}(R)$, entonces $\sigma_\theta(R \times S) = \sigma_\theta(R) \times S$. Si $\mathrm{att}(\theta) \subseteq \mathrm{att}(S)$: $\sigma_\theta(R \times S) = R \times \sigma_\theta(S)$ (versión simétrica). Vale con duplicados.

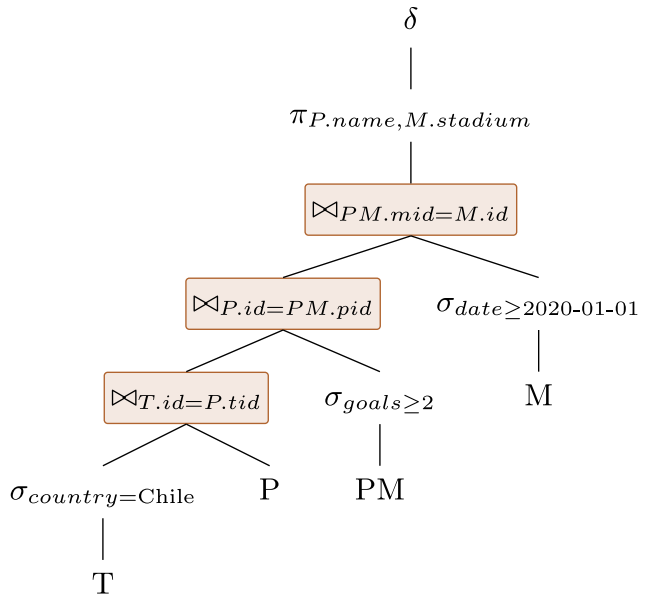
(a) Paso 7 · bajar θ_2 (Regla B, 1 vez)



$\mathrm{att}(\theta_2) = \{P.id, PM.pid\}$ está en $(T' \times P) \times PM'$, hijo izquierdo de la raíz: una aplicación. θ_3 **no baja**: necesita PM y M, que solo se juntan en la raíz.

> **Regla B** (bajar una selección por un producto): si $\mathrm{att}(\theta) \subseteq \mathrm{att}(R)$, entonces $\sigma_\theta(R \times S) = \sigma_\theta(R) \times S$. Si $\mathrm{att}(\theta) \subseteq \mathrm{att}(S)$: $\sigma_\theta(R \times S) = R \times \sigma_\theta(S)$ (versión simétrica). Vale con duplicados.

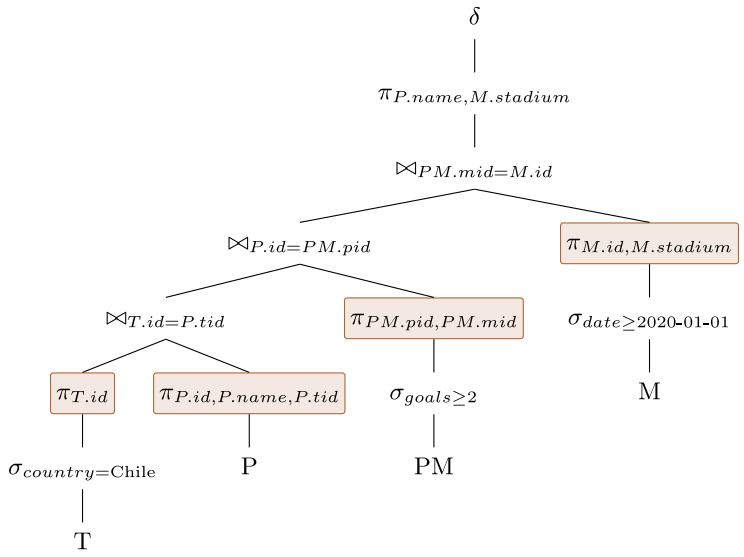
(a) Paso 8 · producto + igualdad = equijoin (Regla D, 3 veces)



$\sigma_{\theta_1}(T' \times P) = T' \bowtie_{\theta_1} P$, luego θ_2 con PM' y por último θ_3 con M'. **Ya no se forma ningún producto cruz.**

> **Regla D** (producto + igualdad = equijoin): $R \bowtie_\theta S = \sigma_\theta(R \times S)$. Convertirlo evita formar el producto y permite elegir un algoritmo de join.

