## Chapter 6.4 Undo/Redo Logging File

*Background:*

In this Chapter, we already saw two different Logging Schema, the difference here is just that when adding Modification Logging Record, *Undo Schema is using the OLD value while Redo Schema is using the New value*. Of course, each one has it’s own defect:

* ***Undo Logging*** - Write the data back right after the end of Transaction which has not been updated to the Disk, and this increases the necessary Disk I/O.
* ***Redo Logging*** - Before we commit and flush the Logging Record, we need to keep any modifications of the Blocks into Buffer Area, which may increase the average Buffer Areas.
* Database Element is not integrity block or block collection, there will be ***Contradiction for Undo Logging and Redo Logging*** during the process of Checkpoint.
  + *There has one buffer area which includes database element A that has been modified by committed Transaction.*
  + *There is another situation that in the same Buffer Area, exists another Database Element B which has been modified by UNCOMMITTED Logging Record.*
  + *Analysis from the first situation, Database Changes should be copied to the Disk. While the second situation, Database Changes should not be copied into the Disk according to the RU1 which requires that the Transaction Logging Record should be updated to the Disk and after that, Database Changes can be updated to the Disk.*

*Introduction:*

In this Chapter, we would look *Logging Type of undo/redo*, which can be used to maintain much more information (which is still one type of cost.), provide much more flexibility on the motion sequence.

Chapter 6.4.1 Undo/Redo Rule

*Definition:*

Undo/Redo Logging is just the same as other two types of logging, of course there has one exception. When there has any modifications on Database Element, one logging record just as <T, X, v, w> needs to be added into the Logging File which means that the Transaction T has changed the value of Database Element X, before updates, the value of X equals to v while after updates, the value of X equals to w. The Undo/Redo Logging Record should obey the constraint below:

*UR1: Before any modifications about Database Element has been updated to Disk because of Transaction T, the Update Record <T, X, v, w> needs to appear on the Disk.*

Undo/Redo Logging Rule would have effect on the same constraint for Undo Logging File and Redo Logging File. More specifically, *<COMMIT T> Logging Record would appears before/after modifications on any of database element updates on the Disk.*

*Example:*

The table below is one variant about Transaction T, and there has some changes on Transaction Sequences. Attention that, Update Logging Record should include the OLD and NEW value in the Modification. Also, it writes the Logging Record <COMMIT T> before Database Element A and B have been written into the Disk.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Step* | *Action* | *t* | *M - A* | *M - B* | *D - A* | *D - B* | *Logging* |
| *1* |  |  |  |  |  |  | *<START T>* |
| *2* | *READ(A, t)* | *8* | *8* |  | *8* | *8* |  |
| *3* | *t := t \* 2* | *16* | *8* |  | *8* | *8* |  |
| *4* | *WRITE(A, t)* | *16* | *16* |  | *8* | *8* | *<T, A, 8, 16>* |
| *5* | *READ(B, t)* | *8* | *16* | *8* | *8* | *8* |  |
| *6* | *t := t \* 2* | *16* | *16* | *8* | *8* | *8* |  |
| *7* | *WRITE(B, t)* | *16* | *16* | *16* | *8* | *8* | *<T, B, 8, 16>* |
| *8* | *FLUSH LOG* |  |  |  |  |  |  |
| *9* | *OUTPUT(A)* | *16* | *16* | *16* | *16* | *8* |  |
| *10* |  |  |  |  |  |  | *<COMMIT T>* |
| *11* | *OUTPUT(B)* | *16* | *16* | *16* | *16* | *16* |  |

Chapter 6.4.2 Recover by using Undo/Redo Logging File

When we need to use Redo/Undo Logging to recover, it allows us to Update Old value to retreat the Transaction T by using the information that we already have. It allows us to Update New value to redo Transaction T. *The Strategy of Undo/Redo Logging is:*

1. *Redo all COMMIT Transactions according to Front - To - Back Sequence.*
2. *Cancel all UNCOMMIT Transactions according to Back - To - Front Sequence.*

***Attention, these two situations all need to be considered:***

Since Undo/Redo Logging Records has great flexibility on Copy Sequence about whether to copy COMMIT Transaction to Disk first or copy the Data Modifications to the Disk first, *we allow Partial/Whole Committed Transactions do not stay on the Disk, also allow Partial/Whole UNCOMMITTED Transactions stay on the Disk.*

*Example:*

For the table below, the crash happens in the different place of Sequence, then Recovery Method is totally different.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Step* | *Action* | *t* | *M - A* | *M - B* | *D - A* | *D - B* | *Logging*  *(Memory)* | *Logging*  *(Disk)* |
| *1* |  |  |  |  |  |  | *<START T>* | *<START T>* |
| *2* | *READ(A, t)* | *8* | *8* |  | *8* | *8* |  |  |
| *3* | *t := t \* 2* | *16* | *8* |  | *8* | *8* |  |  |
| *4* | *WRITE(A, t)* | *16* | *16* |  | *8* | *8* | *<T, A, 8, 16>* | *<T, A, 8, 16>* |
| *5* | *READ(B, t)* | *8* | *16* | *8* | *8* | *8* |  |  |
| *6* | *t := t \* 2* | *16* | *16* | *8* | *8* | *8* |  |  |
| *7* | *WRITE(B, t)* | *16* | *16* | *16* | *8* | *8* | *<T, B, 8, 16>* | *<T, B, 8, 16>* |
| *8* | *FLUSH LOG* |  |  |  |  |  |  |  |
| *9* | *OUTPUT(A)* | *16* | *16* | *16* | *16* | *8* |  |  |
| *10* |  |  |  |  |  |  | *<COMMIT T>* | *<COMMIT T>* |
| *11* | *OUTPUT(B)* | *16* | *16* | *16* | *16* | *16* |  |  |

* *Assume that crash happens right after Logging Record <COMMIT T> has been Flushed to the Disk (From Main Memory to Disk.)*
  + At this time, we decide to write 16 into A and B on the Disk because of the Flexibility of Undo/Redo Transaction. Because of the actual sequence of Transaction, variable A already equals to 16, while B maybe do not - this is totally up to whether crash happens before step 11 or after 11. If the crash happens right after step 11, then B should equals to 16, otherwise B should equals to 8. Since the crash happens right after <COMMIT>, we neglect the current value of B, update 16 to the Main Memory and Disk.
* *Assume that the crash happens before Logging Record <COMMIT T> reaches the Disk and Transaction T would be considered as Uncommitted Transaction.*
  + The Value of A and B all equals to 8 and here no matter whether the current value of B is, 8 or 16. Normally, we can not make sure whether it is necessary to recover A and B, so we always retreat the Operation.

Chapter 6.4.3 Checkpoint in Undo/Redo Logging File

Non-Static Undo/Redo Logging Checkpoint is much simpler than other Logging to some extent.