OPTIMITZACIÓ FÍSICA

Resultats intermitjos, Join

Aquesta sessió i la següent cobreixen el procés d'optimització física començant per un arbre sintàctic optimitzat i acabant amb un pla d'accés per executar la consulta tenint en compte els índexos disponibles i altres paràmetres del sistema. La part principal del procés estima el cost de les operacions de l'arbre (seleccions, joins, ...) fent servir, entre d'altres, les fórmules de cost de la sessió d'estructures d'accés i les de la sessió d'índexos avançats. Repasseu, si us plau, la introducció i l'apartat "Physical Optimization" del document "Additional Material: Semantic and Syntactic Optimization (Explanation)" de l'autoestudi sobre optimització semàntica i sintàctica. Mireu-vos-ho juntament amb les 10 primeres diapositives d'aquesta sessió i el document Exemple arbre procés que hi ha en el bloc d'aquesta sessió en el LearnSQL.

Resultats intermitjos. Un cop hem identificat tots els arbres de procés que hem de considerar, fem una estimació del millor cost de cadascun. Però com que el cost d'un arbre és la suma de costos de cada node de l'arbre i el cost d'un node depèn, entre altres coses, del volum de la(es) entrada(es) del node, el primer que fem és una estimació del volum de les dades que circulen per l'abre de les fulles (taules/índexos) fins a l'arrel (resultat de la consuta).

L'estimació de volums la farem amb un càlcul de probabilitat. Donada una condició (de selecció, per exemple) i coneixent determinades informacions sobre la BD (quantes files té cada taula, quants valors diferents hi ha en cada columna, etc.) establirem la probabilitat que una fila compleixi la condició i, amb aquesta probabilitat i sabent les files que entren a una operació (cardinalitat de l'entrada) calcularem una estimació de quantes files sortiran de l'operació (cardinalitat de la sortida). A la diapositiva 11 hi teniu una ilustració d'aquesta idea, on |X| vol dir cardinalitat d'X. En aquesta mateixa diapositiva hi teniu com s'han de calcular les cardinalitats de la sortida de les operacions que ens podem trobar en un arbre de procés. Fixeu-vos que:

- Podem començar per les fulles amb la suposició que sabem quantes files hi ha a cada taula i anar calculant la cardinalitat dels pares de les fulles i, a continuació, la dels nodes dels nivells superiors donat que la cardinalitat de l'entrada d'un node d'un nivell és la cardinalitat de la sortida del seu(s) fill(s). La cardinalitat de la rel és el nombre de tuples retornat per la consulta.
- Tot queda en funció del factor de selectivitat (SF, selectivity factor) de la selecció i de la JOIN. A les diapositives següents veiem quin és aquest factor de selectivitat que dependrà, naturalment, de la condició de seleció/JOIN i, com hem dit, de dades com ara la cardinalitat de les taules i sobre els valors dels atributs. Aquestes informacions es coneixen com a estadístiques de la BD i es

van actualizant automàticament o a demanda de l'usuari. Les diapositices 12 i 13 les presenten.

A les diapositives 14 i 15 hi veiem quin és el factor de selectivitat de la selecció segons com sigui la condició. En aquestes diapositives "A" es refereix a un atribut, "c" a una constant i "v" és una variable (per exemple, un altre atribut). "ndist" (nombre de valors diferents d'una columna), "max" (valor màxim d'una columna) i "min" (valor mínim d'una columna) són estadístiques conegudes per l'SGBD. Podeu veure exemples d'aplicació d'aquestes fórmules al document Additional Material: Example of Physical Optimization. Mireu a la primera pàgina les estadístiques de l'exemple i a la segona la consulta a optimitzar i els dos arbres de procés que es generen i dels que cal estimar el cost per triar el millor. A la pàgina 3 es calculen les cardinalitats de les seleccions (que són comunes als dos arbres, PT1 - process tree 1 - i PT2). De moment, fixeu-vos només en les línies SF(...) = ... i |V'| = ... i |P'| = ...

