Chapter 8.1 Serializable and Recoverable

*Background:*

In Chapter Six, only discuss the Log creation and how to recover the Database System Status by using Logging when crash happened. Also, introduce one method of Database Calculation, which means the value moving among Non - Volatile Disk, Volatile Disk, and the Local Address Space in the Transaction. The Logging System does not support the Serializable; It just rebuild and commit Transaction according to the Disk Copy in the Database. Actually, the Commercial Database System does not always support Serializable, in some system, only when the user acclaims, then the Serializable can be realized.

In the Chapter Seven, it only discussed Serializable. According to Principle Design of Schedule, it may need to do some intolerant things for Logging Manager. The much more worse thing is that even crash has not happened, and in principle, the Schedule maintain Serializable. After the Transaction which writes the Database Element aborts, but the thing written before has not been undone, then this may easily cause the Database Element inconsistent.

Chapter 8.1.1 Dirty Data

*Definition:*

*When Data has been written into Buffer, Disk, or both by uncommitted Transaction, then it may cause problem.*

*Example:*

Let’s reconsider Serializable Schedule, but assume that after Transaction T1 read B, and because of some reason, it needs to abort. Such Incidents Sequence is just as the picture below.

|  |  |  |  |
| --- | --- | --- | --- |
| T1 | T2 | A | B |
|  |  | 25 | 25 |
| l1(A); r1(A); |  |  |  |
| A := A + 100; |  | 125 |  |
| w1(A); l1(B); u1(A); |  |  |  |
|  | l2(A); r2(A); |  |  |
|  | A := A \* 2; |  |  |
|  | w2(A); | 250 |  |
|  | l2(B) has been declined. |  |  |
| r1(B); |  |  |  |
| Abort(B); u1(B); |  |  |  |
|  | l2(B); u2(A); r2(B); |  |  |
|  | B := B \* 2 |  |  |
|  | w2(B); u2(B); |  | 50 |

After Transaction T1 aborts, then Schedule would release its Lock on B. This step is the key point, since other Transactions can never get Lock on B.

At this time, Transaction T2 has already read the inconsistent Database Status, which is to say, Transaction T2 reads the value of A which is the status after Transaction T1 has read, and the value B it reads has been committed by the Transaction committed before T1. Under this kind of situation, whether Transaction T1 would write A value of 250 into the disk;

No matter what may happen, the Transaction would read such value from Buffer Area. Since the inconsistency status that Transaction T2 has read, therefore it made Database System in the inconsistency status, among which A != B.

*The Key Problem is that, the Value A Transaction T1 has written is dirty data, no matter it stays in the main memory or stays in the hard disk. The Transaction T2 reads A, then during Calculation, it uses this value during calculation, which makes the Transaction T2 unreliable. So, we need to rollback Transaction T1 and T2.*

*Example:*

Now, consider table below, it gives the next action sequence from the schedule which is based on Time Stamp. But, we assume that this Schedule would not use commitment bit. *( Recall that the commitment bit is used to prevent the value which has not been committed would be read by other Transactions. )*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| T1 | T2 | T3 | A | B | C |
| 200 | 150 | 175 | RT = 0 | RT = 0 | RT = 0 |
|  |  |  | WT = 0 | WT = 0 | WT = 0 |
|  | w2(B); |  |  | WT = 150 |  |
| r1(B); |  |  |  |  |  |
|  | r2(A); |  | RT = 150 |  |  |
|  |  | r3(C); |  |  | RT = 175 |
|  | w2(C); |  |  | WT = 0 |  |
|  | Abort; |  |  |  |  |
|  |  | w3(C); | WT = 175 |  |  |

When Transaction T1 reads B in the second step, then there has no commitment byte to tell Transaction T1 that it needs to wait. So Transaction T1 can continue to execute, and it even can write into Disk and commit. In this example, there has no detailed steps.

At last, Transaction T2 tries to write C in some unrealizable method, therefore Transaction T2 aborts. The effect Transaction T2 writes into Disk has been undone. Therefore, B has been restored to the value before Transaction T2 reads and writes into the Disk.

However, we can see that B has been used in Transaction T1, for example, Transaction T2 can use this value to calculate A, B and/or C value and write back to the Disk. Therefore, after Transaction T1 has read B (Dirty Data), it can cause the Database into some inconsistent status.

*Attention that, if Commitment Bit has been used, then the second step r1(B) would be delayed, we need to wait Transaction T2 to abort and the value of B has returned to its original value, this action can be enabled to happen.*

Chapter 8.1.2 Cascade Rollback

*Definition:*

Just as we see the example above, if the Transaction can get Dirty Data, then we need to execute *Cascade Rollback*.

* *When the Transaction T has been aborted, we need to make sure how many Transactions have read Data that Transaction T has written, we need to abort these Transactions recursively.*
* *Log File are those Log Files that provide data that has been changed before modification, then we can use Log Files to retreat such influence from Transaction Abortion.*
* *If influence of Dirty Data has not reached Disk, then we can use the Copy of Hard Disk to recover the Data.*

Then just as we discussed before, then commitment bit in Schedule which is based on Time Stamp forbids the Transaction which may has read Dirty Data continue to execute, therefore, when calling the Schedule would not generate cascade rollback.

The Schedule which is based on Validation may also avoid Cascade Rollback, since write into Database or Buffer may only happen after the Transaction has been committed.

Chapter 8.1.3 Recoverable Schedule

Chapter 8.1.4 Schedule to Avoid Cascade Rollback

Chapter 8.1.5 Management on Rollback Based on Lock

Chapter 8.1.6 Commit based on Array

Chapter 8.1.7 Logical Logging

Chapter 8.1.8 Recover from Logical Logging