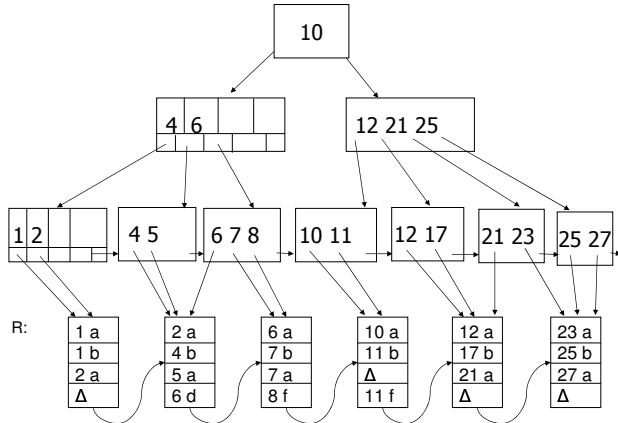


1 Indexes (30 minutes)

Given the following index sequential data file $R(v,x)$ and B+ tree indexing v :



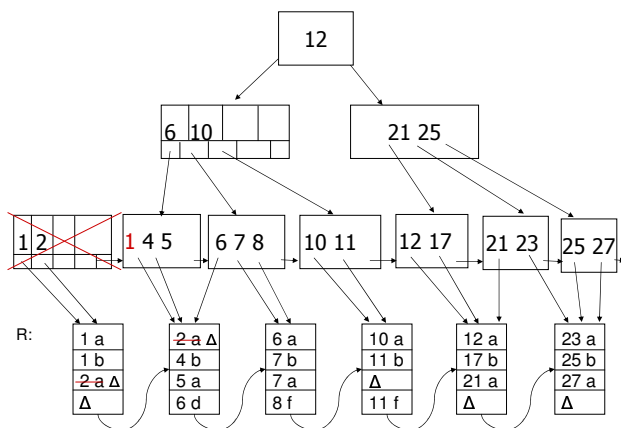
Assume the number of keys per block $N = 4$. You may assume that deletions in the data file are marked with a special value Δ that indicates the item is deleted. Records marked with Δ can be reused by a later insert.

Answer the following:

1.1 Deletion (7 minutes, 6 points)

Show the results of the following SQL transaction on the index and data file:

```
/* Transaction  $T_1$  starts here */
delete from R where  $v = 2$ ;
commit /*  $T_1$  */;
```



Scoring: One point each for: showing deletion to data file, maintaining a balanced tree, merging at the leaf, borrowing in the middle, adjusting the root, and correcting pointers. Minimum 2 points if resulting B-tree is correct, 1 point if data file is correct.

1.2 Logging (7 minutes)

For the following, you may assume undo, redo, or undo/redo logging. State which one you use.

1.2.1 Log (4 minutes, 4 points)

Show the log records that will be needed to ensure that you can properly recover if a failure occurs during or after the above SQL statement. Also show a valid point at which each disk block you show as modified (question 1.1) can be written to disk.

Undo: $\langle \text{Start } T_1 \rangle$, $\langle T_1, R, "2 a" \rangle$, $\langle T_1, R, "2 a" \rangle$, $\langle \text{flush log before database writes} \rangle$, $\langle \text{database output before commit} \rangle$, $\langle \text{commit} \rangle$.

Scoring: one point for a decent attempt, and one point each for: Ensuring the log flush occurs at an acceptable point, showing an acceptable time for the data write to occur, and for log entries that give the ability to recover the delete.

1.2.2 Index logging (3 minutes, 2 points)

We have talked of logging database updates. Briefly discuss the following:

1. I claim it is not necessary to log changes to the index. Why?

The log can be rebuilt from the data file, as long as the data file is correct. Logging just changes to data will ensure that we can recover the data file to the correct state.

2. Is there a reason you might want to log changes to the index?

At recovery, we would need to rebuild the index. If we log changes, we can just recover the changes instead of rebuilding the index.

1.3 Locking (18 minutes)

Assume record-level locking, with two lock types (Shared and eXclusive). Assume 2-phase locking.

1.3.1 The wrong way (11 minutes)

Since I have an index on $R(v)$, I can go directly to the records with $v=2$. Thus I only need two locks: eXclusive locks on the items with $v=2$.

