Implementation of Databases
Assignment #6

Due on January 24, 2017

Prof. Dr. rer. pol. Matthias Jarke

Dr. rer. nat. Christoph Quix

Submitted by:

Sanchit Alekh, Idil Esen Zulfikar

MSc. Software Systems Engineering

Problem 1

Serializability and Recoverability

Consider the following two transactions running concurrently:

$$T1 = r1(A)w1(A)r1(B)w1(B)$$
$$T2 = r2(B)r2(A)w2(A)w2(B)$$

1. Is conflict serializability guaranteed? Why or why not?

Solution

The question warrants that **T1** and **T2** are executed concurrently. If T1 and T2 are passed on as inputs to a serializer, the only legal and conflict serializable schedules we obtain are the ones which are serial schedules, i.e. *T1-T2* or *T2-T1.*
This is because both *T1 and T2* ask for write locks on the same database objects, i.e. A and B. Therefore, a lock-based serializer would ensure that one transaction waits for the other to finish its execution and release its locks, ultimately resulting in a serial schedule.
Therefore, *Conflict Serializability* is guaranteed in the case that the final sequence of execution of the transactions is the same as a serial schedule.

2. Is cascading rollback possible? If not, explain why not.

Solution

Cascading Rollbacks are **not possible**. This is because in effect, a locking-based scheduler will finally ensure that the two transactions are executed in a serial order, since both want to obtain write locks for the same objects.
Since the schedules are serial in nature, all transactions read from transactions which are already committed. Therefore, cascading aborts are avoided.

3. Is deadlock possible?

Solution

Yes, deadlock is **possible.** Consider a schedule like this:

$$S(1,2) = wl_1(A)r_1(A)w_1(A)wl_2(B)r_2(B)wl_1(B)r_1(B)wl_2(A)r_2(A)$$
$$w_2(A)w_1(B)c_1wu_1(A)wu_1(B)w_2(B)c_2wu_2(A)wu_2(B)$$

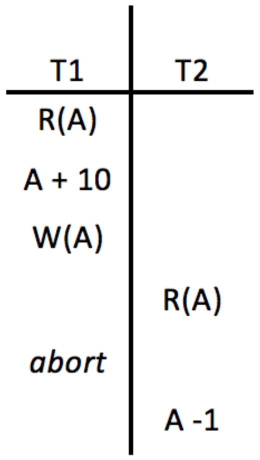
Problem 1 continued on next page. . .

In this schedule, T1 requests, and is granted a write lock on A. T2 requests a write lock on B, and is granted. Now, T1 requests a write lock on B to execute further. However, the lock is possessed by T2 and T1 has to wait till T2 completes its execution and releases its lock on B. At the same time, T2 wants to get a lock on A, which T1 possesses, and must wait for T1 to finish execution.
Therefore, we arrive at a situation where both the transactions are waiting on each other to finish execution in order to retrieve the required locks to execute further.
This is a classic example of a **deadlock**.

4. Explain with example what are “Dirty Reads” and “Phantom Reads” ?

Solution

Dirty reads happen when one transaction reads a database object that has been modified by another transaction but which has not been committed. So, the read is called as dirty read.It is a write-read conflict. For example, there are two transactions T_1 and T_2. T_1 reads and writes database object a then, before T_1 is committed, then T_2 reads a and T_1 is aborted. In such a case, a value T_2 read is an value which is not in the database and the database is inconsistent. As a real example, T_1 takes stock value of a product that is object a and the stock is 100. T_1 increases stock value to 110. Then T_2 reads the same stock value of product, the database object a, as 110. Then, T_1 is aborted, so a is restored to 100 and T_2 continues with value a is 110 that is not in the database and it makes database inconsistent. Such a schedule is illustrated in Figure 1.



T1	T2
R(A)	
A + 10	
W(A)	
	R(A)
abort	
	A -1

Figure 1: Dirty Read

The phantom read arises if a transaction retrieves a collection of objects but an object is inserted in the database, it would cause transactions sees the different result for collection of objects. As an example, let says T_1 reads all of products whose category_id is 1, and a new product whose category_id is 1 is added by a second transaction. So, T_1 does not see new added product and that causes T_1 to execute the transaction with missing object. If T_1 retrieves the collection of objects again, it sees different results than first collection of objects because one more product has been added. Phantom read can be seen in Figure 2.

