# horizontal lineDatabase R&D Exercise

Assignment 9

I confirm that this is my own work and that use of material from other sources, including the Internet, has been properly and fully acknowledged and referenced.

|  |  |
| --- | --- |
| Name: | Pang, Jinhao |
| Date: | 2022.12.14 |
| NYU ID: | N19475049 |
| Course Section Number: | csci-ga.2433-001 |



**Total in points** (100 points total): \_\_\_\_\_

**Professor’s Comments:**

|  |
| --- |
|  |

20.19. Write a program to create all possible schedules for the three transactions in Figure **20.8(a), and to determine which of those schedules are conflict serializable and which are not. For each conflict-serializable schedule, your program should print the schedule and list all equivalent serial schedules.**

Graphical user interface, text, application

Description automatically generated

The idea to find all equivalent serial schedules is that we want to find a path in a directed graph to go through all the nodes. If there are no edges between two nodes, then let them have an undirected edge when constructing the graph. Then there must exist a path covering all the nodes when the schedule is conflict serializable.

**20.23. Consider the three transactions T1, T2, and T3, and the schedules S1 and S2 given below. Draw the serializability (precedence) graphs for S1 and S2, and state whether each schedule is serializable or not. If a schedule is** **serializable, write down the equivalent serial schedule(s).**

**T1: r1 (X); r1 (Z); w1 (X);**

**T2: r2 (Z); r2 (Y); w2 (Z); w2 (Y);**

**T3: r3 (X); r3 (Y); w3 (Y);**

**S1: r1 (X); r2 (Z); r1 (Z); r3 (X); r3 (Y); w1 (X); w3 (Y); r2 (Y); w2 (Z);**

**w2 (Y);**

**S2: r1 (X); r2 (Z); r3 (X); r1 (Z); r2 (Y); r3 (Y); w1 (X); w2 (Z); w3 (Y);**

**w2 (Y);**

S1:

Diagram, schematic

Description automatically generated

The schedule is serializable. T3->T1->T2

S2:

Diagram, schematic

Description automatically generated

The schedule is not serializable.

**21.27. Why is two-phase locking not used as a concurrency control method for**

**indexes such as B+-trees?**

The two-phase locking makes sure serializability for transactions to behave correctly. While for B+-tree, there is no need to ensure the order as long as the data is accurate. The insertion will not be affected by the order.

**21.29. The MGL protocol states that a transaction T can unlock a node N, only if none of the children of node N are still locked by transaction T. Show that without this condition, the MGL protocol would be incorrect.**

Without his condition, node N will be unlocked even though there is at least one child locked by a transaction. Then when accessing the same data through other transactions, it may lead to a conflict that the data have not finished processing yet. MGL makes sure all the locks held by T are released after T completes to prevent conflicts.

**22.23. Figure 22.6 shows the log corresponding to a particular schedule at the point of a system crash for four transactions T1, T2, T3, and T4. Suppose that we use the immediate update protocol with checkpointing. Describe the recovery process from the system crash. Specify which transactions are rolled back, which operations in the log are redone and which (if any) are undone, and whether any cascading rollback takes place.**

Recovery process in the text book:

1. Use two lists of transactions maintained by the system: the committed

transactions since the last checkpoint and the active transactions.

2. Undo all the write\_item operations of the active (uncommitted) transac- tions, using the UNDO procedure. The operations should be undone in the reverse of the order in which they were written into the log.

3. Redo all the write\_item operations of the committed transactions from the log, in the order in which they were written into the log, using the REDO procedure defined earlier.

Rollback: T2, T3

Redone: [write\_item, T1, D, 20, 25], [write\_item, T4, D, 25, 15], [write\_item, T4, A, 30, 20]

Undone: [write\_item, T2, B, 12, 18], [write\_item, T3, C, 30, 40], [write\_item, T2, D, 15, 25]

Cascading rollback: No

**22.24. Suppose that we use the deferred update protocol for the example in Figure 22.6. Show how the log would be different in the case of deferred update by removing the unnecessary log entries; then describe the recovery process, using your modified log. Assume that only REDO operations are applied, and specify which operations in the log are redone and which are ignored.**

Difference:

|  |
| --- |
| [start\_transaction, *T*1] |
| [read\_item, *T*1, *A*] |
| [read\_item, *T*1, *D*] |
| [write\_item, *T*1, *D*, 20, 25] |
| [commit, *T*1] |
| [checkpoint] |
| [start\_transaction, *T*2] |
| [write\_item, *T*2, *B*, 12, 18] |
| [start\_transaction, *T*4] |
| [write\_item, *T*4, *D*, 25, 15] |
| [start\_transaction, *T*3] |
| [write\_item, *T*3, *C*, 30, 40] |
| [write\_item, *T*4, *A*, 30, 20] |
| [commit, *T*4] |
| [write\_item, *T*2, *D*, 15, 25] |

Recovery procedure:

REDO all the WRITE operations of the committed transactions from the log, in the order in which they were written into the log, i.e., [write\_item, T4, D, 25, 15], [write\_item, T4, A, 30, 20]

Redone: [write\_item, T4, D, 25, 15], [write\_item, T4, A, 30, 20]

Others will be ignored.

**22.26. How are log sequence numbers used by ARIES to reduce the amount of REDO work needed for recovery? Illustrate with an example using the information shown in Figure 22.5. You can make your own assumptions as to when a page is written to disk.**

Procedure in textbook:

To reduce the amount of unnecessary work, ARIES starts redoing at a point in the log where it knows (for sure) that previous changes to dirty pages have already been applied to the database on disk. It can determine this by finding the smallest LSN, M, of all the dirty pages in the Dirty Page Table, which indicates the log position where ARIES needs to start the REDO phase. Any changes corresponding to an LSN < M, for redoable transactions, must have already been propagated to disk or already been overwritten in the buffer; otherwise, those dirty pages with that LSN would be in the buffer (and the Dirty Page Table). So, REDO starts at the log record with LSN = M and scans forward to the end of the log.

The smallest LSN in the Dirty Page Table is 1, so the REDO starts from Lsn=1. Assume page C is written to disk when T2 is update, the Lsn=7. Then when REDO from Lsn=1<7, we do not need to change here and proceed to Lsn=2, if Lsn of B is less than two, it will be update with the change by Lsn=2. Similarly to Lsn=6 for page A. When comes to Lsn=7, no need to change since 7=7.