[CBDE] 3r LAB: Scrutinizing the Internals of Column-Oriented DBs

Laia Ondoño i Laia Igelmo Amorós

Novembre 2020

1 Question 1

For each question in the 3rd Lab SQL quiz:

1. Explain what structures you created:
   (a) Question 1:
   Inicialment, vam descartar crear bitmaps pels atributs de les queries perquè no els generàvem correctament i ens donava un cost massa elevat. En adonar-nos, però, vam veure que tenien un cost molt més baix en comparació amb els arbres B+.

   Per tal de no deixar cap atribut sense índex, vam generar 4 bitmaps, un per a cada atribut. Tot i això, vam veure que agrupar els atributs cand i val en un mateix bitmap comportava una millora respecte a l'anterior, tenint en compte que la freqüència d'ús d'aquest bitmap seria del 80% de les execucions (Q2 i Q3).

   Paral·lelament i tenint en compte que la query 3 representa el 50% de les execucions totals, vam pensar que seria bo afegir una vista materialitzada d'aquesta query, d'aquesta manera podem accedir directament a les dades anant al disc i, això ens permet no haver d'executar cada vegada la resolució de la query.

   Vam crear les següents estructures:

   - 3 bitmaps:
     - Un d'ells respecte pobl
     - Un altre respecte l'edat
     - I, finalment, un tercer respecte cand i val
   - Una materialized view respecte la Query 3

   Provant diferents estructures vam veure que aquesta era la més eficient, donava 9,3 i ocupava 264 blocs. Per tant, entrava dins del límit de blocs de l'exercici.

Tanmateix no vam comprimir els bitmap index amb "ALTER TABLE poll˙answers MINIMIZE RECORDS˙PER˙BLOCK; " i no vam aconseguir la nota màxima. De totes maneres, considerarem en aquesta explicació el resultat amb els bitmaps comprimits.

(b) Question 2:

Vam començar fent l'algoristme de l'affinity matrix per tal d'obtenir la millor fragmentació vertical. Però vam veure que a Oracle, al ser row-oriented, no funciona com hauria amb la millor fragmentació vertical perqué no està preparada per a fer-ho.

Així que vam haver de buscar una altra fragmentació vertical que tingués un cost més baix.

La millor opció que hem trobat és crear un fragment vertical que inclogui tots els atributs que es fan servir a les queries (pobl, edat, cand, val) i una altra fragmentació que uneix tots els atributs restants (les respostes) exceptuant el ref.

Un cop vam aconseguir la millor fragmentació vertical per a Oracle, vam mirar quina era la millor combinació d'indexs. La millor combinació que hem trobat és afegir 3 bitmap indexs:

- Un bitmap per a pobl.
- Un bitmap per a edat.
- Un ultim bitmap per a cand i val.

El resultat que ens ha donat aquesta solucio de cost és de 94,3. Encara que el cost real, mirant el pla d'execució, és de 93.

2. Explain Oracle's execution plan considering the structures you created

(a) Question 1:

A la primera Query, com era d'esperar, només fa servir les bitmap indexes 1 i 2, que son els corresponents a pobl (indexbitmap1) i edat (indexbitmap2).

Un cop ha fet un scaneig de tot el bitmap (perque no hi ha rangs ni condicions de parada ho mira tot), ho converteix a rowids per despres utilitzar-ho per a poder accedir a la taula (ja que Oracle és row oriented).

Quan ha convertit ambdos bitmaps en rowids, fa join dels dos sets.

Oracle fa una vista de la join i tot seguit fa el group by per pobl de la query. Agafar el bitmap, convertir-ho a rowid i després fer la join surt a compte pel tipus d'operació, ja que fa un group by més endavant.

Finalment fa el Select.