Les diapositives 16 i 17 estan dedicades al FS de la JOIN. La idea és que el FS sigui la probabilitat que una tupla del producte cartesià de les dues relacions compleixi la condició de JOIN. D'aquesta manera, la cardinalitat de la JOIN de R i S serà |R|*|S|*FS. A la diapositiva 16 hi ha alguns casos que no corresponen a JOIN a través de clau forana, on R i S són relacions, A i B són els atributs de JOIN i el primer cas representa el producte cartesià. A la diapositiva 17 hi ha els casos en què fem JOIN a través de clau forana i clau primària referenciada:

- En el primer cas, es fa directament la JOIN, sense seleccions prèvies. Fixeu-vos que un exemple podria ser (on departaments seria R, empleats seria S, num_dept seria B i num seria A)

  ```
  SELECT * FROM departaments D, empleats E WHERE E.num_dept = D.num
  ```

  Sabem que en aquesta consulta obtindrem tantes tuples com empleats (cada empleat fara JOIN amb un i només un departament). D'aquí que la cardinalitat de la JOIN, que hem dit que es calcula com |R|*|S|*FS, ha de ser |S| i, en conseqüència, el FS ha de ser 1/|R|.c
- En el segon cas fem una selecció prèvia sobre la relació R (la que té la clau primària). Un cas podria ser

  ```
  SELECT * FROM departaments D, empleats E
  WHERE E.num_dept = D.num AND D.ciutat = 'St Esteve de les Roures'
  ```

  Sabem que en aquesta consulta hem d'obtenir els empleats de departaments que són en aquella ciutat. Sota la hipòtesi de distribució uniforme de valors als atributs, hi ha el mateix nombre d'empleats a cada departament. De manera que la proporció d'empleats de St. Esteve ha de ser igual a la proporció de departaments de St. Esteve. Per tant, ara la cardinalitat de la JOIN ha de ser |S|*sfr si li diem *sfr* a la proporció de tuples de R que han superat la selecció (o sigui, el factor de selectivitat de la selecció). Si R' és el resultat de la selecció, la

cardinalitat de la JOIN és |R'|*|S|*FS, que acabem de dir que ha de ser |S|*sfr. Per tant, FS = sfr/|R'|

- En el tercer cas fem una selecció prèvia sobre la relació S (la que té la clau forana). Per exemple

```
SELECT * FROM departaments D, empleats E
WHERE E.num_dept = D.num AND E.edat = 30
```

Sabem que en aquesta consulta hem d'obtenir els empleats que tenen aquella edat combinats cadascun amb el seu únic departament. La cardinalitat de la JOIN ha de ser la mateixa que la de la selecció, o sigui |S'|. La cardinalitat de la JOIN és |R|*|S'|*FS, que acabem de dir que ha de ser |S'| i, ara, FS = 1/|R|.

A les pàgines 4 i 5 de l'exemple d'optimització física que ja hem esmentat abans hi teniu exemples d'aplicació d'aquestes fórmules:

- La JOIN entre W i V' és del tercer tipus (s'ha fet selecció prèvia a la taula que té la clau forana).
- La que es fa entre WV' i P' és del segon tipus (s'ha fet selecció prèvia a la taula que té la clau primària).
- La que es fa entre V' i P' és com l'anterior.
- La que es fa entre W i V'P' és com la primera.

