Pauta Ayudantía 8

Hashing, índices y reescritura de consultas

IIC2413, Bases de Datos
Benjamín Saavedra

Convenciones

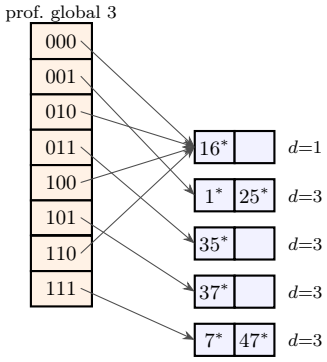
- **Extendable hashing.** Los datos son naturales en binario y $h_n(k)$ son los *últimos* n bits de k. Un bucket guarda a lo más B datos. Se parte con profundidad global 0 y un bucket vacío de profundidad local 0. Se cuenta un *split* cada vez que un bucket se divide y un *doblamiento* cada vez que la profundidad global aumenta en uno. Si después de un split el dato sigue sin caber, se aplica de nuevo el algoritmo de inserción hasta que quepa; por eso una misma inserción puede provocar varios splits y varios doblamientos.
- **B+tree de orden d.** Hojas con entre d y $2d$ datos y directorios (salvo la raíz) con entre d y $2d$ separadores. Split de hoja: d datos a la izquierda, $d+1$ a la derecha, y el separador se *copia* al padre. Split de directorio: el separador *sube*. Se inserta sin redistribución.
- **Operadores con duplicados.** Como en la clase 16, los operadores existen en dos versiones, sin y con duplicados. SQL trabaja con duplicados por defecto: una tupla puede aparecer más de una vez en una relación. Si t aparece r veces en R y s veces en S, entonces aparece $r+s$ veces en $R \cup S$ (como `UNION ALL`) y mín(r, s) veces en $R \cap S$ (`INTERSECT ALL`). El δ deja una sola copia de cada tupla y π no elimina duplicados.
- **Costos.** Se miden en páginas leídas. Cada pregunta indica el modelo que corresponde.

Pregunta 1: La historia de un extendable hashing

Un extendable hashing con $B = 2$ recibió *solo inserciones* de los datos

$$S = \{1, 7, 16, 25, 35, 37, 47\},$$

en un orden que **no conocemos**. Su estado final es:



(a) Sin conocer el orden, determine cuántos splits y cuántos doblamientos ocurrieron en total. Justifique que esos números no dependen del orden de inserción.

(b) Dé un orden de inserción de S en el que *una sola inserción* provoque todos los doblamientos. Justifique por qué ninguna inserción, en ningún orden, puede provocar más doblamientos que esos.

(c) Dé un orden de inserción de S en el que ninguna inserción provoque más de un split.

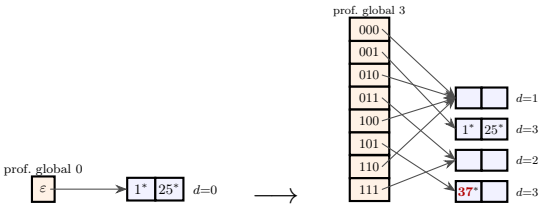
(d) Para el estado final de la figura, determine si existe un dato $k \notin S$ cuya inserción provoque: (i) ningún split; (ii) un split sin doblamiento; (iii) un split con doblamiento; (iv) dos splits. Si existe, dé una; si no existe, justifique. Si (ii) no existe, ¿cuál es la secuencia más corta de inserciones que lo provoca?

Pauta.

(a) Cada split reemplaza un bucket por dos: el número de buckets sube en 1. Se parte con 1 bucket y se termina con 5: hubo exactamente $5 - 1 =$ **4 splits**. Cada doblamiento sube la profundidad global en 1. Se parte con $n = 0$ y se termina con $n = 3$: hubo exactamente **3 doblamientos**. Como solo hay inserciones (no hay merges), ambos contadores solo pueden crecer, y su valor final queda determinado por el estado final, *cualquiera* sea el orden y aunque una misma inserción provoque varios.