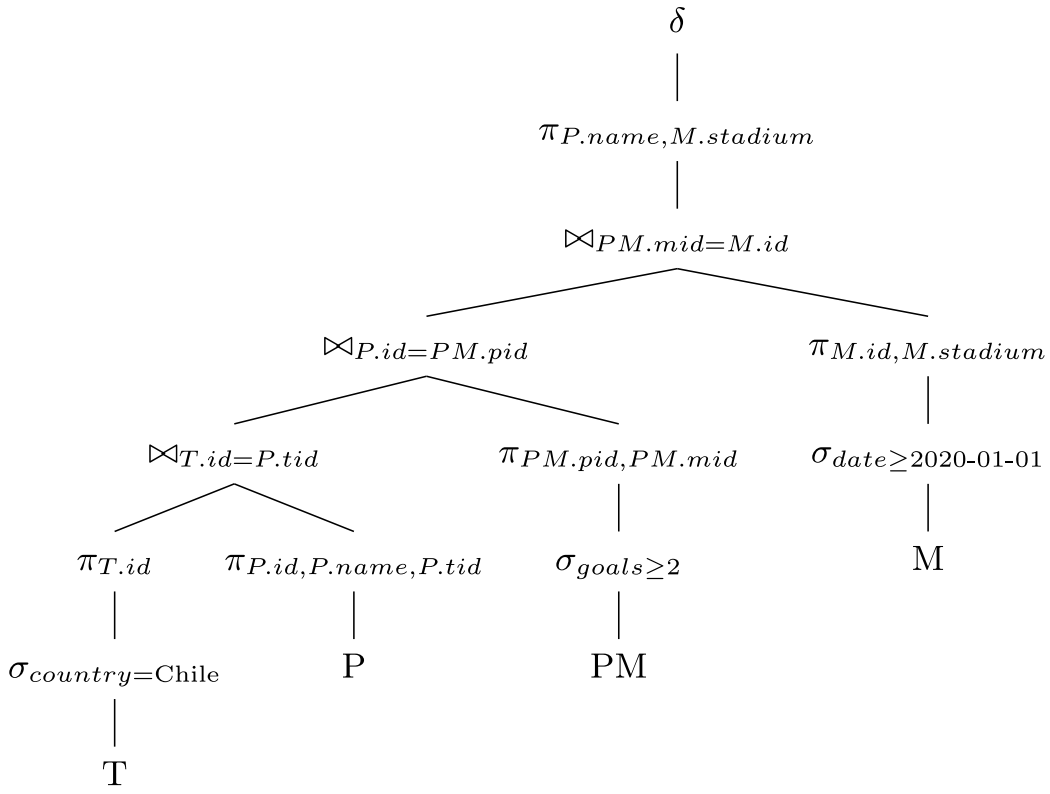
(a) Paso 9 · bajar proyecciones (Regla PP, 4 veces)



Sobre cada hoja filtrada va lo que se usa **más arriba** (joins y salida). Las π van **encima** de las σ: la selección todavía necesita su atributo.

> **Regla PP** (bajar proyecciones): $\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(S))$ si M y N no interfieren con el join ni con L. Para $\pi_L(\sigma_\theta(R)) = \pi_L(\sigma_\theta(\pi_M(R)))$ se exige además $\mathrm{att}(\theta) \subseteq M$.

(a) Resultado: el plan reescrito completo



Reglas usadas: A $\times 1$ · B $\times 9$ $(3+2+1+2+1)$ · D $\times 3$ · PP $\times 4$.

(a) Verificación del plan final

Cada operador usa **solo atributos que recibe**:

Operador	Usa	Lo recibe de
$\sigma_{country=\text{Chile}}$	$T.country$	T ✓
$\bowtie_{T.id=P.tid}$	$T.id$, $P.tid$	$\pi_{T.id}(\ldots)$ y $\pi_{P.id,P.name,P.tid}(P)$ ✓
$\bowtie_{P.id=PM.pid}$	$P.id$, $PM.pid$	el join anterior y $\pi_{PM.pid,PM.mid}(\ldots)$ ✓
$\bowtie_{PM.mid=M.id}$	$PM.mid$, $M.id$	el join anterior y $\pi_{M.id,M.stadium}(\ldots)$ ✓
$\pi_{P.name,M.stadium}$	$P.name$, $M.stadium$	P y M ✓

