Data Management – AA 2019/20
Solutions for the exam of 09/07/2020

Problem 1
If S is a schedule on transactions $T_1, \ldots, T_n$, then the *reduced precedence graph* $\rho(S)$ associated to S is a graph that has the transactions in S as nodes, and has an edge from T_i to T_j if and only if S contains two conflicting actions $a_i(x)$ in T_i and $a_j(x)$ in T_j on the same element x such that $a_i(x)$ precedes (not necessarily directly) $a_j(x)$ in S, and there is no action $\alpha_k(x)$ in S between $a_i(x)$ and $a_j(x)$, with k different from i and j. Prove or disprove the following claims:

1. If $\rho(S)$ is cyclic, then S is not conflict serializable.
2. If $\rho(S)$ is acyclic, then S is conflict serializable.
3. If $\rho(S)$ is cyclic, then S is not view serializable.

Solution to problem 1

1. The claim can be proved by noticing that $\rho(S)$ is a subset of the precedence graph $P(S)$ associated to S. Indeed, the two graphs have obviously the same nodes, and, each edge in $\rho(S)$ is the witness of two conflicting actions in S, and therefore such edge also appears in $P(S)$.

2. The claim can be disproved by the following counterexample S_1:

$$r_1(x)\ r_2(x)\ w_3(x)\ w_3(y)\ w_1(y)$$

   The reduced precedence graph $\rho(S_1)$ contains one edge from T_2 to T_3 (because of the conflicting actions $r_2(x)$ and $w_3(x)$), and one edge from T_3 to T_1 (because of the conflicting actions $w_3(y)$ and $w_1(y)$), and therefore is acyclic. On the other hand, the precedence graph $P(S_1)$ contains, in addition, the edge from T_1 to T_3 (because of the conflicting actions $r_1(x)$ and $w_3(x)$), which creates a cycle in $P(S_1)$, showing that S_1 is not conflict serializable. Thus, $\rho(S_1)$ is acyclic, and S_1 is not conflict serializable.

3. The claim can be disproved by the following counterexample S_2, which is exactly the one used to show that there are view serializable schedules that are not conflict serializable:

$$w_1(x)\ w_2(x)\ w_2(y)\ w_1(y)\ w_3(x)\ w_3(y)$$

   Indeed, the reduced precedence graph $\rho(S_2)$ contains a cycle (involving T_1 and T_2), but S_3 is clearly view serializable (for example, the serial schedule T_1, T_2, T_3 is view equivalent to S_2).

Problem 2 Consider the following schedule S:

$$r_1(x)\ r_2(x)\ w_3(x)\ w_3(z)\ c_3\ r_4(z)\ w_4(y)\ c_4\ w_1(y)\ c_1\ r_2(y)\ c_2$$

1. Is S a 2PL schedule with both shared and exclusive locks? Motivate your answers in detail.
2. Describe the behavior of the timestamp-based scheduler when processing S, assuming that, initially, for each element α of the database, we have rts(α)=wts(α)=wts-c(α)=0, and cb(α)=`true`, and assuming that the subscript of each action denotes the timestamp of the transaction executing such action.

Solution to problem 2

1. S is not a 2PL schedule with both shared and exclusive locks: after action $r_2(x)$, transaction T_2 must release the shared lock on x to allow $w_3(x)$ to be executed, and therefore it must enter the shrinking phase. This means that, before releasing such lock, it should acquire the shared lock on y (for the future action $r_2(y)$). However, this cannot be done without blocking the action $w_4(y)$ appearing before $r_2(y)$.

2. Here is the behavior of the timestamp-based scheduler when processing S:

    (a) $r_1(x) \rightarrow$ OK $\rightarrow$ rts$(x) = 1$

    (b) $r_2(x) \rightarrow$ OK $\rightarrow$ rts$(x) = 2$

    (c) $w_3(x) \rightarrow$ OK $\rightarrow$ wts$(x) = 3$, cb$(x) =$ false

    (d) $w_3(z) \rightarrow$ OK $\rightarrow$ wts$(z) = 3$, cb$(z) =$ false

    (e) $c_3 \rightarrow$ OK $\rightarrow$ wts-c$(x) = 3$, wts-c$(z) = 3$, cb$(x) =$ true, cb$(z) =$ true