(b) Como en total hay 3 doblamientos, ninguna inserción puede provocar más de 3: esa es la cota, y para alcanzarla una sola inserción debe provocarlos todos. Como n sube de a uno, justo antes $n = 0$: hay un único bucket, lleno, con 2 datos, y el tercero dispara todo. Al dividir ese bucket, los tres datos se reparten por el bit siguiente; si coinciden en él siguen juntos y hay que repetir (una *cascada*), hasta el primer bit en que no coinciden, donde quedan repartidos 1 y 2 y caben ($B = 2$). Con $n = d = 0$ cada split necesita además un doblamiento, así que para obtener 3 buscamos tres datos cuyo primer bit distinto sea el bit 3: que coincidan en los bits 1 y 2 y difieran en el 3, es decir, que terminen igual en dos bits ($\{1, 25, 37\}$ o $\{7, 35, 47\}$). Con otra función de hash el razonamiento es el mismo, con los bits de $h'(k)$; cambian los grupos.

Sirve el orden $1, 25, 37, 7, 16, 35, 47$. Con $1 = \texttt{001}$ y $25 = \texttt{11001}$ el único bucket queda lleno. Al insertar $37 = \texttt{100101}$, los tres coinciden en el bit 1 (`1`) y en el bit 2 (`0`), así que los dos primeros splits los dejan juntos y obligan a doblar ($n = 0 \to 1 \to 2$); recién el bit 3 separa a 37 de $\{1, 25\}$ (que valen `0`, `0` frente a `1`). Total para la inserción del 37: **3 splits y 3 doblamientos**. Ojo: los tres comparten solo 2 bits y aun así hay 3 splits, porque el split del bit en que difieren también cuenta.



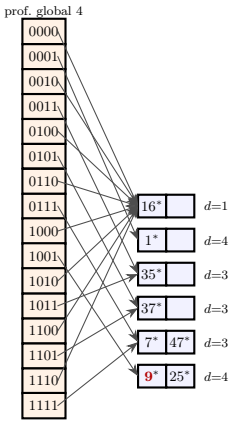
Los demás datos ($7, 16, 35$ y 47) no vuelven a doblar el directorio (el 47 provoca solo el cuarto split).

(c) Una inserción provoca 2 o más splits cuando, tras dividir, los tres datos siguen juntos (una cascada), y solo pueden hacerlo los grupos $\{1, 25, 37\}$ y $\{7, 35, 47\}$, que comparten 2 bits y difieren en el 3. Si el tercero de un grupo llega a un bucket que tiene a los otros dos y profundidad d, ese bucket ya usó d de esos bits: quedan $2 - d$ splits que no separan nada, más el del bit 3 que sí separa, $3 - d$ en total ($d = 0$: 3; $d = 1$: 2; $d = 2$: 1). Para evitar la cascada el bucket debe tener ya $d \geq 2$ cuando llega el tercero, y eso se logra haciendo crecer la profundidad de a un bit: partiendo de $\{1, 25\}$, un dato sirve si su bit $d + 1$ difiere del de 1 y 25 (un solo split). Primero el 16 (bit 1), luego 7, 35 o 47 (bit 2), y recién entonces el 37.

El orden creciente $1, 7, 16, 25, 35, 37, 47$ funciona, y también $1, 25, 16, 7, 37, 35, 47$. En el creciente las inserciones provocan, (splits, doblamientos): 1: $(0, 0)$; 7: $(0, 0)$; 16: $(1, 1)$; 25: $(1, 1)$; 35: $(0, 0)$; 37: $(1, 1)$; 47: $(1, 0)$. Ninguna pasa de un split, y los totales son 4 y 3, lo que concuerda con (a). (El $1, 25, 37, \ldots$ de (b) es el ejemplo contrario.)

(d) Los buckets finales son $\{16\}$ con $d = 1$ (entradas `000`, `010`, `100`, `110`), $\{1, 25\}$ (`001`), $\{35\}$ (`011`), $\{37\}$ (`101`) y $\{7, 47\}$ (`111`), los cuatro con $d = 3 = n$.

