Data Management – exam of 13/07/2022

Problem 1
In a schedule S on transactions $\{T_1, \ldots, T_n\}$ we say that two transactions T_i, T_j share the element X of the database if there exist actions $\alpha(X)$ in T_i and $\beta(X)$ in T_j such that α is either r_i or w_i and β is either r_j or w_j. Moreover, S is called "chary" if (i) no transaction in S uses the same element twice, and (ii) for every $T_i, T_j \in \{T_1, \ldots, T_n\}$, T_i and T_j share at most one element. Prove or disprove the following claims:

1.1 Every chary schedule is view-serializable.
1.2 Every chary schedule on two transactions is conflict-serializable.
1.3 Every chary schedule on two transactions is a 2PL schedule with exclusive and shared locks.

Solution

3.1 The claim can be disproved by the following counterexample:

$$r_1(x)\, w_2(x)\, r_3(y)\, w_1(y)\, r_2(z)\, w_3(z)$$

Indeed, the schedule S is clearly chary but is not view serializable, because in any serial schedule with the same transactions of S, the read-from relation is non-empty, and therefore differs from the one of S, which is empty.

1.2 We prove the claim by showing that every schedule that is not conflict-serializable is not chary. Consider a schedule S on two transactions $\{T_1, T_2\}$ and assume S not conflict-serializable, meaning that there is a cycle in the precedence graph associated to S. This in turn means that there are at least two different pairs of conflicting actions in S, say $\alpha(x), \beta(x)$ and $\gamma(y), \delta(y)$ appearing in different orders in S. There are two cases: either $x = y$, or $x \neq y$. In the first case, we have at least three actions using the same element, and therefore at least one of the two transactions uses the same element twice, thus contradicting one of the conditions for chary schedules. In the second case, T_1 and T_2 share two different elements, thus again contradicting one of the conditions for chary schedules.

1.3 The claim can be proved by osserving that every chary schedule on transactions T_1, T_2 can follow the following protocol, which is clearly a specialization of the 2PL protocol:

- At the beginning of the schedule, T_1 acquires the lock for every element it uses, except for the element shared with T_2.
- Immediately after that, T_2 acquires the lock for every element it uses, except for the element shared with T_2.
- Just before the first action (say by transaction T_i) on the element x shared by the two transactions, T_i locks the element x, executes the action, and then immediately releases the lock on x. Just before the action of T_j (where $j \neq i$) on the element x, T_j acquires the lock on x.
- At the end of the schedule, all the locks held by T_1 and T_2 are released.

Problem 2
Let S be the schedule: $r_1(Z)\, w_3(Y)\, w_3(V)\, r_1(Y)\, r_2(V)\, w_2(Y)\, w_3(X)\, r_2(X)\, r_2(Z)\, r_3(Z)\, w_4(Z)\, w_4(X)\, w_2(X)$

2.1 Tell whether S is accepted by the 2PL scheduler with exclusive and shared locks. If the answer is yes, then specify the 2PL schedule obtained from S by adding suitable lock and unlock commands. If the answer is no, then explain the answer.
2.2 Tell whether S is view-serializable. If the answer is yes, then illustrate a serial schedule which is view-equivalent to S. If the answer is no, then explain the answer.
2.3 Answer all the following questions, motivating the answers: (i) Is S recoverable? (ii) Is S ACR? (iii) Is S strict?

Solution

2.1 Since every schedule accepted by the 2PL scheduler with exclusive and shared locks is view-serializable, and since S is not view-serializable (see later), we conclude that S is not accepted by the 2PL scheduler with exclusive and shared locks.

2.2 Simply by considering the last fragment of S:

$$\cdots r_2(Z)\, r_3(Z)\, w_4(Z)\, w_4(X)\, w_2(X)$$

we conclude that S is not view-serializable. Indeed, in any serial schedule in which T_4 comes before T_2, we have that $r_2(Z)$ reads-from $w_4(Z)$ (and such reads-from pair is not is S), and in any serial schedule in which T_2 comes before T_4, we miss the final write $w_2(X)$ which is present in S. This implies that any serial schedule on the same transactions of S differ from S in the read-from relation or the final-write set, in turn implying that S is not view-serializable.