    (f) $r_4(z) \rightarrow$ OK $\rightarrow$ rts$(z) = 4$

    (g) $w_4(y) \rightarrow$ OK $\rightarrow$ wts$(y) = 4$, cb$(y) =$ false

    (h) $c_4 \rightarrow$ OK $\rightarrow$ wts-c$(y) = 4$, cb$(y) =$ true

    (i) $w_1(y) \rightarrow$ OK (Thomas rule)

    (j) $c_1 \rightarrow$ OK

    (k) $r_2(y) \rightarrow$ read too late $\rightarrow$ T_2 aborted

Problem 3

Let `R(A,B,C,D)` be a relation stored in a heap file with 89.100 pages, and consider the following query:

```
select A, count(*) from R group by A
```

We have two options for executing the query: (1) We execute it at a single processor P having 300 buffer frames available. (2) We carry out a parallel execution of the query using n processors (with $n > 0$), each one with 16 buffer frames available. You are asked to tell if there is a value for n such that strategy (2) is more efficient (with respect to the elapsed time) than strategy (1). If the answer is positive, then tell which is the minimum value n such that the above condition holds, and tell which is the corresponding cost, in terms of both the total number of page accesses, and the elapsed time. If the answer is negative, then explain why. In both cases, motivate your answer in detail.

Solution to problem 3

If we execute the query at a single processor with 300 buffer frames available, then, after the projection on `A` (whose size in terms of number of pages is 89.100/4= 22.275) computed in pipleline mode, we have to use a two pass algorithm for grouping (because $22.275 > 300$ and $22.275 < 300 \times 299$), for example based on sorting. The cost is $3 \times 22.275 = 66.825$ page accesses (which coincides with the elapsed time, in this case).

Suppose that we use two processors, each with 16 buffer frames available. We split the 22.275 pages into two pieces of 11.138 pages each, based on a simple hash function on `A`, and send each piece to a different processor. In this case, since at each processor we only have 16 buffer frames available, at each of the processors we need 4 passes (because $11.138 > 15 \times 15 \times 16$, while $11.138 \leq 15 \times 15 \times 15 \times 16$), and therefore the elapsed time is $7 \times 11.138=$ 77.966, which is higher than the elapsed time of the single processor case.

Suppose that we use three processors, each with 16 buffer frames available. We split the 22.275 pages into three pieces of 7.425 pages each based on a hash function on `A`, and send each piece to a different processor. In this case, since at each processor we only have 16 buffer frames available, at each of the processors we still need 4 passes (because $7.425 > 15 \times 15 \times 16$, while $7.425 \leq 15 \times 15 \times 15 \times 16$), and therefore the elapsed time is $7 \times 7.425 = 51.975$, which is lower than the elapsed time of the single processor case.

We conclude that there are values for n such that strategy (2) is more efficient (with respect to the elapsed time) than strategy (1), and the minimum of such values is 3.

Problem 4
Consider the relation City(code,name,region,country,population,mayor), with 1.000.000 tuples, the relation Theater(city,tcode,size), that contains, for each value of the attribute city, an average of 10 tuples with that value, and the following query Q:

```
select name, region, avg(size)
from City, Theater
where code = city
group by region, name
order by region, name
```

We know that 300 values fit in one page of our system, and that our buffer has 50 free frames. Tell which physical structures (including possible indexes) you would choose for storing the two relations in such a way to optimize the above query Q, and tell the cost of executing the query under the assumptions that the two relations are stored according the chosen method, and all values occupy the same space. Motivate your answers in detail.

Solution to problem 4

The query is essentially a join between the two relations, followed by a grouping and an "order by" clause. To support the efficient execution of the join one could think of storing `City` sorted on `code`, and `Theater` sorted on `city`, so that the join would be performed simply by the "merge" step of the simple sort-based join algorithm. However, with this method, we would need to perform further costly operations (grouping and "order by") after the join. The number of pages of `City` is 1.000.000 $\times$ 6 /300 = 20.000, the number of pages of `Theater` is 10 $\times$ 1.000.000 $\times$ 3 / 300 = 100.000. The merge step of the join would cost 120.000 page accesses. Writing such pages would cost 120.000 again, and then we would need to sort the pages according to `region,name` (in 3 passes), and finally computing the grouping

