Data Management – AA 2017/18 – exam of 06/02/2018

Solutions to selected problems

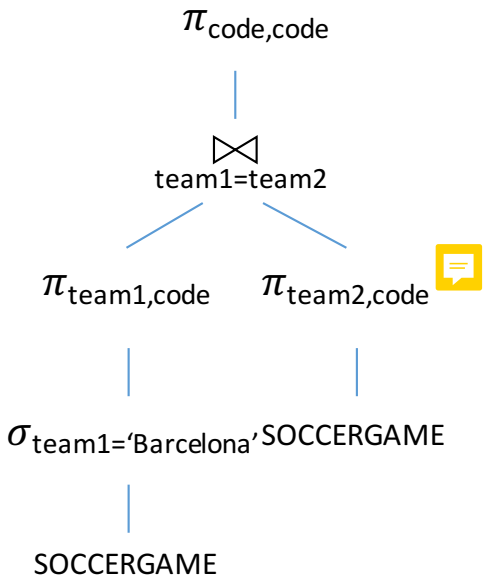
Problem 2
Consider the relation `SOCCERGAME(code,team1,team2,date,result)`, that for each soccer game stores information about its code, the teams participating in the game, the date and the result. The relation has 1.900.000 tuples, stored in 190.000 pages. We assume that we have 70 buffer frames available, that all fields and pointers have the same length, and that for every value of `team1` there are 100 games in the relation. Also, there is a nonclustering B^+-tree index on `SOCCERGAME` with search key `team1` using alternative 2. Consider the query

```
select s1.code, s2.code
from SOCCERGAME s1 join SOCCERGAME s2 on s1.team1 = s2.team2
where s1.team1 = 'Barcelona'
```

2.1 Describe the logical query plan associated to the query code, and illustrate both the logical and the physical query plan you would select, motivating the choices.

2.2 Tell which is the cost (in terms of number of page accesses) of executing the query according to the selected physical query plan.

Solution to problem 2
We do not show the the logical query plan associated to the query code because it is obvious. It is immediate to verify that pushing both selection and projection is beneficial. Thus, we obtain the following selected logical query plan:



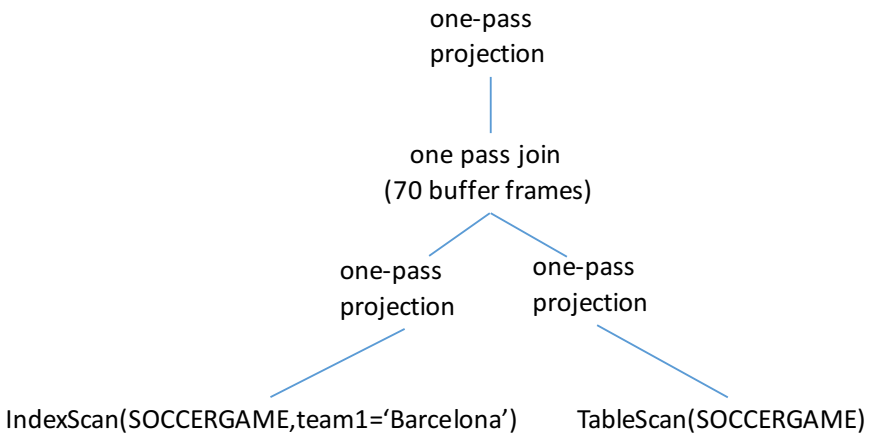
We now have to decide about the physical query plan. There are two decisions to take: how to perform the selection and the projection on `SOCCERGAME` for the left-hand side operand, and which algorithm to use for the projection of the right-hand side of `SOCCERGAME` and the join.

- As for the selection on `SOCCERGAME`, we obviously use the index to retrieve the 100 tuples corresponding to the 100 games involving "Barcelona". We can actually store directly the projection of these tuples on <`team1,code`> in the buffer, occupying 4 buffer frames (if 10 tuples with 5 attributes fit in one page, then 100 tuples of 2 attributes fit in 4 pages).

Since 10 tuples of SOCCERGAME fit in one page, every page has space for 25 data entries every page has space for 25 index entries. This means that we can consider a fan-out of 19, and, taking onto account the 67% occupancy rule, we can consider that every leave has 19 data entries. The index is nonclustering, and therefore dense, implying that we have 1.900.000/19 = 100.000 pages in the leaves. Since for every value of team1 there are 100 games in the relation, we need to access 100/19 = 6 pages besides the first leaf accessed, and then we have to access 100 pages by following the pointers to the relation pages. In total we have $\log_{19}$ 100.000 + 6 + 100 = 110 page accesses.

- We have just seen that the left-hand side operand fits in 4 pages of the buffer. So, we can simply use the one-pass join algorithm: we read the pages of SOCCERGAME one at a time, we compute both the projection on <team2,code> and the join in the buffer, and we use one buffer frame for the output, as usual.

