Fall 2010 Exam #3 Answers

Question 1.

(a) Conflicts $w_1(X)\, r_3(X)$, $r_1(Z)\, w_2(Z)$, $w_3(Y) r_2(Y)$. Conflict graph: $T_1 \rightarrow T_3$, $T_1 \rightarrow T_2$, and $T_3 \rightarrow T_2$.

This schedule is serializable because there is no cycle in the conflict graph. An equal serial schedule is given by T_1, T_3, T_2.

(b) Suppose we use only a single type of lock (only the exclusive lock).

$$lock_1(X)\, r_1(X)\, lock_2(Z)\, r_2(Z)\, w_1(X)$$
$$unlock_2(Z)\, lock_1(Z),\, r_1(Z)$$
$$unlock_1(X)\, lock_3(X) lock_3(Y) r_3(X)\, r_3(Y)\, w_3(Y)$$
$$unlock_3(Y) need\_lock_2(Y) - - fail[not\, possible\, r_2(Y)\, w_2(Z)]$$

As T_2 needs to unlock Z, it has entered the shrinking phase. At this point, T_2 cannot get any locks. If this schedule happened as given, at this point T_2 cannot have the lock to Y and it cannot later get this lock in the shrinking phase. So, this schedule is not possible.

(c) Serializable schedules are guaranteed to give the same results as a serial schedule, which by definition are considered correct schedules since no transaction changes the data of another transaction.

Question 2.

(a) No, as recovery first does a redo of all the actions to bring the database to the state it was at before the crash and then undoes only the work by uncommitted transactions.

(b) REDO is needed if the data pages changed by a committed transactions changes may not be recorded to the disk. This is possible. Suppose the log containing the update is written to disk, but power is lost before the data page is written. In this case, REDO will be neeeded.

UNDO is needed if data pages changed by uncommited transactions can be written to disk. Clearly, this is the case as the data page are written any time during the execution of the transaction to disk.

(c) REDO: If we follow the method in the book, only the pages changed for the committed transactions (T1,T3) should be redone. Of these, log entries for 10 (P1) and 14 (P3) are not written to disk and should be redone.

If we were redoing all the transactions in forward order (also an acceptable method), we would need to redo: 10, 14 and 15.

UNDO: We will undo all changes made by T2: 11 and 15. However, only 11 is recorded on disk and needs to be undone.

Question 3.

(a) Cost is PAGES(R) = 500

(b) We can only scan the index for condition on B, which would scan 1/3 of the leaf nodes (due to the selectivity of the condition on B), at a cost of 1 + 80/3 pages.

After scanning the index, we expect to find about: $60,000 * 1/200 * 1/3 * 1/10 = 10$ tuples that satisfy the conditions on B,C and D.

For these tuples, we will return the E attribute, hence we must read them from disk, at a cost of at most 10 additional disk reads.

Total cost is: $1 + 80/3 + 10$.

(c) Using the index on $R(C)$, we would need to scan 1+ceiling(20/200) = 2 nodes. We will find $60,000/200 = 300$ matching tuples. We need to read these tuples to check the remaining query conditions and return attribute E. The total cost of this is at most 300 pages. Hence, total cost is $2 + 300$.

(d) The best index for this query would be on attributes R(C,D,B,E) which allows us to query the smallest portion of the index and return all the attributes needed for this query. This way, we will not need to read any data pages to answer the query.

Question 4.

(a) Sorting PAGES(R) = 2,000 with M = 50. Step 1, sort the relation into 2000/50=40 sorted groups. Total cost: 4,000 page I/O (read once and write once).

Step 2, we need to merge the 40 groups, which we can do at one step (40 is less than 50). Total cost is one additional read at 2,000 pages.

Total cost = 6,000.

(b) Block nested loop join: read R once, and S two times (2,000/1,000). Total cost = 2000 + 2*5000=12,000.

(c) The cost of three way join is the first the join of R and S. Then, this is pipelined into the next join, causing T to be read 3 times with total additional cost of 30,000. Total cost of the whole join is 12,000+30,000= 42,000.

Question 5.

(a)
```
SELECT
    p1.id as person1
    , p2.id as person2
FROM
    person p1
    , person p2
WHERE
    p1.id < p2.id
    and p1.fatherid = p2.fatherid
    and p1.motherid = p2.motherid
    and p1.birthday = p2.birthday
```

If you wanted to check that twins birthdays are at most c days apart, you would write:

```
(p1.birthday <= p2.birthday
 and p2.birthday - p1.birthday <= interval (c days))
or  (p1.birthday > p2.birthday
 and p1.birthday - p2.birthday <= interval (c days))
```

Apparently, the Guinness world record for c is 87 days.

(b)
```
SELECT
    p.name
    , p.gender
FROM
    person p
    , person p2
WHERE
    p2.motherid = p.id
    or p2.fatherid = p.id
GROUP BY
    p.id
    , p.name
    , p.gender
HAVING
    count(*) >= 8
```