Data Management – exam of 11/06/2021
Solutions

Problem 1
It is well-known that the scheduler called "Serialization Graph Tester" aims at ensuring conflict serializability, based on the precedence graph associated to the current schedule. In order to prevent the precedence graph to grow indefinitely, consider the following strategy: when a transaction t commits, remove from the precedence graph the node corresponding to the transaction t, as well as its ingoing and outgoing edges. Prove or disprove that such strategy is correct with respect to the goal of checking the current schedule for conflict serializability.

Solution to problem 1
We disprove that the strategy is correct. By referring to the following schedule S

$$r_1(x)\ \ w_2(x)\ \ w_2(y)\ \ c_2\ \ r_1(y)\ \ c_1$$

it is immediate to see that if we remove T_2 and all its edges from the precedence graph when c_2 is issued, we miss to recognize the cycle involving T_1 and T_2 that will be created when $r_1(y)$ is issued, and therefore we miss the opportunity to figure out that S is not conflict serializable.

Problem 2
Consider the following schedule S (where we have relaxed the condition that no transaction contains more than one occurrence of the same action):

$$B(T_1)\ \ r_1(A)\ \ B(T_2)\ \ w_2(A)\ \ B(T_3)\ \ r_3(A)\ \ r_3(B)\ \ c_2\ \ r_1(A)\ \ r_1(B)\ \ B(T_4)\ \ r_4(A)\ \ c_1\ \ c_3\ \ c_4$$

where the action B means "begin transaction", the initial values of A and B are 10 and 50, respectively, and every write action increases the value of the element on which it operates by 10.

1. Suppose that S is executed by PostgreSQL, and tell which are the values read by all the "read" actions in the schedule, in both the following two cases: (1) all the transactions are defined with the isolation level "read committed"; (2) all the transactions are defined with the isolation level "repeatable read".
2. Tell whether S is a 2PL schedule with both exclusive and shared locks, motivating the answer in detail.

Solution to problem 2

1.1 If all the transactions are defined with the isolation level "read committed", we have the following: $r_1(A)$ reads 10, $r_3(A)$ reads 10, $r_3(B)$ reads 50, $r_1(A)$ reads 20, $r_1(B)$ reads 50, and $r_4(A)$ reads 20.

1.2 If all the transactions are defined with the isolation level "repeatable read", we have the following: $r_1(A)$ reads 10, $r_3(A)$ reads 10, $r_3(B)$ reads 50, $r_1(A)$ reads 10 **(this is the only difference with respect to "read committed": with "repeatable read" as isolation level, no two reads of the same element from the same transaction can result in different values read from the database)**, $r_1(B)$ reads 50, and $r_4(A)$ reads 20.

2 S is not a 2PL schedule with both exclusive and shared locks. It is sufficient to observe that T_1 must release the lock on A for letting T_2 to proceed, but at that point it cannot re-acquire the lock on A without violating the 2PL protocol.

Problem 3
Let $R(A,B)$ and $S(C,D)$ be two relations with $B(R)$ and $B(S)$ pages, respectively, where R is stored in a file sorted on B, and S is stored in a heap. Let us denote by $\sigma(R,S)$ the operator computing the following result:

$$\sigma(R,S) = \{\ x \in R[A] \mid \text{for all } y \text{ such that } (x,y) \in R, \text{ we have that } (x,y) \in S\ \}$$

Assuming that $M > 2$ free buffer frames are available, describe the most efficient algorithm that you think we can use to compute the result of $\sigma(R,S)$, and tell which is the cost (as a function of M, $B(R)$ and $B(S)$) of such algorithm in terms of number of page accesses.

Solution to problem 3
(1) If $M - 2 \geq B(R)$, then we can use a trivial one-pass algorithm, with cost $B(R) + B(S)$.

(2) Otherwise, unfortunately, the fact that R is sorted on B does not help at all for computing the result of $\sigma(R,S)$, because what we have to compute is a subset of the projection of R on A. One might think of a nested-loop algorithm for the problem, but the fact that we might have many tuples anywhere in R with the same value x as a first component suggests that we cannot adopt a simple nested-loop strategy for solving the problem. Indeed, when we have a page P of R under consideration, and we consider a tuple $t = \langle x, y \rangle$ in P, we cannot decide if x is in the result by simply loading all the pages of S in the buffer, because there might be other tuples of the form $t = \langle x, y' \rangle$ in the subsequent pages of R, and such tuples are crucial with respect to the goal of deciding if x is in the result.