Podem veure l'execution plan utilitzant el rastrejador automàtic de la primera query a la figura 1:

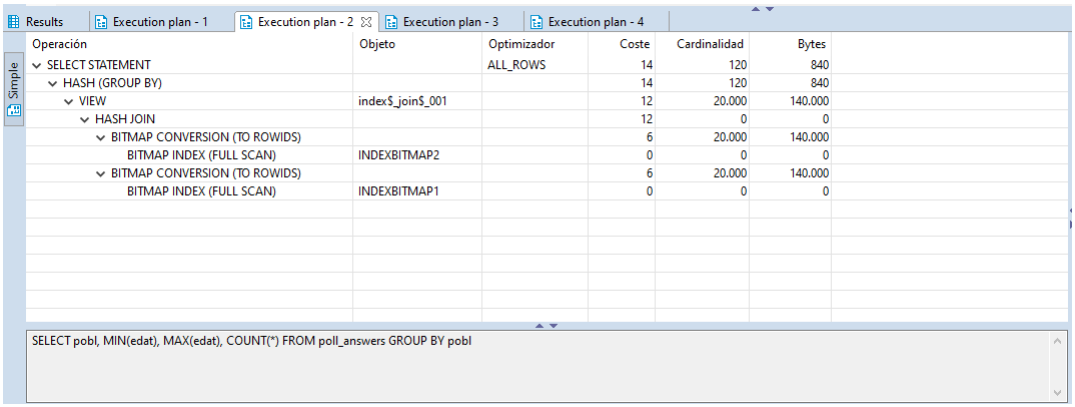

Figure 1: Execution plan de la Query 1

A la segona Query veiem com, primer de tot, es fa un escaneig de tot el bitmap de edat i de (cand,val), corresponents a indexbitmap2 i indexbitmap3 respectivament.

Un cop ha convertit la representació del bitmap de edat i de (cand, val) a un set de rowids. Fa el join d'aquestes dos. A aquest join ens referirem a ell com a join(edat, cand, val)

Tot seguit, fa l'escaneig del bitmap de pobl, corresponent a l'index indexbitmap1 i, com amb els altres dos anteriors, converteix la representació del bitmap a un set de rowids.

Després d'això, fa join de el join anterior "join(edat, cand, val)" amb el set de rowids de pobl. i fa la vista del resultat per després fer el group by de pobl, edat i cand.

Igual que amb la query 1, agafar el bitmap, convertir-ho a rowid i després fer la join surt a compte pel tipus d'operació, ja que fa un group by més endavant.

Finalment fa el Select.

Podem veure l'execution plan utilitzant el rastrejador automàtic de la segona query a la figura 2:

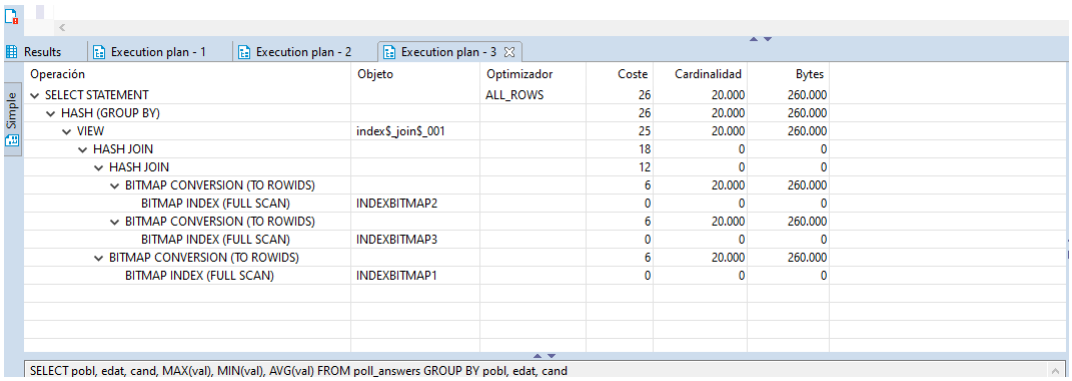

Figure 2: Execution plan de la Query 2

En la query 3 podem veure com, al haver fet una materialized view just d'aquella query anomenada view1, l'unic que s'ha de fer és analitzar la materialized view i fer el Select, això comporta un cost molt petit.