2.3 (i) Since T_1 and T_2 are the only transactions with actions reading from other transactions (actually, the only transaction T_3) we conclude that S is recoverable: indeed, it is sufficient that c_3 appears before both c_1 and c_2 to make the schedule recoverable. (ii) It is immediate to note that the action $r_1(Y)$ of T_1 reads from $w_3(Y)$ of T_3 before the commit of T_3 (indeed, the commit of T_3 cannot occur before $w_3(X)$ and $r_3(Z)$); this shows that S is not ACR. (iii) Since S is not ACR, we conclude that it is not strict.

Problem 3

Let R(<u>A</u>,B,C), S(<u>A</u>,D,E), T(A,B,C) be three tables (where T is a bag) and let τ indicate the ternary operator such that $\tau(\mathsf{R},\mathsf{S},\mathsf{T}) = \delta(\mathsf{T} \cup_b \pi_{\mathsf{A,B,C}}(\mathsf{R} \bowtie \mathsf{S}))$, where δ denotes duplicate elimination, $\cup_b$ denotes bag union and $\bowtie$ denotes natural join.

3.1 Design and describe in detail a one pass algorithm that, given R,S,T as above, each one stored as a heap, computes τ(R,S,T).

3.2 Tell what is the weakest condition under which the algorithm can be used and illustrate the cost of the algorithm in terms of number of page accesses.

3.3 Tell what does it change if all the tables have A as key and are stored as sorted file with search key A.

Solution

3.1 A one-pass algorithm could be easily defined in the case where the pages of all the three tables $\delta(\mathsf{T}), \mathsf{R}', \mathsf{S}$ fit in $M-1$ frames, where

- δ(T) denotes the set of tuples of T without duplicates,
- R′ denotes the tuples of R that are not in T, and
- M is the number of buffer frames available (one frame is reserved for the output, as usual).

Indeed, in this case, we can analyze all the pages of R,S,T once using the following strategy:

(1) we first load T, and while doing this we eliminate its duplicates, thus obtaining δ(T) in the buffer;

(2) we load R by avoding to load the tuples of R already appearing in δ(T), thus obtaining R′ (since A is the key for R, R′ does not have duplicates);

(3) we load S;

(4) we consider every tuple $s \in \mathsf{S}$: if s has a joining tuple t in R, then we put t in the output frame, and we delete t from δ(T). At the end, we copy to the output all the tuples of δ(T) still appearing in the buffer.

The condition under which the above algorithm can be used is: $B(\delta(\mathsf{T})) + B(\mathsf{R}') + B(\mathsf{S}) \leq M-1$.

We can actually do better by noticing that we can avoid storing one of the tables R, S in the buffer. Indeed, we can store in the buffer only the smallest between the two tables. After step (1) above, and after storing the smallest between the two tables, if the smallest is R, then we read S in one buffer frame and we execute step (4) above while reading S. If the smallest is S, then we read R in one buffer frame and for each tuple t of R we do the following: if t does not join with S, then we ignore it. Otherwise, we put t in the output frame, and we delete t from δ(T).

3.2 From the algorithm just described it is easy to conclude that the weakest condition under which we can use a one-pass algorithm for computing $\tau(\mathsf{R,S,T})$ is $B(\delta(\mathsf{T})) + B(X) \leq M - 2$, where X is R' if R' is smaller than S, and is S if S is smaller than R'. Obviously, the cost of the algorithm is $B(\mathsf{T}) + B(\mathsf{R}) + B(\mathsf{S})$.

3.3 If all the tables are stored as sorted file on the key A, then it is immediate to see that we can execute a trivial one-pass algorithm by using only 4 frames (one for T, one for R, one for S and one for the output) and by adopting an obvious "merge" strategy on the sorted tuples.