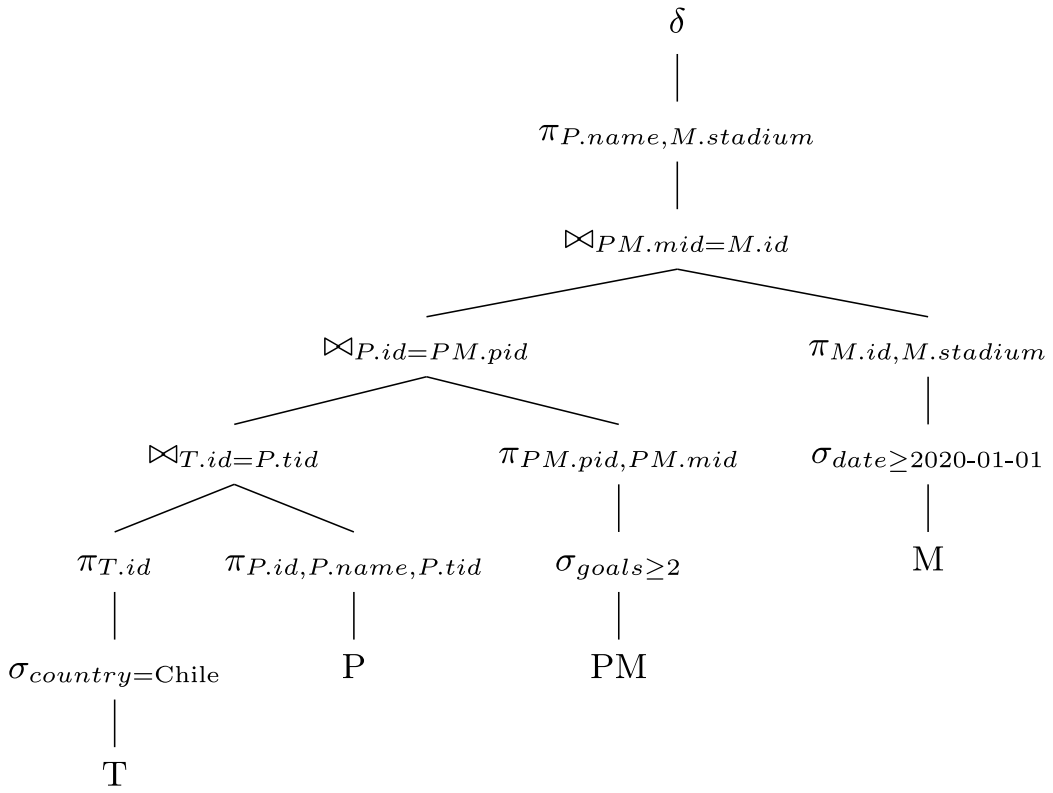
(b) Pregunta

Calcule el **tamaño** de los planes de join (la suma de las cardinalidades de los resultados de sus joins):

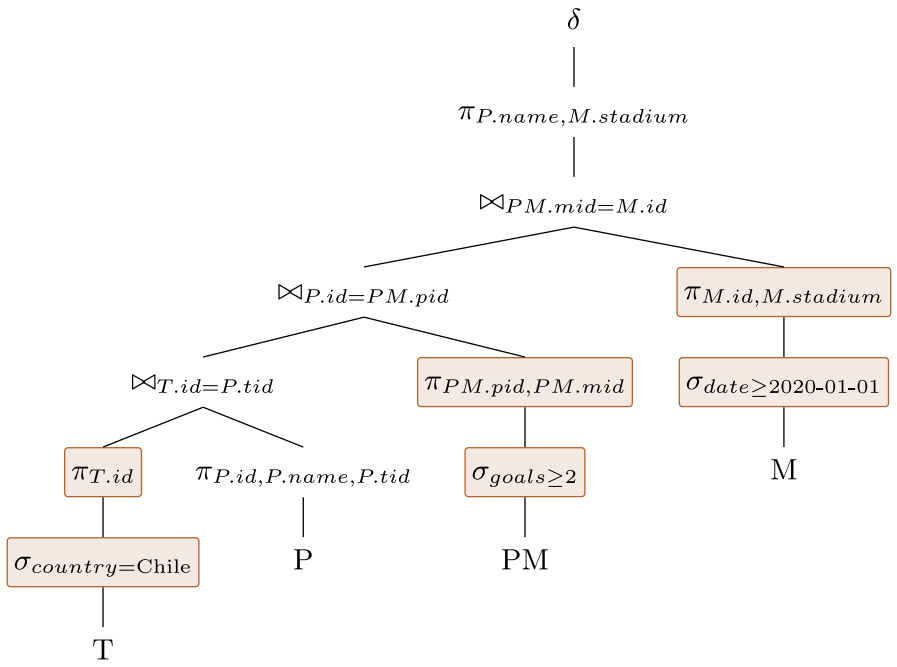
- A: $((T' \bowtie P) \bowtie PM') \bowtie M'$
- B: $T' \bowtie (P \bowtie (PM' \bowtie M'))$
- C: $(T' \bowtie P) \bowtie (PM' \bowtie M')$

