Data Management – exam of 18/02/2010
Solutions

Problem 1 Consider the following schedule

$$S = r_1(A)\, r_3(C)\, w_3(B)\, r_2(A)\, w_1(B)\, w_1(A)\, w_2(A)\, r_1(C)\, w_3(C)\, r_3(A)\, r_2(D).$$

1. Tell whether S is view-serializable or not. Explain the answer in detail.
2. If the answer to the previous question was yes, then show a serial schedule that is view-equivalent to S, otherwise exhibit a minimal set (i.e., a set of minimal cardinality) of actions of S to remove so that the remaining schedule is not only view-serializable, but also conflict-serializable.

Solution

1. S is not view-serializable. Indeed, every serial schedule S' on the set of transactions constituting S will have either transaction t_1 before t_2, or t_2 before t_1. In the former case (t_1 before t_2), the pair $\langle w_1(A), r_2(A)\rangle$ is in the READS-FROM relation associated to S', while it is not in the READS-FROM relation associated to S. In the latter case (t_2 before t_1), the pair $\langle w_2(A), r_1(A)\rangle$ is in the READS-FROM relation associated to S', while it is not in the READS-FROM relation associated to S. Actually, it is easy to see that S is not even conflict-serializable, because the precedence graph associated to S is cyclic.

2. The answer to the previous question was no, and therefore we have to exhibit a minimal set (i.e., a set of minimal cardinality) of actions of S to remove so that the remaining schedule is conflict-serializable. To answer the question, we build the precedence graph associated to S, and we label each edge of such graph with the set of all conflicting pairs giving rise to such edge. The edges of the graph and their labels are:

   - $t_1 \rightarrow t_2$ labeled with $\{\langle w_1(A), w_2(A)\rangle, \langle r_1(A), w_1(A)\rangle\}$
   - $t_1 \rightarrow t_3$ labeled with $\{\langle w_1(A), r_3(A)\rangle, \langle r_1(C), w_3(C)\rangle\}$
   - $t_2 \rightarrow t_1$ labeled with $\{\langle r_2(A), w_1(A)\rangle\}$
   - $t_2 \rightarrow t_3$ labeled with $\{\langle w_2(A), r_3(A)\rangle\}$
   - $t_3 \rightarrow t_1$ labeled with $\{\langle w_3(B), w_1(B)\rangle\}$

   It is easy to see that no single action exists such that, removing the action from S, we get a schedule that is conflict serializable. On the other hand, if we remove $w_1(A)$, we remove all edges connecting t_1 with t_2 (in both directions), and by removing one of the actions in the set $\{w_3(B),\, w_1(B)\}$, or one of the actions in the set $\{r_3(C),\, w_3(C)\}$, we eliminate the cycle $t_1 \rightarrow t_3 \rightarrow t_1$ in the graph. It follows that, for example,

   $$\{w_1(A), w_3(B)\}$$

   is one minimal set (i.e., a set of minimal cardinality) of actions of S to remove so that the remaining schedule is not only view-serializable, but also conflict-serializable.

Problem 2 Consider again the schedule S shown in Problem 1, and tell whether there is a way to insert the "commit" command for every transaction in S such that the resulting schedule S' is ACR (Avoid Cascading Rollback). If the answer is yes, then show the schedule S', otherwise explain the answer.

Solution A schedule is ACR if no transaction reads from a transaction that has not committed yet. By referring to the schedule S of Problem 1, it is easy to see that t_3 reads from t_2 (indeed, $r_3(A)$ reads from $w_2(A)$), but t_2 cannot commit before t_3, since the action $r_2(D)$ appears after $r_3(A)$. It follows that there is no way to insert the "commit" commands for every transaction in S such that the resulting schedule is ACR.

Problem 3 We define WOB to be the class of schedules defined as follows: a schedule S on transactions $T = \{t_1, \ldots, t_n\}$ belongs to WOB if and only if (i) S is a complete schedule constituted only by "write" actions, and (ii) there exists a function $\psi : T \rightarrow \{1..n\}$ such that for every pair $\langle \alpha, \beta \rangle$ of conflicting actions in S, where β is an action of transaction t_k occurring in S after the action α of transaction t_h, we have that $\psi(t_k) > \psi(t_h)$.

1. Provide the definition of view-equivalence and view-serializability.
2. By using *only* the definition of view-equivalence and view-serializability, and the definition of WOB, prove or disprove that every schedule in WOB is view serializable.

Solution

1. A complete schedule S is view- serializable if there is a serial schedule S that is view-equivalent to S, where two schedules are view-equivalent if they have the same READS-FROM relation, and the same FINAL-WRITE set.

2. We now prove that every schedule in WOB is view serializable.

   By definition of WOB, we know that, if S is in WOB, then there exists a function $\psi : T \rightarrow \{1..n\}$ such that for every pair $\langle \alpha, \beta \rangle$ of conflicting actions in S, with β occurring after α in S, we have that $\psi(\beta) > \psi(\alpha)$. Let S' be the serial schedule on T reflecting the order represented by ψ, i.e., such that for all $i, j \in \{1..n\}$, transaction t_i appears before t_j in S' if and only if $\psi(t_i) < \psi(t_j)$.

