BD2 – 8/06/2015 – first part - solutions

Please feel free to answer your test in English, Italian, or any mixture

1) Consider the following query

SELECT R.B, S:C, count(*), Avg(S.D)
FROM R, S
WHERE R.IdS = S.IdS and 10 < R.A < 20
GROUP BY R.B, S.C
ORDER BY S.C

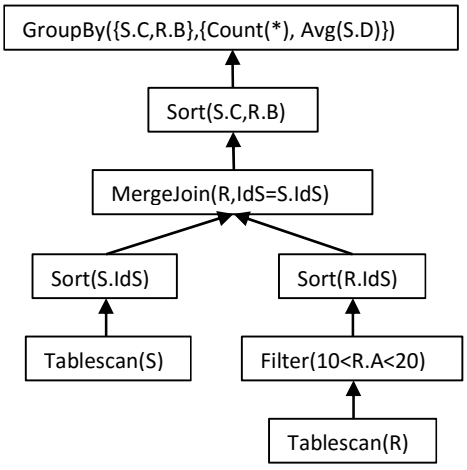
Assume that R and S are stored as heap files. Assume that S.IdS is primary key and R.IdS is a foreign key that refers to S.
Assume that unclustered RID-sorted indexes are defined on attributes R.IdS, R.A, S.IdS, S.C.
Assume the following table for the optimization parameters of tables R and S, and of indexes on R.A and S.C.
The size of indexes R.IdS and S.IdS can be computed by assuming that each leaf may contain 500 RID's and ignoring the space needed to contain the value of IdS.
If you need Cardenas formula Φ(n,k), approximate it with min(n,k).

	NReg	NPag	NLeaf	NKey	Min	Max
R	200000	4000				
S	10000	1000				
Idx.R.A			440	100	1	100
Idx.S.C			24	1000	1	1000

a) Draw a logical access plan for the query
b) Compute NLeaf for Idx.R.IdS and Idx.S.IdS

NLeaf Idx.R.IdS: 200.000/500 = 400
NLeaf Idx.S.IdS: 10.000/500 = 20

c) Draw an efficient access plan for the query that uses no indexes and compute its cost



For simplicity we inserted no projection operator, although that could reduce the cost of sort operators.

The elements of the plan that have a cost are tablescan and sort. We compute the cost of each. For each Sort we first compute the input size.

C(Tablescan(S)) = NPag(S) = 1000

C(Tablescan(R)) = NPag(R) = 4000

Sort(S.IdS): Input Size: 1000. C(Sort(S.IdS)) = 2*NPag(Input) = 2000

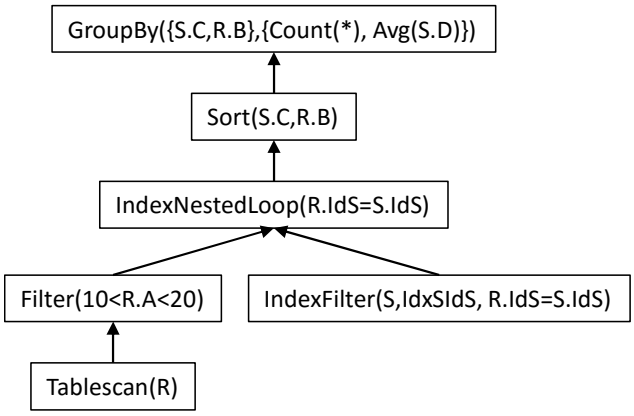
Sort(R.IdS): Input Size: sf(10<R.A<20)*4000= 10/100*4000 = 400 C(Sort(S.IdS)) = 2*NPag(Input) = 800

Sort(S.C, R.B): Input Size: sf(R,IdS=S.IdS)* sf(10<R.A<20)*NRec(R)*NRec(S)*(LRec(R)+LRec(S))/DPag
Records of R are quite smaller than records of S, hence we approximate LRec(R)+LRec(S) with LRec(S), hence we rewrite the above formula ash
NRec(R) * sf(R,IdS=S.IdS) * sf(10<R.A<20) *NRec(S)*LRec(S)/DPag
= NRec(R) * sf(R,IdS=S.IdS) * NPag(S) * sf(10<R.A<20)
= (200.000/10.000)*(1000/10) = 2.000
C(Sort(S.C, R.B)) = 4.000

Hence, the total cost of the plan is: 1000+4000+2000 + 800 + 4000 = 11.800

d) Compute the cost of an efficient access plan which is based on an IndexNestedLoop where R is the external relation, using all the indexes that are useful for this plan

This is a possible access plan:



We do not use an index to access R since the condition on 'R.A' is not selective enough.

The total cost is given by: C(TableScan(R)) + EReg(Filter(…))*(CI + CD(IndexFilter(…)) + Cost(Sort(…))
We already know that C(TableScan(R))+ Cost(Sort(…)) = 8000, we have to compute EReg(Filter(…))*(CI + CD(IndexFilter(…)) + Cost(Sort(…)).

EReg(Filter(…)) = sf(10<R.A<20)*NReg(R) = 1/10*200.000 = 20.000
CI(IndexFilter(…)) = sf(join condition)*NLeaf(IdxSIdS) = 1
CD(IndexFilter(…)) = sf(join condition)*NReg(S) = 1 – since IdS is key for S

Hence the total cost is 8000 + 20.000*2 = 48.000

e) Do you think that the plan in (c) is the most efficient plan for this query? Why?

Yes. Usually IndexNestedLoop is faster than MergeJoin, for a binary join, when at least one of the two relations is heavily restricted by a selection, and this restricted relation is used as the outer relation. In this case, the only restricted relation is R, but, as we have seen already, the restriction is not strict enough, since the selectivity factor is 1/10. If we used S as the outer relation the result would be even worse, since S is not restricted in any way. Hence MergeJoin (or, equivalently, HashJoin) is the best choice.

2) Consider the following query on R(IdR, A, B, IdS*) and S(IdS, C) , where keys and foreign keys are defined as in exercise (1), where "X" is one attribute from R or from S (for example: R.IdS, R.A, S.C…)

SELECT DISTINCT ***X***, count(*)
FROM R, S
WHERE R.IdS = S.IdS and 10 < R.A < 20 and R.B = 30
GROUP BY R.IdS, R.B, S.C
ORDER BY S.C

a) May X be R.A? may it be R.B? More generally, which attributes may be substituted to X in order to produce a well-formed SQL query?

The only attributes that may appear are R.IdS, R.B, S.C.

b) For each possible attribute that may be substituted to X, specify whether 'DISTINCT' is redundant or is necessary, explaining the answer

DISTINCT is redundant if, and only if, the closure of X includes all of R.IdS, R.B, S.C.

Closure of R.IdS = R.IdS, S.IdS, S.C, R.B : DISTINCT is redundant
Closure of R.B = R.B : DISTINCT is not redundant
Closure of S.C = S.C, R.B : DISTINCT is not redundant