1. **Demonstrate that this is wrong (6 minutes, 3 points)**

The preceding statement (T_1 only needs two locks) is incorrect. Give a transaction T_2 (sequence of reads and writes) and sequence of locks/unlocks for both T_1 and T_2 that does not violate 2-Phase Locking, but is not conflict serializable.

T_2 : update R set $v=v+1$ where ($v=1$ or $v=2$) and ($x='a'$);

Sequence:

$X_2(2a_1)$, $X_2(2a_2)$, $r_2(2a_1)$, $r_2(2a_2)$, $w_2(2a_1, 3)$, $w_2(2a_2, 3)$, $X_2(1a)$, $u_2(2a_1)$, $u_2(2a_2)$,

Transaction T_1 runs here, and does nothing since there are no items with $v=2$.

$r_2(1a)$, $w_2(1a, 2)$, $u_2(1a)$.

If $T_1 \rightarrow T_2$, then the items 2a should be removed, not changed to 3a. If $T_2 \rightarrow T_1$, then the item currently marked 1a should be deleted. Neither happens - this isn't serializable.

Scoring: One point for an idea, one if your sequence meets 2PL, one if it is not serializable.

2. **Fix the problem (4 minutes, 2 points)**

What locks should T_1 have?

To delete all items where $v=2$, we must read the relation. Therefore we need a shared lock on the entire relation.

Scoring: one point for locking index or all items, one for shared lock.

1.3.2 Index locking (6 minutes, 2 points)

Does the presence of an index affect the locks we need? Briefly explain your answer (3-5 sentences.)

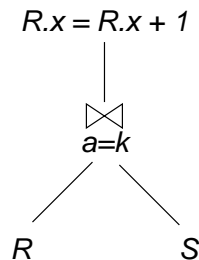
We may need to lock the index to perform updates to the index, otherwise another transaction may falsely assume that items that aren't visible don't exist (e.g., if we take the right branch from the root to find 11, but the middle row has already been changed so that 11 is found from the left branch).

2 Query Plan (45 minutes)

Use the following two relations:

| | |
|------------------|-----------------------|
| $R(a, x)$ | $S(\underline{k}, z)$ |
| Blocks(R) = 10 | Blocks(S) = 100 |
| Tuples(R) = 100 | Tuples(S) = 2000 |
| Values(R,a) = 10 | Values(S,k) = 2000 |
| Values(R,x) = 5 | Values(S,z) = 100 |

The following query plan contains an update ($x = x + 1$) that increments the value of x in relation R for the tuples that are returned by the result of the join.



Assume that an ID of each tuple is passed through the query. The ID enables direct access to the tuple. If a tuple contains values from two relations (e.g., a cross product or join), the ID of both tuples is included. Thus when we get to the update, we can directly access the tuple that needs to be updated. The cost of storing the ID is negligible (i.e., R with tuple IDs attached still takes 10 blocks.)

You should also assume that buffer space is constrained: You can only hold 5 blocks in memory.

2.1 Nested Loop Join (10 minutes, 4 points)

Assume that we perform a nested loop join, where the outer loop is relation R (we choose an item from R, then run through all tuples in S). Estimate the I/O cost (number of blocks read or written). If you just estimate sizes of intermediate results (as opposed to total I/Os), you will get partial credit.

We can read in up to 4 blocks from R at a time, then we need to run through all blocks of S (1 point), giving $10(R) + 3 \cdot 100(S) = 310$ reads (1 point). We will also need to write each block with changes (1 point). Since $V(R,a) < V(S,k)$, we assume all a values in R are contained in S(k). Therefore every value is changed, so we need to write each block. This can be done as soon as we are done processing the relation S in the join (the operation is non-blocking), so we only need write each block once, and don't need any more reads, for a total of $310+10=320$ I/Os (1 point).

2.2 Alternate query processing (20 minutes, 8 points)

Give way of processing the query (e.g., different query plan, different join algorithm, etc.) that you think is better, and estimate the I/O cost for your approach. Assume no indexes (unless you build them, in which case you must include the cost of building indexes in your I/O cost).