The physical query plan will be as follows.



The toal cost is 110 + 190.000 = 190.110 page accesses.

Problem 3

Consider the relation FLIGHT(fcode,company,departure,destination), storing the code, the air company, and the cities of departure and destination of a set of flights, and the relation TAKES(flight,person,date,cost), recording the flights taken by the various persons in the various dates, with the corresponding cost. We know that TAKES has 2.000.000 tuples, each page contains 50 such tuples in the average, and there is a dense B^+-tree index on TAKES with search key ⟨flight,person⟩ using alternative 2. We also know that FLIGHT is stored in a file with 400.000 pages sorted on ⟨fcode,company⟩. Finally, we know that there are 100 buffer frames available, each flight is taken by 500 persons at most, and each value or pointer occupies the same space in memory. Consider the query

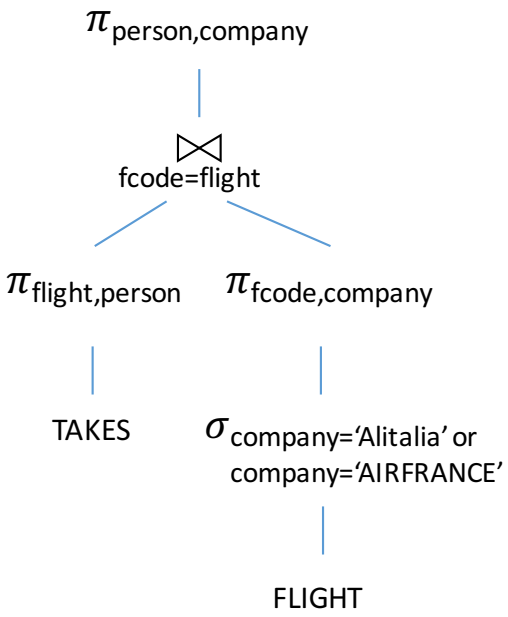
```
select person, company
from TAKES join FLIGHT on fcode = flight
where company = 'ALITALIA' or company = 'AIRFRANCE'
```

3.1 Describe the logical query plan associated to the query code, and illustrate both the logical and the physical query plan you would select, motivating the choices.

3.2 Tell which is the cost (in terms of number of page accesses) of executing the query according to the selected physical query plan.

Solution of problem 3

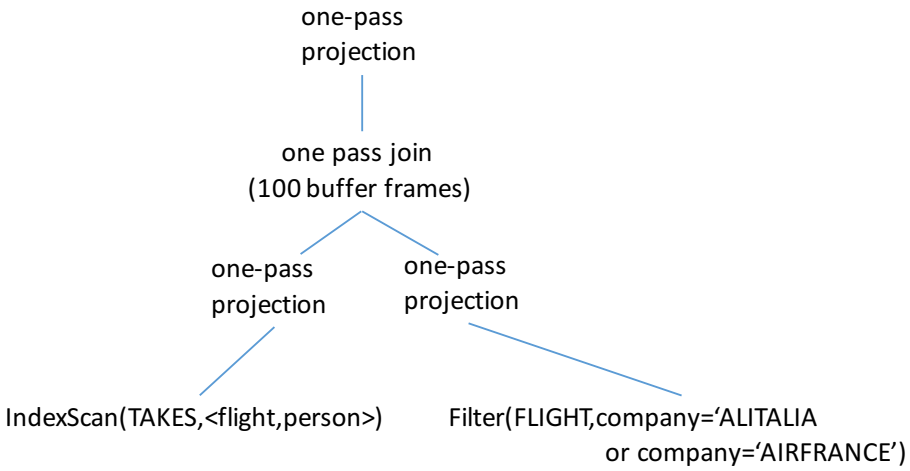
We do not show the the logical query plan associated to the query code because it is obvious. It is immediate to verify that pushing projection is beneficial. Thus, we obtain the following selected logical query plan:



We note that the leaves of the tree index on ⟨`flight,person`⟩ has all attributes needed for the join and the projection, and that the relation `FLIGHT` is sorted on ⟨`fcode,company`⟩.

Also, we note that given that each flight is taken by 500 persons at most, we we have 100 buffer frames available, for each value of `flight` we know that all the joining tuples that are in the relation `TAKES` fit in the buffer. So, for the left-hand side operand, we can use an index-based access, and we can simply use a one-pass algorithm for the join, knowing that there will never be the risk of the number of joining tuples exceeding the size of the buffer.

The physical query plan will be as follows.



For computing the cost, we have to compute the number of pages in the leaves of the tree index. Since each page has room for 200 values, each page has room for 66 data entries, and by taking into account the 67% occupancy rule, we know that each leaf has 60 data entries. Since the index is dense, we have that the number of leaves in the index is 2.000.000/60 = 33.333. The total cost is therefore 400.000 + 33.333= 433.333.