- **(i) Sin split: existe.** Cualquier dato que llegue a un bucket con espacio, es decir, que no termine en `001` ni en `111`: por ejemplo $3 = \texttt{11}$ ($h_3 = \texttt{011}$, bucket $\{35\}$) o cualquier dato par, como 2 (bucket $\{16\}$).
- **(ii) Split sin doblamiento: no existe.** Requiere un bucket *lleno* con $d < n$. El único con $d < n$ es $\{16\}$, que tiene un solo dato: no está lleno. *Secuencia más corta:* dos inserciones. Primero un dato par que llene $\{16\}$, por ejemplo 2 ($\{16, 2\}$), y luego otro par que caiga ahí, por ejemplo 4: como $d = 1 < n = 3$, el bucket $\{16, 2\}$ se divide por el bit 2 ($16 \to \texttt{0}$; $2 \to \texttt{1}$) sin doblar, y el 4 (que tiene bit 2 igual a `0`) se reinserta junto al 16. Para que el segundo dato divida con un solo split, sea cual sea, el primero debe tener el bit 2 distinto del 16 ($2, 6, 10, \ldots$); con 4 y luego 8 habría cascada.
- **(iii) Split con doblamiento: existe.** Un dato que llegue a un bucket lleno con $d = n$, es decir, que termine en `001` o en `111`: por ejemplo $9 = \texttt{1001}$ ($h_3 = \texttt{001}$). Se dobla ($n = 4$), se divide $\{1, 25\}$ por el bit 4 ($1 \to \texttt{0}$, $25 \to \texttt{1}$) y el 9 queda con el 25.



- **(iv) Dos splits: no existe.** Habría que dividir un bucket lleno y que sus datos cayeran todos del mismo lado del bit $d+1$. Los únicos buckets llenos son $\{1, 25\}$ y $\{7, 47\}$, y en ambos los dos datos *difieren* en el bit 4 ($1 = \texttt{0001}$, $25 = \texttt{11001}$; $7 = \texttt{0111}$, $47 = \texttt{101111}$). Un split los separa siempre, y el dato nuevo queda en un bucket con a lo más un dato: cabe.

Pregunta 2: B+tree versus extendable hashing ★

Considere un B+tree de orden $d = 2$ y un extendable hashing con buckets de capacidad $B = 4$ (sin páginas de overflow). En el B+tree cada página (hoja o directorio) guarda a lo más 4 datos o 4 separadores; en el extendable hashing cada página del directorio guarda 5 punteros. El número de páginas de un índice es la suma de las de datos (hojas o buckets) y las de directorio. Se insertan números positivos.

(a) Dé una secuencia de 12 inserciones tal que el B+tree quede con *más* páginas que el extendable hashing.

(b) Dé una secuencia de 12 inserciones tal que el extendable hashing quede con *más* páginas que el B+tree.

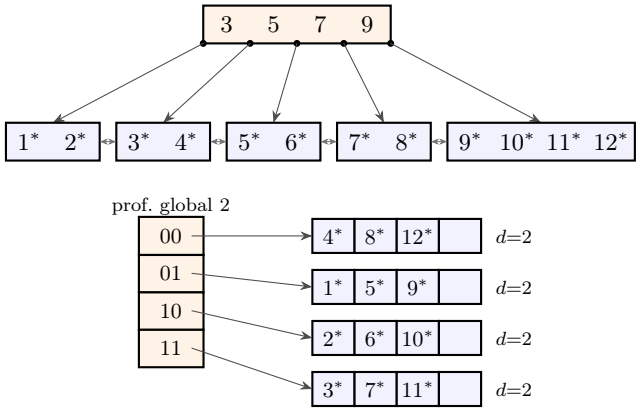
(c) Para cada secuencia, dibuje el estado final de ambos índices y cuente sus páginas.

(d) Explique, en una frase por secuencia, qué característica de los datos explotó cada una y qué índice castiga.