Problem 4

Consider the relations Flight(code,company,type) with 1.000 pages and 10.000 tuples, and Ticket(number,code,company,type) with 2.000 pages and an associated index on Ticket with search key ⟨company,type⟩, for which we know that the cost of retrieving the records with a specific value of attribute company is 3 page accesses. Assume a buffer with 50 frames, and consider the two queries shown below.

Query Q_1:

```
select code, company from Flight
except all   -- not removing duplicates
select code, company from Ticket
```

Query Q_2:

```
select company, type from Flight
except all   -- not removing duplicates
select company, type from Ticket
```

where "except all" denotes bag difference. For both queries Q_1 and Q_2, tell (*i*) whether it is possible to process the query by using a block-nested loop algorithm, and (*ii*) whether it is possible to process the query by using an index-based algorithm. In all four cases, if the answer is positive, then describe the algorithm and tell which is its cost in terms of number of page accesses. If the answer is negative, then motivate the answer in detail.

Solution

(*i*) We analyze the case of processing the queries by using the block-nested loop algorithm.

- In the case of query Q_1, the query consists in a difference between a set (the projection of `Flight` on `code,company` is a set because `code` is a key of `Flight`) and a bag (the projection of `Ticket` on `code,company` may contain duplicates). Therefore, we can surely use the block-nested loop algorithm, because every tuple t of `Flight` appears in one page P_t only, and we decide if keeping the tuple t or not when we have in the buffer the block of `Flight` containing page P_t. Indeed, when we have such block on the buffer, we scan the relation `Ticket` to check whether it contains at least one occurrence of t, in which case we do not copy t in the output, otherwise we copy it in the output. The cost of the algorithm is simply $B(\texttt{Flight}) + B(\texttt{Flight}) \times B(\texttt{Ticket})/50 = 1.000 + 1.000 \times 2.000/50) = 41.000$.
- In the case of query Q_2, since the projection of `Flight` on `company,type` may contain duplicates, the query consists in a difference between two bags, and therefore the block-nested loop algorithm cannot be used: indeed, when loading a block b of pages of `Flight`, we cannot be aware of other duplicates of the tuples in B appearing in the blocks already analyzed, and therefore we cannot decide if we have to keep such tuples or not.

(*ii*) We analyze the case of processing the queries by using an index-based algorithm. In particlar. we refer to an index-based algorithm for computing the difference between two collections using a nested-loop schema as follows: we analyze all the tuples of the first operand and, for each of them, we use the index for finding all the occurrences of the tuple in the second operand. Note that the search key of the index is ⟨company,type⟩, and therefore the index conforms to both the selection condition appearing in Q_1 (code $= v_1$ and company $= v_2$) and the selection condition appearing in Q_2 (company $= v_1$ and type $= v_2$), because in both cases there exists a prefix P of the search key such that, for each attribute in P, there is an equality condition on such attribute in the conjunction (in particular, such prefix P is ⟨company⟩.

- In the case of query Q_1, since the first operand is a set, we can surely use the index-based algorithm: for each tuple $\langle v_1, v_2 \rangle$ in the projection of `Flight` on `code,company`, we know that the tuple has only one occurrence in `Flight`, and therefore it is sufficient to use the index in order to check whether $\langle v_1, v_2 \rangle$ appears in the projection of `Ticket` on ⟨`company, type`⟩: if the answer is positive, then we keep the tuple in the result, otherwise we ignore it. The cost of the algorithm is simply $B($`Flight`$) + NT($`Flight`$) \times 3$, where $NT($`Flight`$)$ denotes the number of tuples in `Flight`. This means that the cost is $1.000 + 10.000 \times 3 = 31.000$ page accesses.
- In the case of query Q_2, the query consists in a difference between two bags, and therefore an index-based algorithm cannot be used: indeed, when considering a tuple in the projection of `Flight` on `code,company`, we cannot be aware of the fact that other duplicates of the tuple have already been considered, and therefore we cannot decide if we have to keep such tuple or not.

Problem 5 (A.Y. 2021/22)
Describe in detail the notion of "star schema" in data warehousing and illustrate the difference between such a notion and the notion of "snowflake schema".

Solution
Please, see the slides of the course, in particular the slides on Data Warehousing.