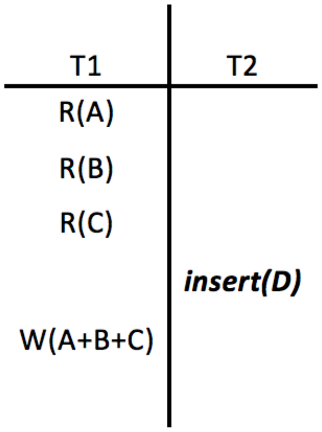


Figure 2: Phantom Read

1. Always lock root before starting (write lock)
2. Key value : minimum of the right child
3. Check if concerned node is safe in every step
4. For the actual insert/delete, don't forget the write operation
5. Pointers of new nodes must be updated. Include a write operation for it

Problem 2

B+ Tree Locking

Given the above B+-tree of degrees k=1 and k*=2, describe the steps involved in executing each of the following operations according to the simple tree locking algorithm.

1. Search Key = 76

Solution

To search for data key = 76, the following schedule is executed by transaction:

$rl(A), r(A), rl(B), ru(A), r(B), rl(F), ru(B), r(F), ru(F)$

First of all, root node A is locked with shared lock, the content of node is read and transaction determines to examine node B. In order to examine node B, transaction obtains shared lock on node B and immediately after it releases shared lock on A.

Then, content of node B is read and node F is determined to examine, so transaction puts a shared lock on F and immediately after it releases shared lock on B to maximize concurrency. Node F is read and searched value is reached, as a result, shared lock on F is unlocked.

2. Insert 82

Solution

The following schedule is executed by transaction for insertion of data key 82:

$$wl(A), r(A), wl(C), r(C), wu(A), wl(G), r(G), wu(C), w(G), wu(G)$$

Just to be safe, first do insertion/deletion, then release lock of parent

In order to insert 82, the root node A is locked with exclusive lock, transaction reads the node A and decide to examine node C for insertion. It obtains an exclusive lock on node C, reads node C and checks safety of node C. Node C is a safe node because it is not full, and the locked ancestors of node C should be released. Therefore, exclusive lock on node A is unlocked. The transaction determines to examine node G. It puts an exclusive lock on node G, reads node G and checks safety of node G. Since node G is not a full node, it is a safe node. Thus, the lock on node C that is ancestor of node G is released. The node G is the leaf node and it has space to write. The transaction modifies the node G and writes 82 between 81 and 83. After modification, it unlocks the node G.

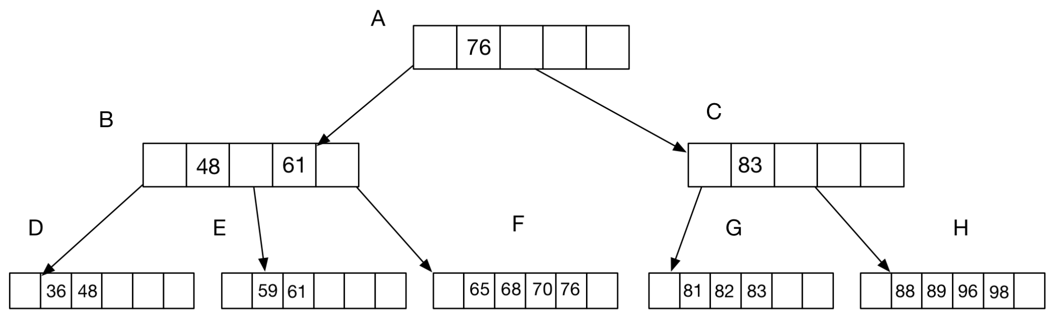


Figure 3: B+ Tree after inserting 82

3. Delete 61

Solution

The following schedule is executed to delete data key 61:

$$wl(A), r(A), wl(B), r(B), wu(A), wl(E), r(E), w(E), wl(D), r(D) merge(D, E, D),$$

$delete(E), wu(D), w(B), wu(B)$ `merge(D,E,D), w(B) to change key, delete(E), wu(B), w(D) to change ptr, wu(D)`