(adaptada de la Interrogación 1 de 2022)

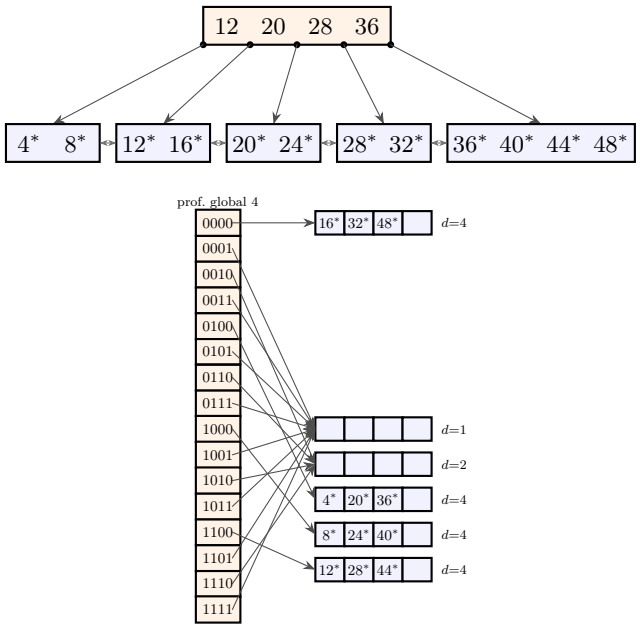
Pauta. Hay muchas respuestas correctas. Dos que funcionan:

(a) Hay que bajar la ocupación del B+tree y subir la del hash. En el B+tree cada split de hoja deja $d = 2$ datos a la izquierda y 3 a la derecha, y un dato nunca sale de su hoja: si los datos nuevos siempre caen en la hoja de más a la derecha (secuencia creciente), las hojas de la izquierda quedan con 2 datos para siempre. Así, 12 datos ocupan 5 hojas (con 6 habría que tener 13 datos, porque la hoja que crea el último split ya tiene 3) más un directorio: 6 páginas, que es lo máximo posible. En el hash conviene lo contrario: que los datos se repartan parejo en los últimos bits, para que ningún bucket se desborde antes de tiempo. Con 12 datos consecutivos cada clase módulo 4 tiene 3 datos, caben en un bucket ($B = 4$) y alcanza $n = 2$: 4 buckets y un directorio de una página. Sirve entonces la secuencia ascendente $1, 2, \ldots, 12$.



B+tree: 5 hojas + 1 directorio = **6** páginas. Extendable hashing ($n = 2$, 4 buckets): $4 + \lceil 4/5 \rceil = \mathbf{5}$ páginas.

(b) Ahora hay que castigar al hash. Su directorio crece según cuántos bits hay que mirar para repartir los datos: si todos comparten un sufijo de s bits, esos bits no reparten nada pero igual obligan a doblar el directorio, y cada split por ellos deja un bucket vacío. Como 12 datos no caben en un bucket, hay que repartirlos en al menos 3 buckets, lo que exige 2 bits que sí los repartan: con $s = 2$ el directorio necesita $n = s + 2 = 4$ (16 entradas, 4 páginas) en vez de $n = 2$. El B+tree no ve el sufijo: mientras la secuencia siga creciendo queda igual que en (a), con 6 páginas. Sirven los múltiplos de 4, $4, 8, 12, \ldots, 48$, que terminan todos en `00`: el hash queda con $n = 4$, 4 buckets de 3 datos y 2 vacíos (los de los splits por los bits 1 y 2).



B+tree: **6** páginas. Extendable hashing: $6 + \lceil 16/5 \rceil = 6 + 4 = \mathbf{10}$ páginas.

(c) Ver las figuras de (a) y (b), con los conteos indicados (6 vs. 5 y 6 vs. 10).

(d) La primera secuencia es *ordenada*: castiga al B+tree (hojas a medio llenar). La segunda es una secuencia de datos que *comparten sufijo* en binario: castiga al extendable hashing, cuyo directorio crece exponencialmente con el largo del sufijo común (el problema se mitiga aplicando h_n al resultado de una buena función de hash).