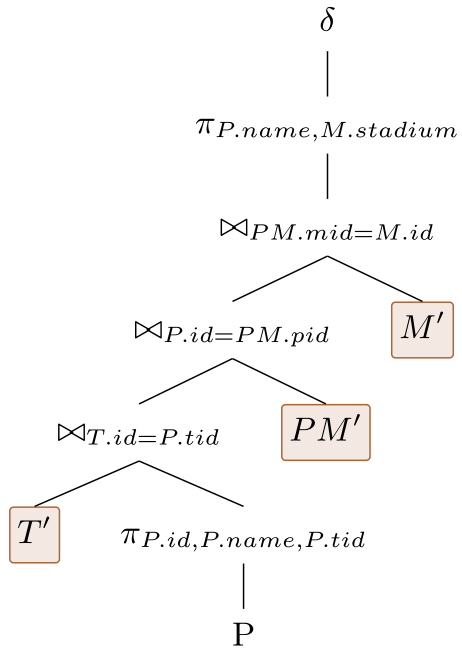
donde T', PM' y M' son las relaciones ya filtradas por sus condiciones. ¿Cuál es el mejor y por qué?

(b) Partimos del plan reescrito completo



(b) Qué son T', PM' y M'





Cada subárbol resaltado (**σ y luego π sobre la relación**) pasa a ser una hoja con apóstrofe:

$$T' = \pi_{T.id}(\sigma_{country=\text{Chile}}(T)) \cdot PM' = \pi_{pid,mid}(\sigma_{goals\geq 2}(PM)) \cdot M' = \pi_{M.id,M.stadium}(\sigma_{date\geq 2020}(M))$$

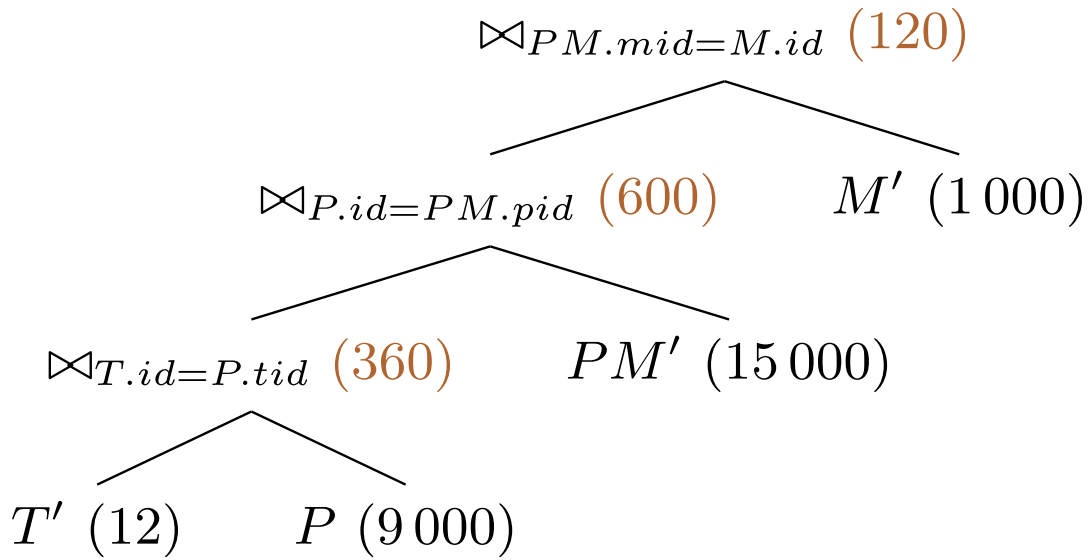
Las π no cambian el **número de filas**, así que T', PM' y M' tienen las cardinalidades de las relaciones **ya filtradas**. P no tiene selección: no lleva apóstrofe.

