Data Management – exam of 26/01/2010

Solutions

Problem 1 Consider the following schedule

$$S = r_1(A)\, r_2(A)\, r_2(B)\, w_1(A)\, w_2(D)\, r_3(C)\, r_1(C)\, w_3(B)\, c_2\, r_4(A)\, c_1\, c_4\, w_3(C)\, c_3.$$

1. Tell whether S is accepted by the 2PL scheduler with exclusive and shared locks. If the answer is yes, then show the schedule obtained from S by adding suitable lock and unlock commands. If the answer is no, then explain the answer.
2. Tell whether S is strict or not, and explain the answer.
3. Tell whether S is recoverable or not, and explain the answer.
4. Tell whether S is conflict-serializable. If the answer is yes, then show a serial schedule that is conflict-equivalent to S. If the answer is no, then explain the answer.

Solution

1. The schedule S is accepted by the 2PL scheduler with exclusive and shared locks. The schedule obtained from S by adding suitable lock and unlock commands is as follows:

$$\begin{array}{c} sl_1(A)\, r_1(A)\, sl_2(A)\, r_2(A)\, sl_2(B)\, r_2(B)\, xl_2(D)\, u_2(A)\, xl_1(A)\, w_1(A) \\ w_2(D)\, sl_3(C)\, r_3(C)\, sl_1(C)\, r_1(C)\, u_1(C)\, u_2(B)\, xl_3(B)\, w_3(B)\, u_2(A) \\ c_2\, u_1(A)\, sl_4(A)\, r_4(A)\, c_1\, u_4(A)\, c_4\, xl_3(C)\, w_3(C)\, u_3(B)\, u_3(C)\, c_3. \end{array}$$

2. S is not strict, because transaction T_4 reads from T_1 that has not committed yet.
3. S is recoverable, because the only transaction reading from another transaction is T_4, that reads from T_1, and T_1 commits before T_4.
4. S is conflict-serializable, because the graph associated to S has edges $T_2 \to T_1, T_1 \to T_3, T_2 \to T_3, T_1 \to T_4$, and such a graph is acyclic. $\langle T_2, T_1, T_3, T_4 \rangle$ is a serial schedule that is conflict-equivalent to S.

Problem 2 Provide the definition of "monotone class of schedules". Using only the definition of monotone class of schedules, and the definition of 2PL schedules (with exclusive and shared locks), prove or disprove the following statement: the class of 2PL schedules (with exclusive and shared locks) is monotone.

Solution For a schedule S, $Trans(S)$ denotes the set of transactions present in S; for $T \subseteq Trans(S)$, $\Pi_T(S)$ denotes the projection of S onto T, i.e., the schedule S obtained from S by deleting all operations of the transactions that are not in T. A class E of schedules is called monotone if the fact that a schedule S is in E implies that for all $T \subseteq Trans(S)$, $\Pi_T(S)$ is in E too.

We now prove that the class of 2PL schedules (with exclusive and shared locks) is monotone. Let S be a 2PL schedule (with exclusive and shared locks). First, observe that every transaction in S is well-formed, and that S is legal. Now, let $T \subseteq Trans(S)$. Suppose that $\Pi_T(S)$ is not in the class of 2PL schedules (with exclusive and shared locks). This means that at least one of the following conditions holds:

1. one of the transactions in $\Pi_T(S)$ is not well-formed,

2. $\Pi_T(S)$ is not legal,

3. $\Pi_T(S)$ does not follow the 2PL protocol.

We show that none of these conditions holds.

1. Since $T \subseteq Trans(S)$, we know every transaction in $\Pi_T(S)$ is well-formed.

2. If $\Pi_T(S)$ is not legal, this means that there is a transaction t in $\Pi_T(S)$ that locks an element before a transaction t' (different from t) releases the lock on such element. It is easy to see that this happens also in S, because the order of actions in S is coherent with the order of actions in $\Pi_T(S)$. So, we get a contradiction.

3. If $\Pi_T(S)$ does not follow that 2PL protocol, then there exists a transaction t in T requiring a lock after an unlock. Since t is in S, this implies that S does not follow the 2PL protocol either, which is a contradiction.

Therfore, we conclude that $\Pi_T(S)$ is in 2PL, and that the class of 2PL schedules (with exclusive and shared locks) is monotone.