Pregunta 3: Diseño de índices para una carga de trabajo

La relación **Pedidos**(id, cliente, fecha, monto, estado) tiene 2 000 000 de tuplas, con 50 tuplas por página ($P = 40\,000$ páginas). Se pueden crear a lo más **3 índices**, **a lo más uno clustered**. Modelo de costos (en páginas leídas) para una consulta que devuelve t tuplas; el optimizador elige siempre el método más barato, incluido el *full scan*. Para el hash se asume que los buckets guardan entradas (llave, puntero), 200 por página; que las páginas del bucket solo contienen entradas de la llave buscada; y que el directorio está en memoria, no en disco (su lectura no cuenta):

Método	Costo
Full scan	$P = 40\,000$
Hash sobre el atributo (solo igualdad; siempre unclustered)	$\lceil t/200 \rceil + t$
B+tree unclustered (hojas con 200 data entries, 2 niveles de directorio)	$2 + \lceil t/200 \rceil + t$
B+tree clustered (las hojas son las páginas de datos, 50 tuplas)	$2 + \lceil t/50 \rceil$

La carga de trabajo, con su frecuencia por hora:

	Consulta	Tuplas devueltas t	Frecuencia
Q_1	`WHERE id = ...`	1	20 000
Q_2	`WHERE cliente = ...`	20	5 000
Q_3	`WHERE fecha BETWEEN ...` (un mes)	50 000	50
Q_4	`WHERE estado = 'pendiente'`	40 000	20

El costo total por hora es $\sum_i f_i \cdot (\text{costo de } Q_i)$.

(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?

Pauta.

(a)

	Full scan	Hash	B+tree unclustered	B+tree clustered
Q_1 $(t = 1)$	40 000	2	4	3
Q_2 $(t = 20)$	40 000	21	23	3
Q_3 $(t = 50\,000)$	40 000	no aplica (rango)	50 252	1 002
Q_4 $(t = 40\,000)$	40 000	40 200	40 202	802

(Por ejemplo, Q_3 unclustered: $2 + 250 + 50\,000 = 50\,252$; clustered: $2 + 1\,000 = 1\,002$.) *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. En el clustered las hojas son las páginas de datos, con tuplas completas (50

por página), y las t tuplas de un rango están contiguas: $\lceil t/50 \rceil$ páginas. En el unclustered las hojas guardan *data entries* (llave y puntero a la tupla), mucho más chicas que una tupla, y caben 200 por hoja: recorrer las hojas cuesta $\lceil t/200 \rceil$, pero después hay que leer la página de datos de cada tupla, que están desparramadas, es decir t lecturas más. En el hash, las t entradas con la misma llave caen todas en el mismo bucket y ocupan $\lceil t/200 \rceil$ páginas (si $t \leq 200$, una sola), y después hay que leer la página de datos de cada tupla: t lecturas más. Por eso el peso del unclustered y del hash está en el término t. Un índice unclustered solo gana al full scan si t es menor que $\approx P$: con estos números, $t \leq 39\,800$ para el hash y $t \leq 39\,798$ para el B+tree (alrededor del 2 % de la relación).

(b) Los índices: **hash sobre** id, **hash sobre** cliente y **B+tree clustered sobre** fecha.

Consulta	Método	Costo	Costo × frecuencia
Q_1	hash(id)	2	40 000
Q_2	hash(cliente)	21	105 000
Q_3	clustered(fecha)	1 002	50 100
Q_4	full scan (ningún índice elegido la ayuda)	40 000	800 000
Total por hora			**995 100**

Sin índices el total sería $40\,000 \cdot 25\,070 = 1\,002\,800\,000$. Observe que Q_4 es el 80 % del costo restante y que *ningún* índice unclustered la ayuda ($t = 40\,000 \approx P$); como solo hay un clustered y Q_3 gana mucho más con él, Q_4 se queda en full scan.

(c) El clustered debe ir donde ahorra más *ponderado por frecuencia*, y ahorra más cuando la consulta devuelve muchas tuplas contiguas (rango). Ahorro por hora respecto de la mejor alternativa sin clustered:

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 Q_2)
id	20 000	$20\,000 \cdot (2 - 3)$: *empeora* (el hash es mejor)

Comprobación con los totales: clustered(fecha) = 995 100; clustered(estado) = 2 161 040; clustered(cliente) con hash(id) = 2 855 000.

