**TRUE OR FALSE**

1. Write Ahead Logging describes a protocol where updated pages must be written to disk before a crash
   1. FALSE
2. During a transaction abort, we redo all updates made by the transaction
   1. FALSE
3. When undoing updates of a transaction, CLR record is logged to describe the undoing of a posterior update.
   1. FALSE
4. In ARIES, UPDATE log records contain the information of the previous state of the page
   1. TRUE
5. The recovery manager is responsible for Atomicity and Durability, as defined by the ACID acronym.
   1. TRUE

**FORCE and STEAL**

We run two transactions, T1 and T2. The transaction table and the dirty page table at the time of the checkpoint are both empty. We know that the system uses page-level locks. We access some system metadata and observe the following:

* On disk, P1 has a pageLSN of 40.
* On disk, P2 has a pageLSN of 50.
* On disk, P3 has a pageLSN of 30.

We also find the following log:

|  |  |
| --- | --- |
| **LSN** | **Record** |
| 0 | BEGIN\_CHECKPOINT |
| 10 | END\_CHECKPOINT |
| 20 | UPDATE: T1 writes P2 |
| 30 | UPDATE: T1 writes P3 |
| 40 | UPDATE: T2 writes P1 |
| 50 | UPDATE: T2 writes P2 |
| 60 | COMMIT: T1 |
| 70 | COMMIT: T2 |

1. The system might be use two-phase locking.

a. True

b. False

c. Not enough information

a. At LSN 20, T1 writes an UPDATE log record with P2. Then, T2 writes an UPDATE log record with P2 before T1 writes a COMMIT log record. So, the system cannot be using strict 2PL.

1. The system uses a FORCE policy.

a. True

b. False

c. Not enough information

c. In this particular case, T1 forces P3 to disk before committing, and T2 forces P1 and P2 to disk before committing. We do not know exactly when T2 forces P2 to disk, nor any of the other updates. Given this information, we see that the system could be using a FORCE policy, but we cannot be certain (it’s possible that the writes happened before the commit log record just by good luck).

1. The system doesn’t use a STEAL policy.

a. True

b. False

c. Not enough information

c. Not enough information. The pageLSN indicate the last update pushed to disk, but we do not know if the updates were written to disk before or after the commits.

1. The system doesn’t use the ARIES recovery algorithm.

a. True

b. False

a. True, because the ARIES we covered uses strict 2PL, and this system does not use strict 2PL, so this system cannot be using ARIES.

**Recovery**

Consider the following log. Some of the records have been omitted. The system crashes immediately after LSN 100 and begins recovery. During analysis, we recreate the transaction table and dirty page table shown below.

|  |  |
| --- | --- |
| **LSN** | **Record** |
| 0 | BEGIN\_CHECKPOINT |
| 10 | END\_CHECKPOINT |
| 20 | UPDATE: T1 writes P1 |
| 30 | UPDATE: T1 writes P3 |
| 40 | UPDATE: T2 writes P2 |
| 50 | ABORT: T1 |
| 60 | ??? |
| 70 | ??? |
| 80 | END: T1 |
| 90 | UPDATE: T2 writes P1 |
| 100 | COMMIT: T2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Transaction Table** | | | **Dirty Page Table** | |
| **Transaction** | **lastLSN** | **Status** | **PageID** | **recLSN** |
| T2 | 100 | Committing | P1 | 20 |
|  |  |  | P2 | 40 |
|  |  |  | P3 | 30 |

1. Which of the following sequences of missing log records (??? in the image above) is correct?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a.   |  |  | | --- | --- | | **LSN** | **Record** | | 60 | CLR: T1 LSN 50 | | 70 | CLR: T1 LSN 30 | | b.   |  |  | | --- | --- | | **LSN** | **Record** | | 60 | CLR: T1 LSN 50 | | 70 | CLR: T1 LSN 20 | |
| c.   |  |  | | --- | --- | | **LSN** | **Record** | | 60 | CLR: T1 LSN 30 | | 70 | CLR: T1 LSN 20 | | d.   |  |  | | --- | --- | | **LSN** | **Record** | | 60 | CLR: T1 LSN 20 | | 70 | CLR: T1 LSN 30 | |

c. When we abort a transaction, we perform CLR’s in reverse order.

1. Which of the following sequences of actions will occur during the REDO phase? "Orig LSN" denotes the LSN of the original action in the log above, to improve readability. (??? indicates your answer to 5.)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a.   |  |  | | --- | --- | | **Orig LSN** | **Record** | | 20 | UPDATE: T1 writes P1 | | 30 | UPDATE: T1 writes P3 | | 40 | UPDATE: T2 writes P2 | | 60 | ??? | | 70 | ??? | | 90 | UPDATE: T2 writes P1 | | b.   |  |  | | --- | --- | | **Orig LSN** | **Record** | | 20 | UPDATE: T1 writes P1 | | 30 | UPDATE: T1 writes P3 | | 40 | UPDATE: T2 writes P2 | |
| c.   |  |  | | --- | --- | | **Orig LSN** | **Record** | | 20 | UPDATE: T1 writes P1 | | 30 | UPDATE: T1 writes P3 | | 40 | UPDATE: T2 writes P2 | | 60 | ??? | | 70 | ??? | | d.   |  |  | | --- | --- | | **Orig LSN** | **Record** | | 20 | UPDATE: T1 writes P1 | | 30 | UPDATE: T1 writes P3 | | 40 | UPDATE: T2 writes P2 | | 90 | UPDATE: T2 writes P1 | |

a. We start redoing actions from the first recLSN in the dirty page table (20). We need to redo all UPDATE log records and CLR log records whose pages are in the dirty page table with recLSN <= LSN.

1. Which of the following sequences of log records will be written during the UNDO phase?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a.   |  |  | | --- | --- | | **LSN** | **Record** | | 200 | CLR: T2 LSN 100 | | 210 | CLR: T2 LSN 30 | | 220 | END: T2 | | b.   |  |  | | --- | --- | | **LSN** | **Record** | | No logs written during UNDO | | |
| c.   |  |  | | --- | --- | | **LSN** | **Record** | | 200 | CLR: T2 LSN 110 | | 210 | CLR: T2 LSN 100 | | 220 | CLR: T1 LSN 50 | | 230 | CLR: T1 LSN 40 | | 240 | CLR: T2 LSN 30 | | 250 | END: T2 | | 260 | CLR: T1 LSN 20 | | 270 | END: T1 | | d.   |  |  | | --- | --- | | **LSN** | **Record** | | 200 | CLR: T2 LSN 110 | | 210 | CLR: T2 LSN 100 | | 220 | CLR: T2 LSN 30 | | 230 | END: T2 | |

b. This is a trick question! Since we log a END: T2 during the REDO phase of the previous problem, we actually have no active transactions that we need to UNDO. In this particular case, no log records are written during UNDO!