To delete data key 61, the transaction obtains an exclusive lock on root node A, reads node A and determines to examine node B. It puts an exclusive node on node B, reads node B and checks safety of node B. Since node B is not half empty, it is a safe node, so the transaction unlocks node A. Then, it decides to examine node E. It obtains an exclusive lock on node E and reads node E. The node E is not a safe node because it is half empty, so the transaction does not release the locked ancestors of node E. It means the lock on B should be held. The node E is the leaf node that is node to modify, so transaction writes the node E to delete 61 . This deletion causes merging with node D. Therefore, the transaction obtains an exclusive node on D, reads node D and merges D and E. After merging, node E is empty and it should be deleted. At the end, node D is 36,48,59 respectively. Modifications is done on leaf level and exclusive node on D should be released. But, the merging and deletion modifies node B that is ancestor of D and E. Transaction deletes 61 and updates 48 with 59. In the final step, it unlocks node B.

`Check safety of both siblings while redistribution, if any one of them is safe, just redistribute, don't merge`

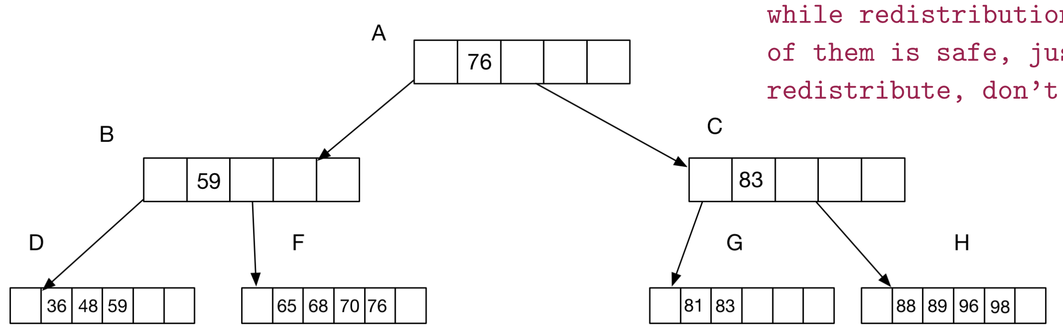


Figure 4: B+ Tree after deleting 61

Problem 3

Recovery

Consider the recovery scenario described in the following. There are three transactions T1, T2, T3 updating pages A, B, C. At the time of the crash, the log contains the following entries:

1. What is done during Analysis after the restart? Be precise about the points at which Analysis begins and ends and describe the contents of any tables constructed in this

phase.

Solution

Analysis begins at the point of the end of the checkpoint. At the checkpoint, all the tables are recalculated and updated.
In the analysis phase, the DBMS will go through the log file and update the values in the *Transaction Table* and the *Dirty Page Table.*
In this case, analysis will start at LSN = 7 and end at LSN = 9. The actions are as follows:

- Read LSN 7. Update LAST_LSN for Transaction T3 in Transaction Table to 7
- Read LSN 7. Check LAST_LSN for T1. Update it to 8
- Read LSN 9. Update the status of T1 to Commit

No changes are made to the *Dirty Page Table.* This is how the Transaction Table looks after the Analysis phase.

TRANSACTION_ID	LAST_LSN	STATUS
T1	8	commit
T2	2	abort
T3	7	active

Table 1: Transaction Table after Analysis Phase

2. What is done during Redo? Be precise about the modifications to any tables due to prcoessing of log records.

Solution

Redo starts from the point with the minimum LSN in the Dirty Page Table. In this case, it is 1. Therefore, Redo will start at LSN = 1.
Starting at LSN = 1, all the operations in the log are re-performed by the DBMS. No tables are further modified. Therefore LSN 1 upto 9 will be re-done.

3. What is done during Undo? Be precise about the modifications to the log and any tables due to processing of log records.

Solution

During *Undo*, all transactions which were active at the point of crash are undone, i.e. rolled back. In this case, according to the Transaction Table, only T3 is active.
Therefore, T3 will be undone and a compensation record will be written to the log.

Problem 3 continued on next page. . .

LSN	LAST_LSN	TRAN_ID	TYPE	PAGE_ID
10	7	T3	compensation	A

Table 2: Compensation Log Record for Undo operation

All the other entries in other tables stay the same. The compensation log record will be added after LSN = 9 and looks like the following: