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:*

Chapter 8.1.2 Cascade Rollback

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