(b) Datos para los tamaños

Base	Valor
$\lvert T' \rvert$	12 equipos chilenos (30 jugadores cada uno: $9\,000/300$)
$\lvert PM' \rvert$	$15\,000$ (con $goals \geq 2$)
$\lvert M' \rvert$	$1\,000$ (20% de los $5\,000$ partidos)
filas de PM' por jugador	$15\,000/9\,000 = 5/3$
fracción de PM' de equipos chilenos	$12/300 = 4\%$

> **Tamaño de un plan** = suma de las cardinalidades de los resultados de sus joins.

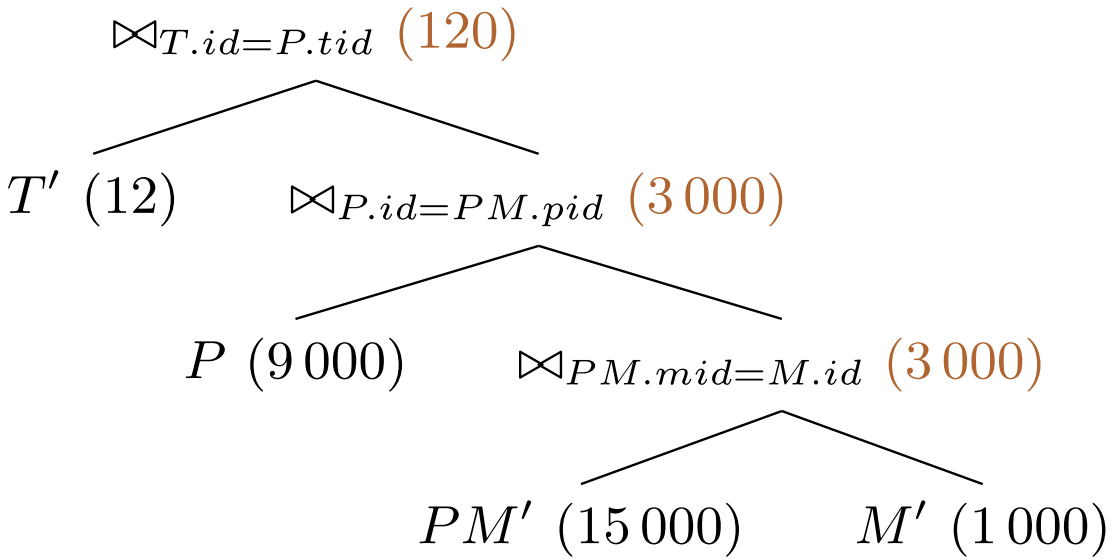
(b) Plan $A = ((T' \bowtie P) \bowtie PM') \bowtie M'$



$360 + 600 + 120 = \mathbf{1\,080}$

$T' \bowtie P$: $12 \cdot 30 = 360$ · $\bowtie PM'$: $360 \cdot 5/3 = 600$ · $\bowtie M'$: $600 \cdot 0{,}2 = 120$

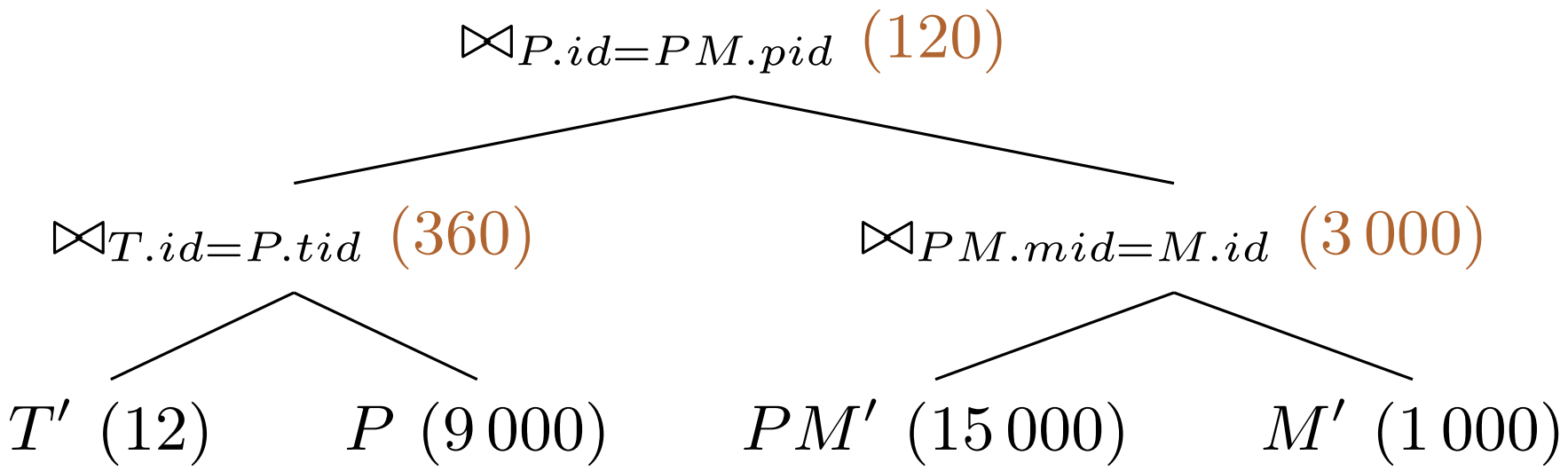
(b) Plan $B = T' \bowtie (P \bowtie (PM' \bowtie M'))$