   We show that S is view-equivalent to S'. We proceed by contradiction. Suppose that S and S' are not view-equivalent. Since S and S' are constituted only by "write" actions, and therefore the READS-FROM relation is empty for both, it follows that the FINAL-WRITE set of S is different from the FINAL-WRITE set of S'. This implies that there is an item A such that $w_h(A)$ is the final write on A in S, and $w_k(A)$ is the final write on A in S', with $h \neq k$. It follows that for the pair of conflicting actions $\langle w_k(A), w_h(A) \rangle$, $w_k(A)$ appears before $w_h(A)$ in S', whereas $w_k(A)$ appears after $w_h(A)$ in S. The fact that $w_k(A)$ appears before $w_h(A)$ in S' implies that $\psi(t_h) < \psi(t_k)$, and therefore the fact that $w_k(A)$ appears after $w_h(A)$ in S contradicts the hypothesis that $\psi : T \rightarrow \{1..n\}$ is such that for every pair $\langle \alpha, \beta \rangle$ of conflicting actions in S, with β occurring after α in S, we have that $\psi(\beta) > \psi(\alpha)$.

Problem 4 Consider the relation `MATCH(referee,date,hometeam,hostteam)`, updated monthly, that stores about 1.000.000 tuples, where each tuple $\langle r, d, h, t \rangle$ means that r was the referee of a match played at date d between teams h and t. We assume that (i) no referee can be in more than 20 matches with the same home team, (ii) the size of each page in our system is sufficient for storing 100 data entries, and 100 index entries, (iii) there is no good hash function for the search key $\langle$`referee`, `hometeam`$\rangle$, and (iv) the following are the most important queries on `MATCH`:

Query 1	**Query 2**
select `referee` from `MATCH` where `date` $> \alpha$ and `date` $< \beta$	select `hostteam`, `date` from `MATCH` where `referee` $= \gamma$ and `hometeam` $= \delta$

Tell which is the method for representing the relation `MATCH` you would choose in order to optimize the computation of the above queries, explaining in detail your answer. Also, tell how many pages are accessed during the execution of Query 2.

Solution Query 1 is an interval-based selection, that is well supported by a clustered tree-index. Relation `MATCH` is subject to updates, and therefore is not static. Hence, to support query 1, we define a clustered B*-tree index (called index 1) on search key ⟨`date`⟩, using alternative 1. To support query 2, we cannot use a hash index (because there is no good hash function for the search key ⟨`referee`, `hometeam`⟩), and therefore we define a second B*-tree index (called index 2) on search key ⟨`referee`, `hometeam`⟩, using alternative 2. Since index 1 is clustered, it follows that index 2 is unclustered.

Let us now turn our attention to computing the number of pages which are accessed during the execution of Query 2. The `MATCH` relation contains 1.000.000 tuples, every page contains 100 data entries (and, therefore, $F = 100$), and every data entry page is 67% full. Thus, 15.000 pages are required for the leaves of index 2. It follows that, in the execution of query 2, given a pair $\langle \gamma, \delta \rangle$ for the search key ⟨`referee`, `hometeam`⟩, we need $log_{100}\, 15.000 = 3$ page accesses to get to the first data entry. Since no referee can be in more than 20 matches with the same home team, in the worst case we need at most another page access to reach all relevant data entries, and at most 20 page accesses to reach the relevant data records. Therefore, the total number of page accesses in the execution of query 2 is as follows:

$$log_{100}\, 15.000 + 1 + 20 = 3 + 1 + 20 = 24.$$

Problem 5 Let T be a relation such that $B = 666.666.667$ (number of pages), and $R = 100$ (number of records per page). Assuming that $D = 15ms$ (time for accessing a page), $C = 100ns$ (time to process one record), and that we have a clustered B*-tree index using alternative 1, with search key equal to the key of T, and fan-out equal to 100, tell which is the time needed for inserting a new tuple in T, explaining in detail your answer.

Solution The method used for inserting a new record on T is as follows:

1. we first execute a selection based on equality to locate the page of the clustered B*-tree index where we have to insert (we assume such a page to be not full),
2. since we are using alternative 1, we simply execute the insertion.

The cost of executing the selection based on equality on the search key on the basis of the clustered B*-tree index is $D \cdot log_F(1.5B)$. The cost of locating where to insert in the page is $C \cdot log_2 R$. The cost of inserting into the page is C, and the cost of writing the page is D. Thus, the total time for inserting a new record in T is (15ms correspond to 15.000 ns):

$$\begin{gathered} D \cdot log_F(1.5 \cdot B) + C \cdot log_2 R + C + D = \\ 15.000 \cdot log_{100}(1.5 \cdot 666.666.667) + 100 \cdot log_2 100 + 100 + 15.000 \cong \\ 15.000 \cdot log_{100} 1.000.000.000 + 100 \cdot log_2 100 + 100 + 15.000 \cong \\ 15.000 \cdot 5 + 100 \cdot 7 + 100 + 15.000 = 90.800ns \end{gathered}$$