## Chapter 6.3 Redo Logging

*Defect About Undo Logging File:*

Before all Data Modifications have been updated to the Disk, we can not COMMIT current Transaction. However, makes the Database Modifications only exist in the Main Memory can help us save Disk I/O. As long as there has Logging File which can help repair Logging File, then this is safe.

Definition:

*Undo Schema:*

Prevent the need to write back any Database Element before COMMIT the Transaction.

*Main Difference between Redo and Undo Logging File:*

1. Undo Logging Record eliminates any influences from unfinished Transaction and neglect any submitted Transactions; While Redo Logging File is to neglect any unfinished Transaction and repeat the procedure to submit all changes for commit Transactions.
2. Undo Logging Record asks us to update all Database Elements to the Disk before COMMIT Logging Record reaches the Disk. While Redo Logging File is to ask Update all COMMIT Logging Records to the Disk and after that update all Database Element Changes to the Disk.

*Conclusion:*

* When using Undo Schema to obey Rule U1 and U2, then we need to use *OLD Value of Database Elements* to revert all changes in the Main Memory and Disk.
* When using Redo Schema, then we need to use *New Value of Database Elements* to update to the Main Memory and Disk.

Chapter 6.3.1 Redo Logging Rule

*Principle:*

In Redo Logging Record, Logging Record <T, X, v> means ‘Update the New Value v for Variable X’, in this record, you can tell that there has no Old Value. Each time when Transaction T tries to modify Database Element X, we must write one Record just as <T, X, v>.

*Rule:*

For Redo Logging File, we can tell that Database Data and Logging File Record can be used by ‘Redo Rule’ to describe the Sequence under which Data and Logging Record reach the Disk - which is called *Logging - Rule - Write - First*.

*R1: Before any Database Data X modifications reach the Disk, first ensure to write one Logging Record about modify all Logging Records, including Update Statement <T, X, v> and <COMMIT T>, they need to show up in the Disk. After that any modifications about Database Element Modifications can show up in the Disk.*

Logging Record <COMMIT T> appears in Logging File only after Transaction finishes, so real updates about Database Data to Disk would take effect. *The Execution Sequence about Transaction when using Redo is:*

1. *Point in the Logging File that includes any Changed Elements.*
2. *COMMIT Logging Record.*
3. *Submit changes of Database Elements to the Disk.*

*Example:*

Consider the Same Transaction before, below gives a possible series Events.

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

*Difference:*

* At first, in Forth and Seventh Line, New Data has been updated to Logging File, but not the OLD Data.
* Second, we notice that the Logging Record <COMMIT T> appears early in the Logging File, it appears in the Eighth Line.
* Logging File has been Flushed and all other updates about Transaction T appears in the Logging File.
* Any modifications about Variable A and B appear in the Logging File Record.

Chapter 6.3.2 Recovery by using Redo Logging File

Chapter 6.3.3 Checkpoint for Redo Logging File

Chapter 6.3.4 Recovery by using Redo Logging File with Checkpoint