A hash join would work as well, since $B(R) \leq 25$. However, the I/O cost is higher – $3 \cdot (10 + 100)$ – even before adding the cost of writing. Since we no longer have the original block of relation R in memory after the join (as we did with the nested loop), it will take another read and write to get, update, and write the original block.

Scoring: two to three points for your join algorithm, one point for getting the update right, one point for commenting on efficiency, four points for analyzing it (scored as above).

2.3 SQL update (10 minutes)

1. SQL update (5 minutes, 2 points)

Give an SQL query that would be correctly answered by the above query plan.

update R set $x = x + 1$ where a in (select k from S);

One point for a join (the “in” does a join), one for an update.

2. Correctness (5 minutes, 1 point)

Would your query plan still perform the correct updates for your SQL query if $S(k)$ was not a key?

The query plan would have to be careful to update each tuple of R only once. Obvious algorithms (e.g., in the nested loop, just increment the value of R when a matching tuple in S is found) would not give correct results.

3 Undo vs. Redo Logging (40 minutes)

Undo logging requires that the log record must be written to disk before the database is changed on disk, and all database changes must be written before the commit. Redo logging requires that the log records and commit be written to disk before any database changes are written to disk.

Keeping these in mind, answer the following. For each, give a brief (two to three sentence) justification of your answer. Remember that you are comparing undo and redo logging – not undo/redo logging.

3.1 Fast Response Time (5 minutes, 3 points)

Which logging method is best if the goal is a fast response time. Response time is defined as the time from when the transaction first reaches the system until it becomes persistent, i.e., the first point at which a crash followed by recovery will result in a system that shows the results of the transaction.

To be persistent, the commit must be written to the log (otherwise the changes will be undone at recovery) (1 point). Undo logging requires that the log be flushed to disk then the data output, then the commit (1 point), redo only requires the log be flushed to disk (1 point). Therefore the commit appears on disk with fewer I/Os with redo logging (1 point).

3.2 High Throughput (5 minutes, 3 points)

Which logging method would give the highest throughput, i.e., the most transactions per second?

Again redo logging never requires that data be written to disk, only that the log be written (1 point). For two reasons, this will give higher throughput. First, there may be fewer I/Os: transactions

may be able to update “dirty” pages before they are written out, thus saving an I/O (1 point). Second, there is more flexibility in scheduling I/Os: Log records will likely be sequential, and data writes can be scheduled when the disk head is already close to the appropriate location (1 point). Thus redo logging will give higher throughput.

3.3 High Availability (5 minutes, 3 points)

Which method would give the highest system availability? Assume failures are “fail-stop”, e.g., a power outage, and that the actual time when the computer is not operating is the same for each logging method. Availability is the time that the computer is ready to process transactions, which may be smaller than the time when the power is on. In other words, which approach gives the smallest recovery time.

Redo logging requires that all actions that had not committed before the last checkpoint be redone (1 point). Undo only requires that uncommitted transactions be undone (1 point). It is likely that there will be many more committed transactions than uncommitted, so undo logging will be faster (1 point).

3.4 Demonstrate your answer (15 minutes, 6 points)

For **one** of questions 3.1, 3.2, or 3.3, give a detailed example demonstrating the correctness of your answer. In other words, provide a sequence of transactions, reads/writes performed by those transactions, I/Os required, and expected time for those I/Os that demonstrates conditions under which your answer is correct.

Throughput undo: $start_1, R_1(A), W_1(A), Log(T_1, A_0), start_2, R_2(A), W_2(A),$
 $Log(T_2, A_{T_1}), flush(log), output(A), commit_1, commit_2, flush(log).$

redo: $start_1, R_1(A), W_1(A), Log(T_1, A_{T_1}), start_2, R_2(A), W_2(A), Log(T_2, A_{T_2}), commit_1, commit_2,$
 $flush(log), output(A).$

Note that in each case I have delayed outputting data and flushing the log as long as possible, and redo still saves flushing the log once.

Scoring: 1 point for undo sequence, +1 if valid, 1 for redo, +1 if valid, 1 for noting difference, 1 for demonstrating that their must be such a difference.

Expectations

My expectation is that an A student should get at least 41 of the possible 49 points. A B student should get at least 36. I would consider 24 passing (C).