Acabem de veure, doncs, com calcular les tuples que surten de cada node però també ens interessa saber quants blocs de disc ocupa la sortida de cada nodes (perquè assumim que després d'executar una operació de l'arbre de procés se n'escriu el resultat en una taula temporal, que serà llegida quan s'executi l'operació del node pare). La diapositiva 18 ens diu com calcular el nombre de blocs a partir de la cardinalitat i la mida dels blocs:

- Mida de la fila: sumem l'espai en bytes que ocupen els atributs que formen part de la sortida (per simplificar, suposem que els atributs són de mida fixa). Eventualment, podem tenir en compte l'espai que ocupi la informació de control de la capçalera de les files.
- Nombre de files que cabran a cada bloc. Dividim la mida en bytes dels blocs per la mida que acabem de calcular, truncant el resultat.
- Finalment, els nombre de blocs necessaris el calculem dividint la cardinalitat de la sortida pel nombre de files per bloc que acabem de calcular. Ara, arrodonim per excés el nombre de blocs.

Ara podeu tornar a mirar les pàgines 3, 4 i 5 de l'exemple d'optimització física i veure com s'hi calculen els blocs necessaris per guardar els resultats de les sleccions i joins. Per exemple, a la pàgina 3, la selecció sobre V, que hem anomenat V', té una cardinalitat de 81632 tuples. Aquestes tuples contenen els atributs *prodId* i *wineId*, que participen a les joins posteriors. Com que aquests atributs ocupen 5 bytes cadascun, les tuples ocupen 10 bytes. Sabem (és una dada que hem de conèixer i la tenim a la pàgina

1) que els blocs de resultats intermitjos són de 500 bytes. Aleshores a cada bloc hi caben 500/10=50 tuples i, per tant, necessitem 81632/50=1633 blocs. La resta de volums es calculen igual.

Un cop coneguts els volums dels resultats intermitjos, passem a calcular el cost de les operacions de cada node segons quina sigui aquesta operació i l'algoritme i estructures d'accés que fem servir. Les operacions que tractarem són selecció, join, ordenació i projecció. A la resta d'aquesta sessió ens ocupem de la join i les altres operacions són el tema de la sessió següent. Per cost entenem el nombre d'accessos a disc que cal fer.

Join. Estudiarem quatre algoritmes de join però, abans, analitzem una disposició de les taules que anomenem estructura cluster dissenyada per fer fàcil la join. La podeu veure a la diapositiva 21 i consisteix en emmagatzemar barrejades les files de dues (o més) taules. El criteri per ubicar les files és posar juntes aquelles que tenen el mateix valor en l'atribut de join, siguin de la taula que siguin. Per poder fer insercions sense provocar desplaçaments, es deixen espais buits en els blocs com ja vam veure que es fa en taules ordenades (en índex cluster). A l'exemple de la diapositiva s'entén que els empleats 14, 9 i 6 tenen el número de departament 1, l'empleat 3 té el número de departament 2, etc. Amb aquesta disposició, la join costaria com els scan individuals de les taules: $1.5(B_R + B_S)$.

Els dos primers algoritmes apliquen una solució intuïtiva: per fer la join de R i S es recorren les files de R i, per a cada fila de la taula R, cerquem les files de la taula S que hi combinen:

- En l'algoritme Row Nested Loops (diapositiva 22), aquesta cerca de les files de S es fa a través d'un índex. Vegeu els costos en funció de si és o no semi-join i de l'índex utilitzat. Un exemple de semi-join (on R és la taula de departaments i S la d'empleats):

  ```
  SELECT D.num_dept, D.nom_dept
  FROM departaments D, empleats E
  WHERE E.num_dept = D.num
  ```

  Si fem aquesta join amb l'algoritme RNL agafant *departaments* com a taula del bucle extern, per cada departament cercarem a l'índex de *empleats* els que tenen aquell número de departament, però no accedirem a la taula d'empleats. En canvi, si la consulta és:

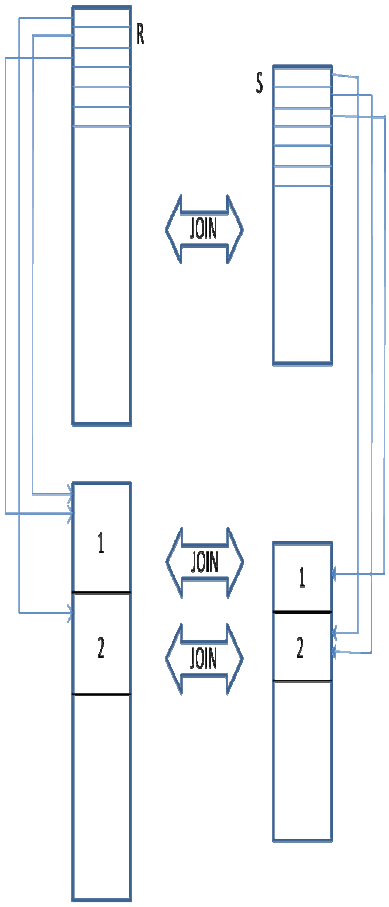
  ```
  SELECT D.num_dept, D.nom_dept, E.num_empl
  FROM departaments D, empleats E
  WHERE E.num_dept = D.num
  ```

  per cada entrada que trobem a l'índex de *empleats* farem un accés a la taula *empleats*.
  En tots els casos, el cost és $B_R + |R|$*(cost de cercar les tuples de S a través d'un índex), on el cost de cercar diverses tuples a través l'índex és el que vam veure a la sessió anterior.

- L'algoritme Block Nested Loops, en canvi, cerca les files de S seqüencialment sense fer servir cap índex. Si aquest algoritme llegís les taules bloc a bloc, faria un total de B_R*B_S accesos però si fem servir un buffer de blocs per a llegir uns quants blocs de cop d'una de les taules aconseguim un cost millor. A la diapositiva 23 hi teniu l'algoritme BNL i el cost: la taula R es llegeix una sola vegada (amb les lectures que calgui de M blocs cada vegada, que són B_R/M) i, per cada paquet de M blocs de R, es llegeix tota la taula S, de manera que al final la taula S s'haurà llegit B_R/M vegades. A més d'aquestes M pàgines de memòria per llegir R, en necessitem una per llegir S i una altra com a buffer de sortida. Les diferents versions de la diapositiva 24 mostren dues execucions de l'algoritme: a l'esquerra una execució on la taula R és la que té més blocs i a la dreta la taula S és la que té més blocs. Es veu l'evolució de l'algoritme i s'afegeix una lletra "R" (de read, lectura) per cada accés. Fixeu-vos que se'n fan menys si la taula més petita és la del bucle extern (la taula R). De fet, el cas millor d'aquest algoritme el tenim quan la taula més petita cap sencera a memòria. En aquest cas, el cost és B_R+B_S perquè llegim una sola vegada cada taula.

L'algoritme se Sort-Match també es basa en una idea força utilitzada, el recorregut simultani de dues seqüències ordenades. A la diapositiva 25 hi teniu l'algoritme per fer una equi-join: començant amb les dues taules ordenades per l'atribut de join, comparem les dues primeres files i, si combinen, posem el resultat a la sortida i passem a la següent fila en totes dues taules; si no combinen és que una (diguem que és S) té el valor més petit que l'altra. Podem assegurar que aquesta fila no combinarà amb cap altra de R perquè les que queden per explorar de R tenen el valor major o igual que el de la fila de S; per això passem a la següent fila de S. Queda clar, doncs, que només anem avançant i que, si les taules R i S són ordenades, el cost serà B_R+B_S. Generalment, però, caldrà ordenar un o totes dues taules, cosa que es pot fer (com veurem a la sessió següent) amb un cost de $2*B*\log_M(B)$, on M+1 són les pàgines de memòria disponible per fer l'ordenació i B els blocs ocupats per la taula. La diapositiva 26 ilustra el procés de fusió de les dues taules prèviament ordenades.