(d) Con un solo índice, el mejor es **hash sobre** id: total 202 840 000 por hora (el hash sobre cliente da 802 905 000, porque Q_1 es cuatro veces más frecuente que Q_2 y cada vez que no hay índice cuesta 40 000). Si Q_4 devolviera 200 tuplas, un hash sobre estado costaría $\lceil 200/200 \rceil + 200 = 201$ por consulta: $20 \cdot 201 = 4\,020$ por hora contra 800 000 del full scan; sí valdría la pena. Lo que cambia es que t pasa a estar muy por debajo del umbral $\approx 2\,\%$.

Pregunta 5: Reescritura y optimización con estadísticas

Esquema (llaves subrayadas):

$$\textbf{Players}(\underline{\text{id}}, \text{name}, \text{year}, \text{tid}), \qquad \textbf{Teams}(\underline{\text{id}}, \text{name}, \text{country}),$$
$$\textbf{Matches}(\underline{\text{id}}, \text{stadium}, \text{date}), \qquad \textbf{PlayersMatches}(\underline{\text{pid}}, \underline{\text{mid}}, \text{goals}).$$

Abreviamos P, T, M y PM. Considere la consulta

```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'
```

Estadísticas del catálogo: $|T| = 300$ equipos (12 chilenos, 10 páginas); $|P| = 9\,000$, repartidos uniformemente entre los equipos; $|PM| = 150\,000$ (3 000 páginas), de las cuales 15 000 cumplen $goals \geq 2$, repartidas uniformemente entre los jugadores; $|M| = 5\,000$ (100 páginas), de los cuales 1 000 cumplen $date \geq$ 2020-01-01. Cada fila de PM referencia exactamente un jugador y un partido, repartidos uniformemente. El *tamaño* de un plan es la suma de las cardinalidades de los resultados de sus *joins*.

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

(b) Calcule el tamaño de los planes de join

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

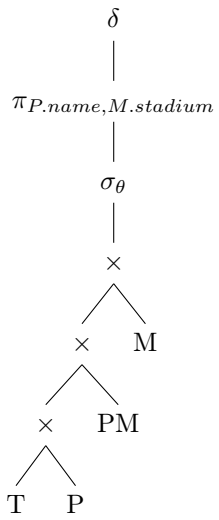
donde T', PM' y M' son las relaciones ya filtradas por sus condiciones. ¿Cuál es el mejor y por qué?

(c) ¿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.

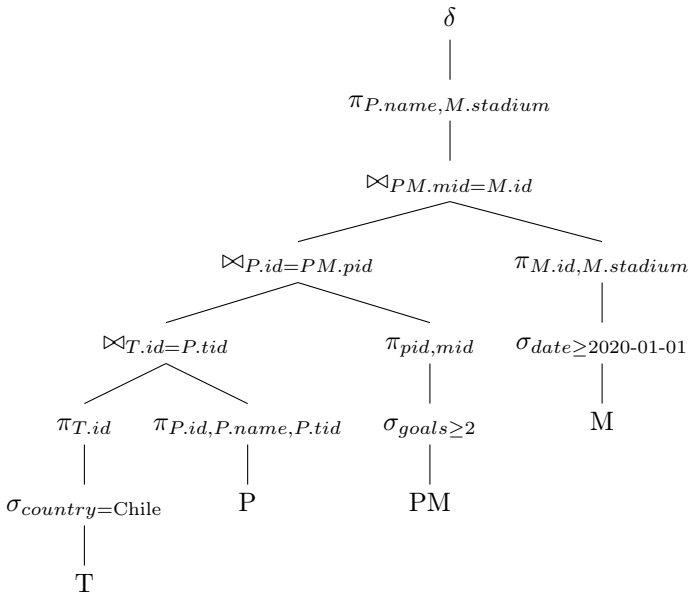
(d) Para elegir cómo leer cada relación filtrada, use el modelo de costos de la Pregunta 3 (full scan = páginas; hash: $\lceil t/200 \rceil + t$; B+tree unclustered: $2 + \lceil t/200 \rceil + t$; B+tree clustered: $2 + \lceil t/\text{tuplas por página} \rceil$). El catálogo indica: un hash sobre Teams.country, 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.

Pauta.