A more effective method is to use a clustered file in order to store the two relations together: every tuple of `City` will be stored together with all the tuples of `Theater` representing theaters located in that city. So, every record in the clustered file will be constituted by 36 values (on the average), namely 6 values for the tuple t of `City`, and 10 $\times$ 3 values for the tuples of `Theater` corresponding to theaters located in the city represented by t. This means that the clustered file needs 36 $\times$ 1.000.000 / 300 = 120.000 pages. Obviously, in order to support the grouping and the "order by" operation, the clustered file is stored sorted on $\langle$`region,name`$\rangle$. With this file organization, the query can be answered simply by scanning the clustered file in one pass, and, while scanning, computing the various groups, each one with the corresponding `avg(size)`. The cost is 120.000 page accesses.

Problem 5

Let `R`$_1$`(`<u>`A`</u>`,B,C,D,E,F)` be stored in a heap file with 840.000 tuples, and `R`$_2$`(`<u>`G`</u>`,H,L)` be stored in a file sorted on `A`, with 9.000.000 tuples and with an associated B$^+$-tree index on attribute `G`. We assume that every value has the same size, that every page has room for 600 values, that $V($`R`$_1$`,B`$) = 100$, $V($`R`$_1$`,F`$) = 300$, and that we have 125 free buffer frames available. Consider the following query:

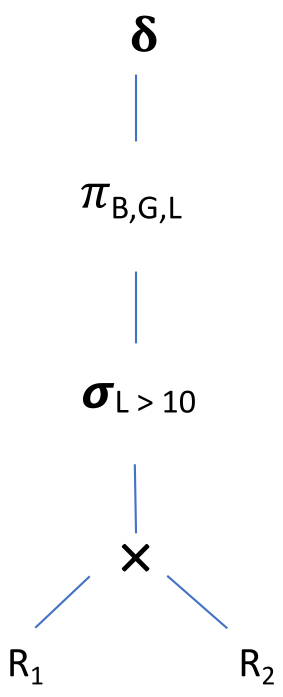
```
select distinct B, G, L
from R1, R2
where F = G and L > 10
```

and answer the following questions:

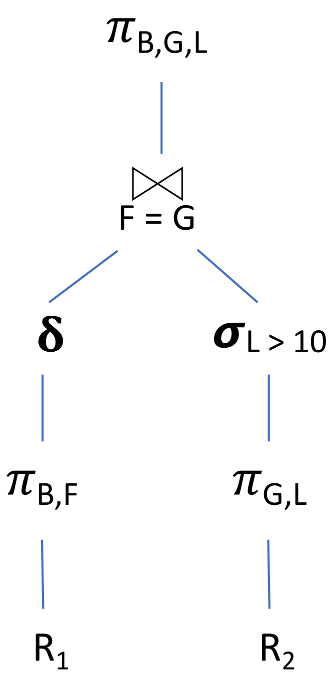
1. Illustrate the logical plan associated to the above query expression.
2. Describe the selected logical plan, motivating the annswer.
3. Describe the physical plan you would choose, and determine the cost of executing the query according the chosen physical plan, motivating the answer.

Solution to problem 5

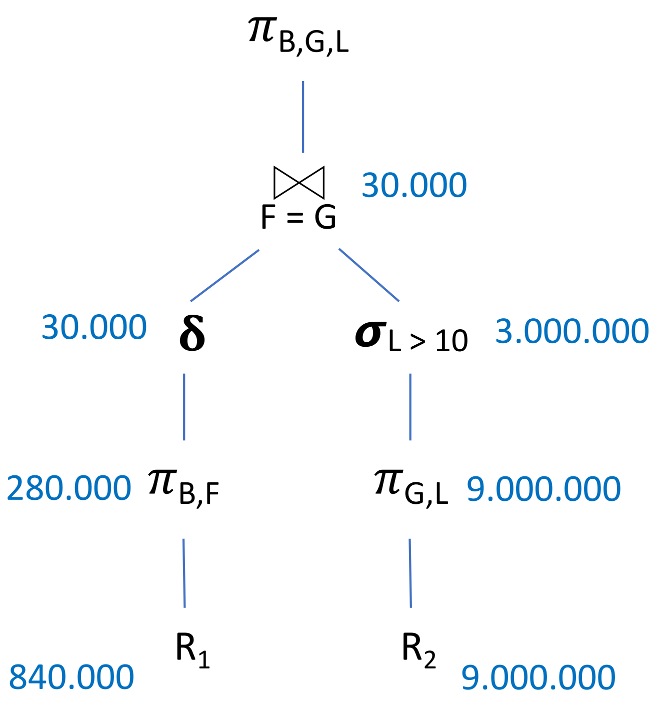
The logical query plan associated to the query is shown below.



