Data Management – AA 2017/18 – exam of 12/01/2018

Problem 1
We have to compute the bag difference between the two tables `R(A,B,C)` and `S(A,B,C)`, where each table is stored in a file sorted on `<A,B,C>`. We know that `R` occupies 4.000 pages, and `S` occupies 5.000. Describe in detail the algorithm you would use to perform the operation, knowing that you have 100 buffer frames available. Also, tell which is the cost of such algorithm in terms of number of page accesses, explaining the answer in detail.

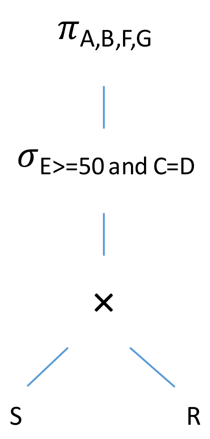
Solution of problem 1
The algorithm is trivial: since both relations are sorted on all the attributes, we can compute the result in one pass, using 3 out of the 100 buffer frames available: one buffer frame F_R for reading the pages of `R`, one buffer frame F_S for reading the pages of `S`, and one buffer frame for the output. Whenever we have a tuple t of `R` under consideration in F_R, we count the number NS_t of occurrences of t in `S`, by reading from the frame F_S (possibly, loading other pages of `S`, if needed). We then scan all the NR_t occurrences of t in `R` (possibly loading other pages in F_R, if needed), and we copy $NR_t - NS_t$ occurrences of t in the output frame (that is written on disk whenever is full). Obviously, the cost is $4.000 + 5.000 = 9.000$ page accesses.

Problem 2
Let `R(A,B,C)` and `S(D,E,F,G)` be two relations, where `R` is stored in 7.500 pages, and the 300.000 tuples of `S` are stored in 3.000 pages. We know that `S` has a dense, clustering B$^+$ index with search key `E` using alternative 2, every value or pointer occupies the same space, and our system has 501 free buffer frames available. Consider the query

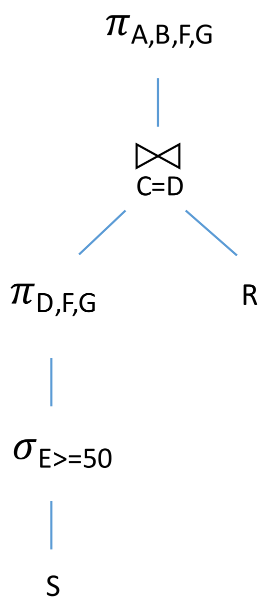
```
select A,B,F,G
from S, R
where E >= 50 and C = D
```

2.1 Describe the logical query plan associated to the query code.
2.2 Describe the selected logical query plan, explaining why such logical plan has been selected.
2.3 Describe the selected physical query plan, explaining why such physical plan has been selected.
2.4 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
The logical query plan associated to the query code is the following:



Obviously, we turn the cartesian product into an equi-join. Also, it is immediate to verify that pushing both selection and projection is beneficial. Thus, we obtain the following selected logical query plan:



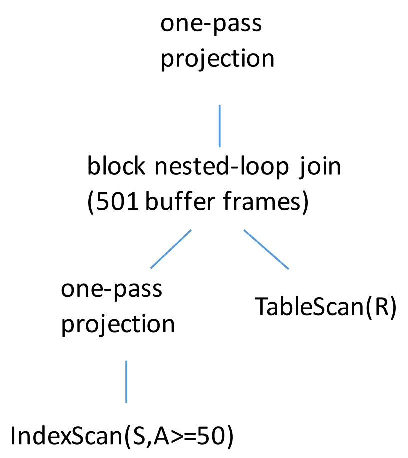
We now have to decide about the physical query plan. There are two decisions to take: how to access `S`, and which algorithm to use for the join.

- As for the access to `S`, since the index on `S` is clustered and the access is range-selection, we can use the index for accessing the `S` file, and get to the first of the 3.000/3 = 1.000 pages storing the tuples satisfying the selection operator. These pages can then be scanned to compute the projection whose size is 1.000 × 3/4 = 750 pages. What is the cost of getting to the right leaf of the index? As usual, we have to compute the number of leaves and the fan-out of the tree. Since we know that 300.000/3.000 = 100 tuples (each with four values) of `S` fit in one page, we deduce that 100 × 4 /2 = 200 data entries (each of two values) fit in one page, and taking into account the 66% occupancy rule, we conclude that 133 data entries are stored in each leaf, and, since the index is dense, we have 300.000/133 = 2.256 leaves of the tree. Also, we conclude that (200 + 100)/2 = 150 is the fan-out of the tree. Therefore, the cost of getting to the right leaf is $\log_{150} 2.256 = 2$.

- As for the algorithm to use for the join, we have just seen that the left-hand side operand has 750 pages. The right-hand side operand will have 7.500 pages. Since we cannot use the one-pass algorithm, let us consider the following two options (note that in both cases we can avoid materializing the left-hand side operand):

    1. **Two-pass algorithm** for the join (assuming that we use the algorithm based on sorting): in this case the cost of the whole query would be as follows: 1.002 + 7.500 + 2 × (750 + 7.500) = 25.002, where 1.002 is the cost of using the index, selecting, projecting and reading the tuples of `S` in the first pass of the algorithm, 7.500 is the cost of reading the pages of `R` in the first pass of the algorithm, and 2 × (750 + 7.500) is the cost of the second pass of the algorithm, where we write the sublists and then read them to produce directly the result with a final projection operation (as usual, we ignore the cost of writing the result).
    2. **Block nested-loop algorithm**: in this case the cost would be 1.002 + 7.500 × (1 + 750/500) = 1.002 + 7.500 × 2 = 16.002, where 1.002 is the cost of using the index, selecting, projecting and reading the tuples of `S` in two steps of the outer loop, and 7.500 × 2 is the cost of reading the relation `R` twice (once for every execution of the outer loop). Note that, as before, we produce directly the result with a final projection operation while writing the output, and we ignore the cost of writing such result.