Per la seva banda, l'algoritme de Hash-Join es val d'una funció de hash per reduir una join de taules grans a un cert nombre de joins de taules més petites. Es fa servir la mateixa funció de hash per reubicar les files de les dues taules enviant cada fila a la part de la nova taula que indica la funció de hash aplicada al valor de l'atribut de join en aquella fila. Això tindrà com a conseqüència que si una fila d'una taula ha quedat en una part diferent que una fila de l'altra taula, segur que tenen valors diferents a l'atribut i, per tant, segur que no combinen. O, el que és el mateix, una fila d'una taula només pot combinar amb les files de l'altra taula que han quedat a la meteixa part, La figura següent il.lustra aquesta idea:



La gràcia de fer aquesta reubicació està en aconseguir que les parts de la taula més petita càpiguen senceres a memòria per poder fer cadascuna de les joins fent servir l'algoritme BNL que, en aquestes circumstàncies, es pot fer llegint una sola vegada cada part. Si tenim memòria suficient, aquesta reubicació de les files de la taula també es pot fer fent una única lectura de les taules originals. Així, doncs, en total fem una lectura de cada taula, una reescriptura de cada taula reubicant files i una segona lectura de cada taula per fer les joins. A la diapositiva 27 hi veieu que es dissenya la funció de hash per tal que doni un nombre entre 1 i p prenent com a p l'arrodoniment per excés de B_S/M (si S és la taula petita). També hi podeu veure que l'algoritme té el cost que us acabo d'explicar si $B_S <= M^2 + M$. No entrem en el detall de demostrar que aquesta condició i la tria de p que hem fet garanteixen tant que es pot fer la reubicació amb una sola lectura com que les parts de la taula petita cabaran a memòria.

La diapositiva 28 mostra una execució de l'algoritme hash-join amb una funció que retorna dos valors (classifica en parells - Even- i senars -Odd-). Les primeres versions mostren com es classifiquen les files de la taula de l'esquerra (des que comencen a entrar dades a la memòria fins que queda buida). Després es fa el mateix per a la taula de la dreta (fins que torna a quedar la memòria buida). Després es fa la join de les parts O i, finalment, la join de les parts E.

A les pàgines 8-11 de l'exemple d'optimització física s'hi càlculen costos de joins segons el que acabem de veure. El primer càlcul és el de la join entre W i V'. A la pàgina 1 se'ns diu quins algoritmes hem de tenir en compte i amb quantes pàgines de memòria; també se'ns diu quins índexos hi ha definits. Calculem els costos, però, de les fulles cap a la rel. Primer teniu a la pàgina 7 el càlcul dels costos de les seleccions fent servir les fórmules de la sessió anterior (estructures d'accés).

La join entre W i V' és per W.wineId = V'.wineIb. W té un índex cluster per wineId, la qual cosa vol dir que, a més, està ordenada per wineId. V', resultat de la selecció sobre V, no està indexada (els resultats intermitjos no ho estan), però com que la selecció s'ha fet amb un scan i V estava ordenada per wineId, V' també ho està.

- El càlcul del cost amb BNL té en compte que, com que la taula W té un índex cluster, ocupa 1.5 l'espai que ocuparia sense cluster. Pel que fa als blocs que ocupa V', els hem calculat a la fase anterior d'estimació de volums.

- RNL només el podem aplicar fent que W sigui la taula interior atès que V' no té índex. Hem de veure, d'altra banda, que necessitem accedir a les files de W per recuperar l'atribut strength. La cardinalitat de V' l'hem calculada abans a la fase d'estimació de volums. I sabent que l'ordre del arbres és 75 (aquesta informació és també a la primera pàgina), hi ha 2/3 * 2*75 = 100 entrades per node en els arbres. Finalment, k (nombre de repeticions = nombre de wines que combinen amb cada vintage) és 1.

- El càlcul del cost del sort-match ha de tenir en compte que no cal ordenar cap de les dues entrades.

La join entre WV' i P' no pot fer servir RNL perquè cap de les dues entrades té índex. El mateix passa amb V' i P'.

Per acabar, la pàgina 12 de l'exemple té un resum dels costos de tots dos arbres de procés.