Podem veure l'execution plan utilitzant el rastrejador automàtic de la tercera query a la figura 3:

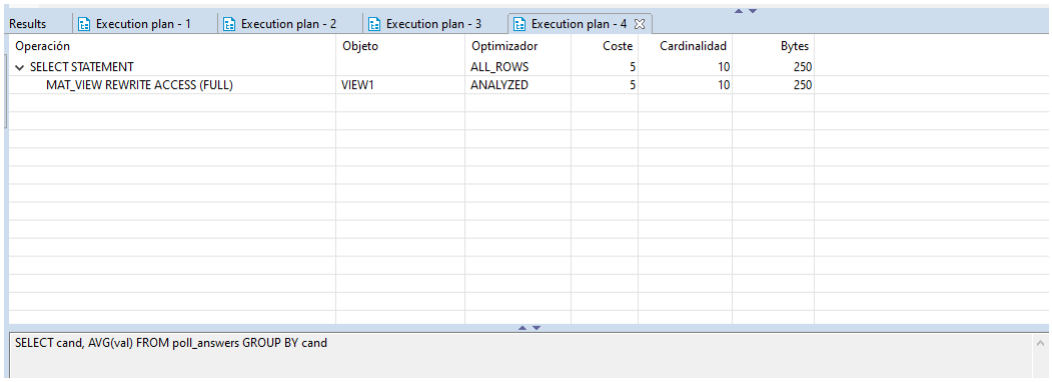

Figure 3: Execution plan de la Query 3

(b) Question 2:

A la primera Query, veiem com l'execution plan entre utilitzar nested tables (mirar figura 4) i sense utilitzar-ne (mirar figura 1) són casi idèntics. Es a dir, el procediment de Oracle amb nested tables és com si ho fes row-oriented. Però cal afegir un join per tal de poder unir tots els fragments verticals dels bitmaps. Cosa que fa amb l'index fast full scan.

També podem destacar com en l'execution plan de la query 1 de la Question 1 el cost és de 14, mentres que amb nested tables és de 93 (veiem com el index fast full scan és el culpable d'aquest cost tant elevat).

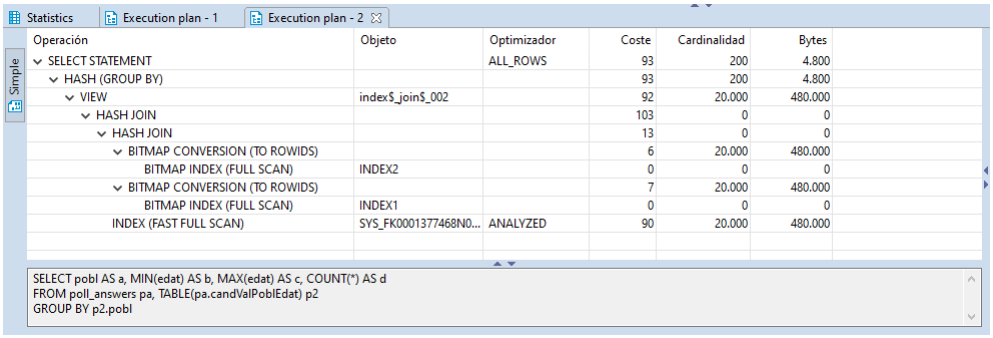

Figure 4: Execution plan de la Query 1

A la query 2, si comparem l'execution plan de la Question 1 (figura 2) i la de la Question 2 (figura 5), veiem que passa el mateix explicat anteriorment a la primera query:

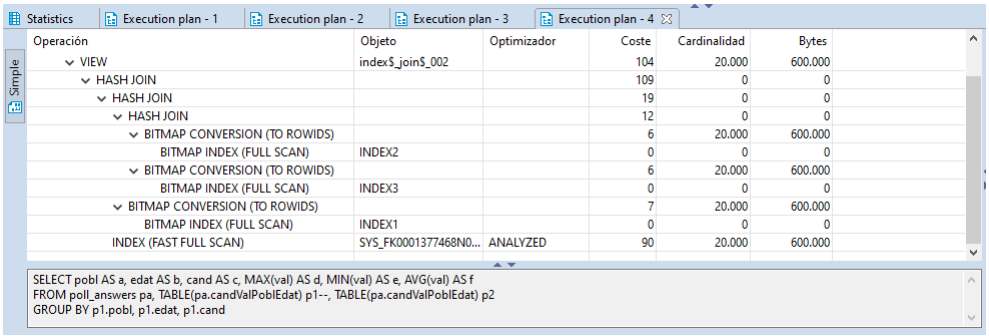

Figure 5: Execution plan de la Query 2

A la query 3, com no podem utilitzar materialized views, Oracle ha d'utilitzar primerament el bitmap index de cand i val per a, tot seguit

fer el join amb l'index fast full scan.

Tot seguit fer la view del resultat, sobre la qual es farà el group by de cand.

Finalment, fa el Select.

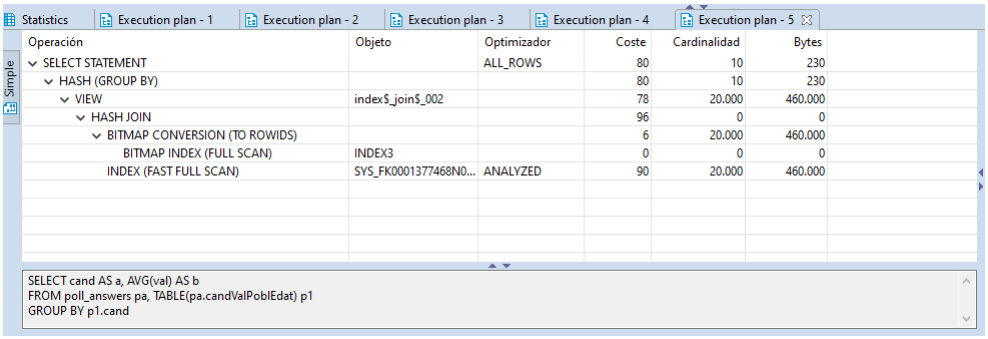

Figure 6: Execution plan de la Query 3

3. For the execution plan given, discuss what steps would be executed differently by a column-oriented databases (refer to the physical data structures as well as query processing techniques used)

   (a) Question 1: Tot el que hem vist a l'exercici anterior respecte la "Question 1" canviaria en gran part en cas de column oriented:

   - Amb la Query 1:
     Per començar, si fos una column-oriented en aquest cas, no s'hauria de fer join de població i edat, ja que no faria falta reconstruïr la taula per conseguir fer el select. Agafaria la taula d'edat i aniria mirant els IDs, d'allà treuria directament la sol·lució, no faria falta late materialization. Fent-ho tot molt més eficient.
   - Amb la Query 2:
     Amb la query 2 passa el mateix que amb la query 1, encara que sigui una query més complexa el tractament és el mateix. No faria falta fer un late materialization per a poder fer el Select, amb les referències tindríem tota la informació necessària.
   - Amb la Query 3:
     Al tenir una vista materialitzada de la query 3, aquesta solució ja és molt eficient de per si.
     Si ho féssim amb una column-oriented, possiblement seria més costos, ja que s'ha de fer el procediment comentat en les altres dos queries anteriors. Es a dir, no hi ha materialització de la taula, però de totes maneres s'ha de recórrer la taula amb les referències per respondre a la query.