It is clear that the block nested-loop algorithm is preferrable, and therefore the physical query plan will be as follows.



Problem 3
Consider the relation `TRAVEL(code,agency,country,date,cost)`, storing information about a set of travels, where for each travel we have its code (primary key), the agency that organized it, the country visited during the travel, and the starting date of the travel. The relation has 640.000 tuples stored in a heap file, where each page contains 80 tuples. Consider the aggregate query Q that, for each agency a, computes the average cost of the travels organized by a, and assume that we have a good hash function on `agency` that distributes the tuples of `TRAVEL` uniformly into 100 buckets. You are asked to describe the algorithm you would use for computing Q, and tell which is the cost of executing the algorithm in terms of number of page accesses, in the following two scenarios:

3.1 under the assumption that we have only the processor where `TRAVEL` is stored, with 101 free buffer frames available;

3.2 under the assumption that we have 100 processors besides the one where `TRAVEL` is stored, each one with 90 free buffer frames available.

Solution of problem 3
It is immediate to verify that `TRAVEL` has 8.000 pages.

3.1 Under the assumption that we have only one processor, with 101 free buffer frames available, we cannot use a one-pass algorithm, and therefore we opt for a two-pass algorithm based on hashing. The cost is $3 \times 8.000 = 24.000$.

3.2 Under the assumption that we have 100 processors besides the one where `TRAVEL` is stored, each one with 90 free buffer frames available, we can use the parallel algorithm based on hashing. We read the pages of `TRAVEL` in one input frame, and we hash each tuple in order to distribute all the tuples to the various buckets, i.e., to the various processors, so as to store $(640.000/100)/80 = 80$ pages in the buffer at each processor. At each processor, we then compute the result. The elapsed time can be characterized by the cost of 8.000 page accesses, i.e., the cost of reading the relation.

Problem 4
Consider the following schedule

$$S = r_1(x)\, r_2(y)\, w_2(x)\, r_3(v)\, w_1(z)\, w_4(v)\, r_1(y)\, w_4(z)\, w_2(z)\, r_3(z).$$

4.1 Tell whether S is a 2PL schedule or not, explaining the answer in detail.

4.2 Construct the precedence graph associated to S, and tell whether S is conflict-serializable or not, explaining the answer in detail.

4.3 Describe the behavior of the timestamp-based scheduler when processing the schedule S, assuming that, initially, rts(α)=wts(α)=wts-c(α)=0, and cb(α)=`true` for each element α of the database, and assuming that the subscript of each action denotes the timestamp of the transaction executing such action.

4.4 Tell whether S is in the class of ACR (Avoiding Cascading Rollback) schedules or not, and whether S is in the class of strict schedules, explaining the answer in detail for both cases.

Solution of problem 4
It is immediate to verify that the precedence graph associated to S is cyclic, and thus S is not conflict-serializable, and therefore is not a 2PL schedule. It is also immediate to verify that S is both in the class of ACR (Avoiding Cascading Rollback) schedules, and in the class of strict schedules. It is also easy to come up with the description of the behavior of the timestamp-based scheduler when processing the schedule S.

.

Problem 5
A schedule is called "conflict-restricted" if it has no conflict of type `write-write`, and no conflict of type `read-write`. Proof or disprove the following claim: every "conflict-restricted" schedule that is in the class ACR (Avoiding Cascading Rollback) is conflict-serializable.

Solution of problem 5 We prove the claim by showing that any "conflict-restricted" schedule S that is in the class ACR is conflict-equivalent to the serial schedule built according to the order in which the various transactions in S commit. We will only use the definition of conflict-equivalent schedules in the proof.

Let S be a "conflict-restricted" schedule S that is in the class ACR, let $T_1, \ldots, T_n$ the transactions appearing in S, and let S' be the serial schedule on $T_1, \ldots, T_n$ built according to the order in which the various transactions in S commit. We prove that S is conflict-equivalent to S' by showing that every pair of conflicting actions appearing in S appear in the same order in S'.

Let a_i be an action on element X of transaction T_i, and b_j an action on X of transaction T_j. Assume that a_i preceeds b_j is S, and that a_i and b_j are conflicting actions. Since S is "conflict-restricted", it must be that b_j is the read action $r_i(X)$ and a_i is the write action $w_j(X)$. Since S is in ACR, the commit action c_i must appear between a_i and b_j is S, and therefore T_i commits before T_j in S. Since S' is the serial schedule on $T_1, \ldots, T_n$ built according to the order in which the various transactions in S commit, it must be that T_i comes before T_j in S', and therefore a_i comes before b_j in S'. Since we have shown that S is conflict-equivalent to the serial schedule S', it follows that S is conflict-serializable, and the claim is proved.