(a) *Plan directo:* $\pi_{P.name,M.stadium}$ sobre σ_θ (todas las condiciones) sobre $((T \times P) \times PM) \times M$, con un δ en la raíz:



Plan reescrito: cada selección baja hasta el nodo más bajo que contiene todos sus atributos (*country* a T, *goals* a PM, *date* a M; regla A para partir la conjunción y B para bajarla), los productos con su igualdad pasan a equijoins (regla D) y se proyecta en cada hoja lo que se usará más arriba (regla PP): $T.id$, $(P.id, P.name, P.tid)$, (pid, mid) y $(M.id, M.stadium)$.



(b) T', PM' y M' son las hojas ya filtradas y proyectadas (las π no cambian el número de filas; P no tiene selección). Cardinalidades base: $|T'| = 12$, $|PM'| = 15\,000$, $|M'| = 1\,000$ (20 % de los partidos).

- A: $T' \bowtie P$ tiene $12 \cdot 30 = 360$ filas (cada equipo tiene $9\,000/300 = 30$ jugadores). Cada jugador tiene $15\,000/9\,000 = 5/3$ filas en PM', así que $\bowtie PM'$ da $360 \cdot 5/3 = 600$. Cada fila de PM' cae en M' con probabilidad 20 %: $\bowtie M'$ da $600 \cdot 0{,}2 = 120$. Tamaño: $360 + 600 + 120 = \mathbf{1\,080}$.
- B: primero $PM' \bowtie M'$ da $15\,000 \cdot 0{,}2 = 3\,000$; $P \bowtie (\cdot)$ conserva 3 000 (cada fila referencia un jugador); $T' \bowtie (\cdot)$ deja la fracción de jugadores de equipos chilenos ($12/300 = 4\,\%$): $3\,000 \cdot 0{,}04 = 120$. Tamaño: $3\,000 + 3\,000 + 120 = \mathbf{6\,120}$.
- C: $T' \bowtie P$ da 360; $PM' \bowtie M'$ da 3 000; el join final deja $3\,000 \cdot 0{,}04 = 120$. Tamaño: $360 + 3\,000 + 120 = \mathbf{3\,480}$.

Los tres planes hacen los mismos tres joins; solo cambia cómo se asocian (y, en B, la conmutatividad para escribirlo con las hojas en el orden T', P, PM', M'). El mejor es A: empieza por la relación más selectiva (T', 12 filas) y arrastra resultados pequeños; B y C forman primero $PM' \bowtie M'$, que es grande. Es la heurística de elegir el orden de joins por conmutatividad y asociatividad.

(c) **Sí es necesario** para `DISTINCT P.name, M.stadium`: puede haber dos jugadores chilenos distintos con el mismo nombre que anotaron al menos dos goles en un mismo estadio desde 2020, o un mismo jugador en dos partidos del mismo estadio; en ambos casos la fila $(name, stadium)$ aparece más de una vez sin el δ. Para `DISTINCT P.id, M.id` **no es necesario**: cada fila del resultado proviene de una fila de PM (la combinación (pid, mid) es su llave primaria, y P, T y M se unen por sus llaves, es decir, muchos-a-uno hacia PM), y el par $(P.id, M.id) = (pid, mid)$ identifica esa fila. Luego ninguna fila del resultado se repite y $\delta(R) = R$.

(d)

Selección	Índice del catálogo	Con índice	Full scan	Decisión
$\sigma_{country=\text{Chile}}(T)$ $(t = 12)$	hash	$\lceil 12/200 \rceil + 12 = 13$	10	**full scan**
$\sigma_{goals \geq 2}(PM)$ $(t = 15\,000)$	B+tree unclustered	$2 + 75 + 15\,000 = 15\,077$	3 000	**full scan**
$\sigma_{date \geq 2020}(M)$ $(t = 1\,000)$	B+tree clustered	$2 + \lceil 1\,000/50 \rceil = 22$	100	**usar el índice**

Solo el índice clustered ayuda. El hash sobre *country* es inútil porque T es tan chica (10 páginas) que leerla toda cuesta menos que ir a buscar las 12 tuplas una por una; el unclustered sobre *goals* pierde porque devuelve el 10 % de PM (mucho más del $\approx$ 1–2 % que justifica un índice unclustered).