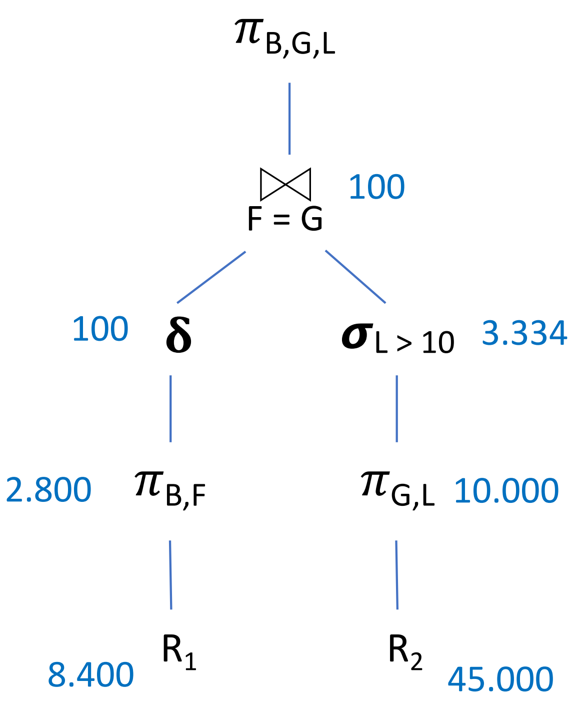
After pushing selection, duplicate elimination and projections we have the following logical query plan:



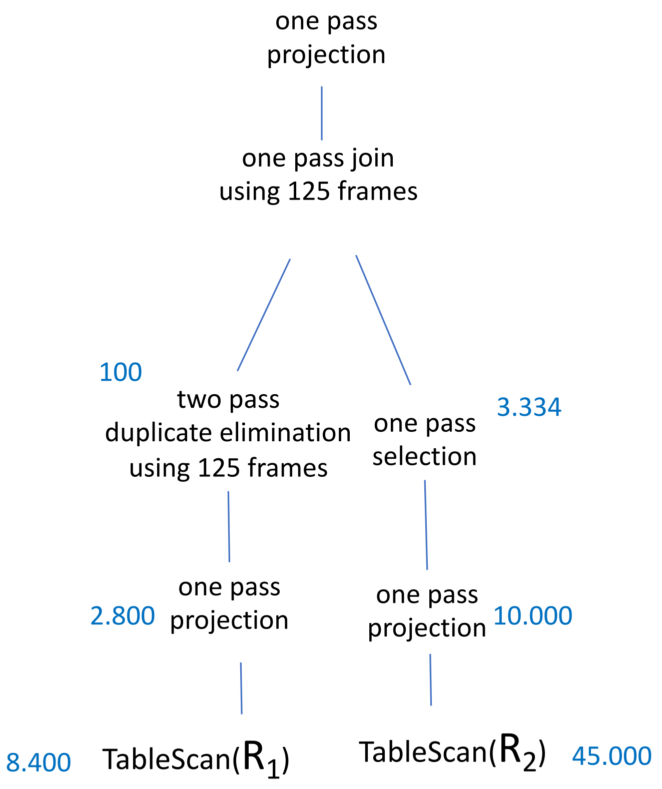
Note that duplicate elimination has not been pushed through the branch of R_2, because the right-hand side operand of the join contains a key of the relation, and therefore does not have duplicates. Note alse that after duplicate elimination we cannot have more that $V(R_1,B) \times V(R_1,F) = 30.000$ tuples. The number of tuples of the various nodes of the plan is shown here:



The number of pages is shown here:



Here is the physical query plan:



whose cost is: 8.400 (reading of $\texttt{R}_1$) + 45.000 (reading of $\texttt{R}_2$) + 2.800 (writing of the 2.800 / 125 = 23 sublists) + 2.800 (reading of the 23 sublists) = 59.000.