$3\,000 + 3\,000 + 120 =$ **6 120**

primero $PM' \bowtie M'$: $15\,000 \cdot 0{,}2 = 3\,000$ · $P \bowtie (\cdot)$: cada fila referencia un jugador $\Rightarrow 3\,000$ · $T' \bowtie (\cdot)$: $3\,000 \cdot 0{,}04 = 120$

(b) Plan $C = (T' \bowtie P) \bowtie (PM' \bowtie M')$



$360 + 3\,000 + 120 =$ **3 480**

$T' \bowtie P$: 360 · $PM' \bowtie M'$: $3\,000$ · unión de ambos: $3\,000 \cdot 0{,}04 = 120$

(b) ¿Cuál es el mejor?

Plan	Tamaño
A	**1 080**
C	3 480
B	6 120

> A: empieza por la relación más selectiva (T', 12 filas) y arrastra resultados pequeños. B y C forman primero $PM' \bowtie M'$, que ya tiene $3\,000$ filas.

(c) Pregunta

> ¿Es necesario el δ de la consulta? Dé un ejemplo si es necesario. ¿Y si el `SELECT` fuera `DISTINCT P.id, M.id` ? Justifique usando las llaves.

(c) ¿Es necesario el δ?

`DISTINCT P.name, M.stadium` : SÍ.

Pueden repetirse $(name, stadium)$ de dos formas: dos jugadores chilenos **distintos con el mismo nombre** que marcaron ≥ 2 goles en un mismo estadio desde 2020, o **un mismo jugador** en dos partidos del mismo estadio.

`DISTINCT P.id, M.id` : NO.

Cada fila del resultado sale de **una fila de** PM, cuya llave primaria es (pid, mid); y P, T, M se unen por llave (muchos-a-uno hacia PM). Como $(P.id, M.id) = (pid, mid)$, no hay repetidos: $\delta(R) = R$.

(d) Pregunta

> Para elegir cómo leer cada relación filtrada, use el modelo de costos de la pregunta 3 (full scan $=$ páginas; hash: $\lceil t/200 \rceil + t$; B+tree unclustered: $2 + \lceil t/200 \rceil + t$; B+tree clustered: $2 + \lceil t/\text{tuplas por página} \rceil$). El catálogo indica: un hash sobre `Teams.country`, un B+tree unclustered sobre `PlayersMatches.goals` y un B+tree clustered sobre `Matches.date`. Decida, para cada una de las tres selecciones, si conviene usar el índice o hacer full scan.

(d) Cómo leer cada relación filtrada

Selección	t	Índice	Con índice	Full scan	Decisión
$\sigma_{country=\text{Chile}}(T)$	12	hash	13	10	**full scan**
$\sigma_{goals\geq 2}(PM)$	15 000	B+tree unclustered	15 077	3 000	**full scan**
$\sigma_{date\geq 2020}(M)$	1 000	B+tree clustered	22	100	**usar el índice**

Modelo de la pregunta 3 (páginas: $T = 10$, $PM = 3\,000$, $M = 100$). **Solo el clustered ayuda**: lee páginas contiguas. El hash pierde porque T es tan chica que leerla toda cuesta menos que buscar 12 tuplas una a una; el unclustered pierde porque devuelve el 10% de PM.