(b) Question 2: En aquesta comparació veiem com la fragmentació vertical d'Oracle no implementa bé l'algorisme de l'affinity matrix perque aquest esta pensat per a bases de dades column-oriented i Oracle és row-oriented cosa que fa que actui de manera poc eficient.
L'excecució en column-oriented a les tres queries seria recorrer les estructures generades i agafar la informació de manera directa sense haver de materialitzar cap taula.
L'execució de les nested tables és molt semblant a l'execució fent servir l'estructura de la Question 1, inclús és més ineficient perqué fa un join extra per unir les taules resultants del bitmap amb l'index fast full scan.

2 Question 2

For the second exercise, use the affinity matrix method to decide how to fragment the database vertically. Consider now your solution for Exercise 2 in the 3rd Lab SQL quiz. Is Oracle yielding the best result when using the same vertical fragmentation strategy as suggested by the affinity matrix method? Justify your answer.

Primer de tot, cal que obtinguem l'affinity matrix per tal de decidir com fragmentar verticalment la base de dades.

Comencem per fer el attribute usage matrix, si mirem a la figura 7:

	ref	pobl	edat	cand	val	resposta_i i = 1, 2, 3, 4, 5
Q1 (20%)	0	**1**	**1**	0	0	0
Q2 (30%)	0	**1**	**1**	**1**	**1**	0
Q3 (50%)	0	0	0	**1**	**1**	0

Figure 7: Attribute usage matrix

Veiem com tant a la query 1 com a la 2, que representen un 50% de les execucions, s'utilitzen els atributs pobl i edat.
També podem veure com a la query 2 i 3, que representen un 80% de les execucions, s'utilitzen els atributs cand i val.
Finalment veiem com a la query 2 coincideixen els 4 atributs nombrats anteriorment (pobl, edat, cand i val), que constitueix un 30% de les execucions.

Aquesta informació la podem convertir en la nostra attribute affinity matrix, tal i com veiem a la figura 2:

	ref	pobl	edat	cand	val	resposta_i i = 1, 2, 3, 4, 5
ref	0	0	0	0	0	0
pobl	0	**50**	**50**	**30**	**30**	0
edat	0	**50**	**50**	**30**	**30**	0
cand	0	**30**	**30**	**80**	**80**	0
val	0	**30**	**30**	**80**	**80**	0
resposta_i i=1, 2, 3, 4, 5	0	0	0	0	0	0

Ara hem d'ordenar la matriu per tal de veure l'ordre del atributs per formar els clusters i veure quins atributs tenen més afinitats entre ells i quins clusters són els més crítics. A la figura 8 podem veure com queda la taula:

	cand	val	pobl	edat	ref	resposta_i i = 1, 2, 3, 4, 5
cand	**80**	**80**	**30**	**30**	0	0
cand	**80**	**80**	**30**	**30**	0	0
pobl	**30**	**30**	**50**	**50**	0	0
edat	**30**	**30**	**50**	**50**	0	0
ref	0	0	0	0	0	0
resposta_i i=1, 2, 3, 4, 5	0	0	0	0	0	0

Figure 8: Affinity matrix ordenada per afinitat i pes dels atributs.

Veiem com tant cand i val tenen una correlació molt alta d'un 80%, seguits per població i edat amb un 50%. Per tant veiem com tant cand i val haurien d'anar a una fragmentació i com pobl i edat n'haurien d'anar a un altra junts. Els altres atributs haurien d'anar per separat. Per tant, ens quedaria de la següent manera:

- Fragmentació 1: (cand, val).
- Fragmentació 2: (pobl, edat).
- Els altres atributs per separat.

Com s'ha comentat a l'apartat **??**, Oracle no implementa bé l'algorisme de l'affinity matrix perque aquest esta pensat per a bases de dades column-oriented i Oracle és row-oriented, cosa que fa que actuï de manera poc eficient.
Per això veiem com utilitzant nested tables i una simulada estructura column-oriented, el resultat és pitjor.