The above considerations suggest that we should have all tuples of R with the same value x in the first component close to each other. In order to obtain this condition, we can simply sort R on A and S on C: this way, we can adopt a "one pass merge" strategy for finding, given $x \in R[A]$, all the tuples in R and S with x as a first component, thus allowing us to decide fast whether x should be included or not in the result. So the algorithm can be structured as follows (where we assume that $M > 2$, and we devote M_R buffer frames to R and M_S buffer frames to S):

1. produce $M_R = \lfloor (M-2) \times B(R)/(B(R) + B(S)) \rfloor$ sorted sublists of R (where the sorting is on A), by means of as many passes as needed;
2. produce $M_S = \lfloor (M-2) \times B(S)/(B(R) + B(S)) \rfloor$ sorted sublists of S (where the sorting is on C), by means of as many passes as needed;
3. perform the "merge" phase by using one buffer frame for the output, where we put only those tuples satisfying the condition, and we write the output frame in secondary storage whener it is full.

Note that $M_R + M_S = M - 2$, which means that the merging phase can indeed be performed correctly.

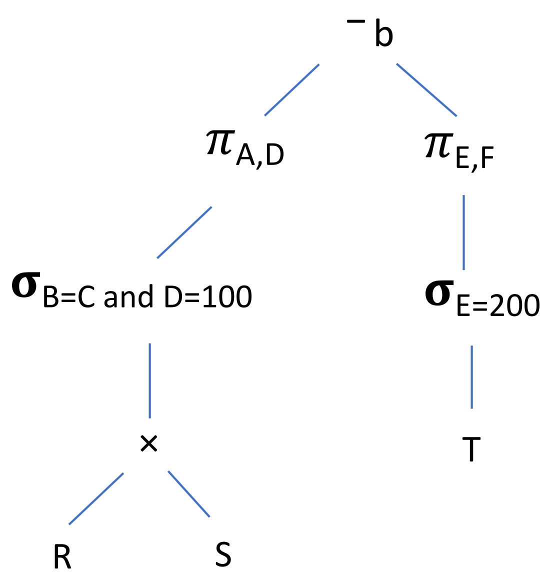
The cost of the algorithm is $2 \times P_R \times B(R) + 2 \times P_S \times B(S) + B(R) + B(S)$, where P_R is the number of passes needed to produce the sorted sublists of R, i.e., $log_M B(R)$, and P_S is the number of passes needed to produce the sorted sublists of S, i.e., $log_M B(S)$.

Problem 4
Consider the relations `R(A,B)` with `B` as the key and 1.000 pages, `S(C,D)` with 3.000 pages and `T(E,F)` with 250.000 pages, each of them stored in a heap. We know that each page of every relation has space for 100 tuples. We also know that V`(S,C)` = 1.000, V`(S,D)` = 10, V`(T,E)` = 1.000 and that we have 302 free buffer frames available. Consider the query ("minus" is the difference operator):

```
select A,D from R, S where B=C and D=100
 minus
select E,F from T where E = 200
```

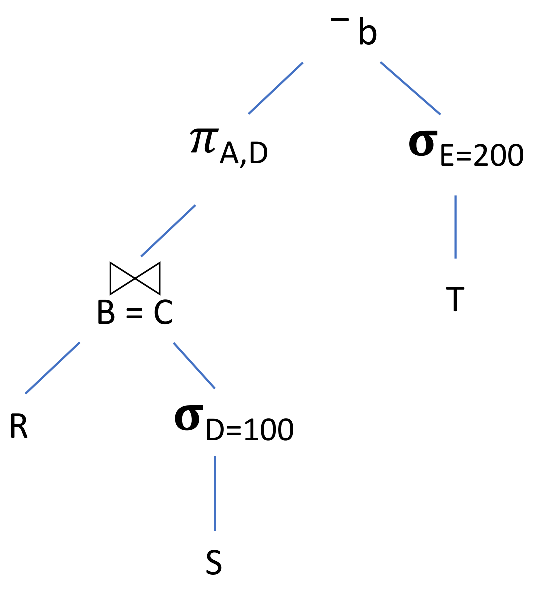
Show the logical query plan associated to the query, as well as the logical query plan and the physical query plan you would choose for executing the query efficiently. Also, tell which is the cost (in terms of number of page accesses) of executing the query according to the chosen physical query plan.

Solution to problem 4
The logical query plan associated to the query is shown below:



Logical query plan

The chosen logical query plan is shown below:

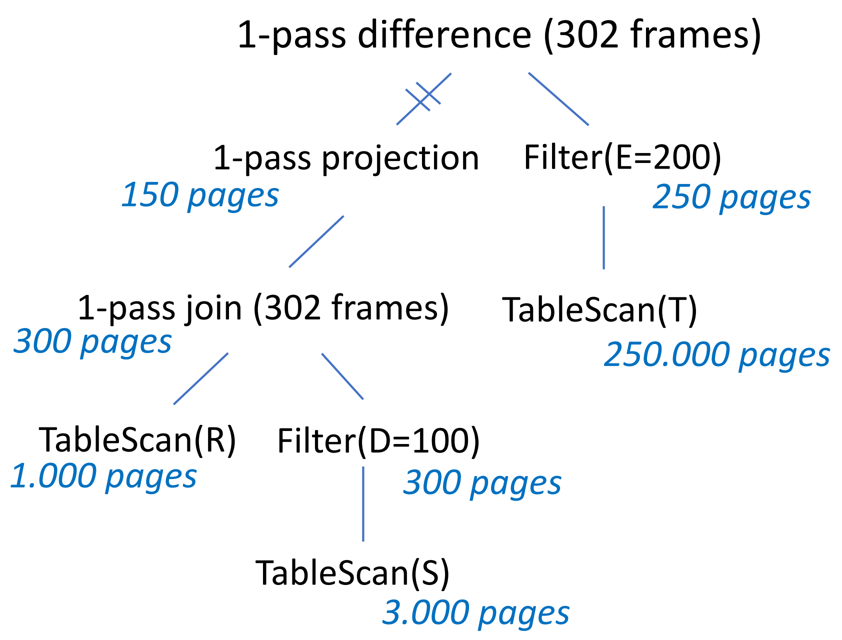


The chosen logical query plan

We can estimate the size of the various operands as follows:

- T has 250.000 pages and 25.000.000 tuples
- The result of the σ operator over T has 250.000 / V(T,E) = 250.000 / 1.000 = 250 pages
- R has 1.000 pages and 100.000 tuples
- S has 3.000 pages and 300.000 tuples
- The result of the σ operator over S has 3.000 / V(S,D) = 3.000 / 10 = 300 pages
- Since B is the key of R, we have that V(R,B) coincides with the number of tuples in R, i.e., 1.000 $\times$ 100 = 100.000, and therefore the maximum between V(R,B) and V(S,C) is 100.000. It follows that the result of the join has 100.000 $\times$ 30.000 / 100.000 = 30.000 tuples, i.e., 300 pages
- Since the join has 4 attributes, and the π operator over the join has 2 attributes, the result of the π operator over the join has 300/2 = 150 pages.

It is easy to see all the operations can be done in one pass, and that the result of the projection after the join must be materialized, because we do not have space in the buffer to store it. It follows that the physical query plan is as shown below:



Physical query plan

Since all operators are evaluated in one pass, the cost is 250.000 (reading T) + 3.000 (reading S) + 1.000 (reading R) + 150 (writing the result of the projection after the join) + 150 (reading the result of the projection after the join for the 1-pass difference) = 254.300 page accesses.

Problem 5

Let $\texttt{R}_1\texttt{(A,C)}$ be a relation with $B(\texttt{R}_1)$ pages stored in a heap, and let $\texttt{R}_2\texttt{(}\underline{\texttt{A}}\texttt{,C)}$ be a relation with A as the key and with $B(\texttt{R}_2)$ pages also stored in a heap. We know that $B(\texttt{R}_1) < B(\texttt{R}_2)$, and we have to compute the difference between $\texttt{R}_1$ and $\texttt{R}_2$, in two different scenarios: (*a*) $\texttt{R}_1$ has no duplicates; (*b*) $\texttt{R}_1$ has duplicates. For each of the above scenarios, tell whether the "block-nested loop algorithm" can be used or not; if the answer is negative, then motivate the answer in detail, and if the answer is positive, then describe the algorithm and characterize its cost in terms of number of page accesses, assuming that we have $M > 2$ buffer frames available.

Solution to problem 5

Here, by "the difference between R_1 and R_2" we mean $R_1 - R_2$.

1. If $\texttt{R}_1$ does not contain duplicates, then the block-nested loop algorithm can be used. Indeed, we load one "block" ($M - 2$ pages forming, say, the block b) of R_1 at the time, and we load in the $(M - 1)$-th buffer frame all the pages of R_2, one at a time. After we have seen all the pages of R_2 we know which are the tuples in b that are in the result, and therefore we can store such tuples in the M-th frame (when such frame is full, we copy its content in secondary storage). Since the tuples in b cannot appear in the subsequent blocks of $R_!$, this strategy is correct. The cost is $B(R) + B(R) \times B(S)/(M - 2)$.

2. If $\texttt{R}_1$ does contain duplicates and $M - 2 \geq B(R_1)$, then the block-nested loop coincides with the one pass algorithm and can be trivially used.

   If $\texttt{R}_1$ does contain duplicates and $M - 2 < B(R_1)$, the block-nested loop algorithm cannot be used, because once we have loaded a block b of R_1 and we have analyzed all the tuples of R_2, we cannot take any decisions on the tuples in b, since the same tuple can appear in the subsequent blocks of R_1.

In the case where by "the difference between R_1 and R_2" we mean $R_2 - R_1$, we notice that R_2 does not have duplicates, and therefore the algorithm can be analogous to the one illustrated for case 1 above, and the cost would $B(S) + B(R) \times B(S)/(M - 2)$.