Problem 3 Consider the relation `PRODUCT(prodcode,size,year)`, and the relation `SOLD(prodcode,shopcode,cost)`, where `SOLD` stores information about products sold in shops, with the corresponding cost. We want to compute the equi-join of `PRODUCT` and `SOLD` on the attribute `prodcode`. We know that

- the products are 500.000,
- in every page used for the relation `PRODUCT` we have 10 tuples,
- in the average, every product is sold 20 times,
- we have a B^+-tree index with search key `prodcode` on `SOLD`, using alternative 1, with fan-out 10, and such that every leaf page contains 50 data entries.

If we use the index-nested loop join algorithm for computing the join, which is the cost of the computation in terms of the number of page accesses (ignoring the cost of writing the result)? Explain your answer in detail.

Solution Since the products are 500.000 and in every page used for the relation `PRODUCT` we have 10 tuples, It follows that there are 50.000 pages for `PRODUCT`. Since in the average, every product is sold 20 times, it follows that `SOLD` has 10.000.000 tuples. Since every leaf page of the index on `SOLD` contains 50 entries, the number of pages required to store all the entries is 200.000. Since the index uses alternative 1, and therefore is clustered, its pages have 67% of occupancy, and the leaf pages are 300.000.

The index-nested loop join algorithm in our case works as follows: it loads every page of the relation `PRODUCT`, and for every tuple of such relation, it uses the index to retrieve that tuples of `SOLD` satisfying the equi-join condition. Taking into account that the fan-out of the index is 10, and that in the average 20 such tuples, probably stored in 2 pages, will be retrieved, the cost of retrieving such tuples is $log_{10}\, 300.000 + 1$. We conclude that the number of page accesses is

$$50.000 + 500.000 \times (log_{10}\, 300.000 + 1) = 3.550.000$$

Problem 4 Provide the definition of the "immediate effect" method for writing values in secondary storage.

Solution In the immediate effect method, the update operations are executed immediately on the secondary storage after the corresponding records are written in the log, and therefore the buffer manager writes the effect of an operation by a transaction T on the secondary storage before writing the commit record of T in the log.

Problem 5 Suppose that our DBMS uses the "immediate effect" method for writing values in secondary storage, and a failure occurs when the log contains the following records (note that by "CKP" we denote a checkpoint record, and observe that we have *not* shown the active transactions in checkpoint records)

$B(T1); D(T1; O1; A1); B(T2); I(T2; O2; B2); B(T3); B(T4); D(T4; O3; B3); U(T1; O4; B4; A4);$
$C(T3); CKP; B(T5); U(T5; O5; B5; A5); B(T6); CKP; B(T7); C(T1); C(T4);$
$U(T7; O6; B6; A6); U(T6; O3; B7; A7).$

Describe in detail all the actions performed by the recovery manager to deal with the failure.

Solution Since our DBMS uses the "immediate effect" method for writing values in secondary storage, we can avoid redoing transactions. This means that we can use a simplified version of the warm restart procedure to deal with the failure.

1. The modified warm restart procedure begins by analyzing the log backward, until the most recent checkpoint. The active transactions at the the most recent checkpoint are T_1, T_2, T_4, T_5, T_6.
2. We set $s(UNDO) = \{T_1, T_2, T_4, T_5, T_6\}$. The set $s(REDO)$ is not needed.
3. We analyze the log forward, adding to $s(UNDO)$ the transactions with the corresponding begin record (i.e., T_7), and deleting from $s(UNDO)$ those with the commit record (i.e., T_1, T_4). At the end of this phase we have $s(UNDO) = \{T_2, T_5, T_6, T_7\}$.
4. Undo phase: we go backward through the log again, undoing the transactions in $s(UNDO)$ until the begin record of the oldest transaction in the set $s(UNDO)$ (which is T_2). This means undoing the following operations:
   - $U(T6; O3; B7; A7)$ – undoing this operation means doing nothing, since during the execution of the various operations before the failure, $U(T6; O3; B7; A7)$ had no effect, since O_3 was deleted by T_4,
   - $U(T7; O6; B6; A6)$ – undoing this operation means restoring the value $B6$ in O_6,
   - $U(T5; O5; B5; A5)$ – undoing this operation means restoring the value $B5$ in O_5,
   - $I(T2; O2; B2)$– undoing this operation means deleting O_2.
5. At this point, we do not need to go forward again, because, as we said before, we can avoid redoing transactions.