## Chapter 7.1 Serialized Scheduling and Serializable Schedule

*Rightness Principle:*

Back to the Rightness Principle, here if each Transaction is executed isolated without currency Transaction, then *the status of Database System would be turns from one right status to another right status.*

But practically, Transaction would be executed with other Transactions currently, therefore, Rightness Principle would not be suitable here. This Chapter introduces the concept of *Scheduler* and *Result of ‘Serializable Schedule’ would be the same as execution one Transaction one time.*

Chapter 7.1.1 Schedule

*Definition:*

Scheduler is one Sequence of one or multiple Transactions. We need to pay attention that when we do research on Concurrency Control, the important Read and Write happens in the Main Memory but not Disk. Which is to say that, variable A which has been put into the Main Memory by Transaction T can be visited by Transaction T but also can be visited by other Transactions.

*Example:*

Let’s consider two Transactions and their executions would bring what influence to the Database according to the sequence. The Transactions T1 and T2 are just like below, variable t and s are local variables for Transactions T1 and T2. They are not Database Elements.

|  |  |
| --- | --- |
| *T1* | *T2* |
| *READ(A, t);* | *READ(A, s);* |
| *t := t + 100;* | *s := s \* 2;* |
| *WRITE(A, t);* | *WRITE(A, s);* |
| *READ(B, t);* | *READ(B, s);* |
| *t := t + 100;* | *s := s \* 2;* |
| *WRITE(B, t);* | *WRITE(B, s);* |

We assume that the only constraint is that A = B. Since Transaction T1 add 100 to A and 100 to B, but T2 multiple 2 to A and B, we know that under isolation, their running situation can stay consistent.

Chapter 7.1.2 Serialized Schedule

*Definition:*

*One Schedule must be consist by all actions of one Transaction, and then all actions of other Transactions, so such Schedule is Serialized, no mixture.*

*Example:*

For Transactions in pic 7 - 2, there have two Serialized Schedule Sequence, one is that Transaction T1 is executed before Transaction T2, while the other is that Transaction T2 is executed before Transaction T1. The pic 7 - 3 gives the incident sequence that Transaction T1 is before Transaction T2, the initial status is A = B = 25. All variables A and B are in the Main Memory, but not values in the Disk.

|  |  |  |  |
| --- | --- | --- | --- |
| *T1* | *T2* | *A* | *B* |
|  |  | *25* | *25* |
| *READ(A, t);* |  |  |  |
| *t := t + 100;* |  |  |  |
| *WRITE(A, t);* |  | *125* |  |
| *READ(B, t);* |  |  |  |
| *t := t + 100;* |  |  |  |
| *WRITE(B, t);* |  |  | *125* |
|  | *READ(A, s);* |  |  |
|  | *s := s \* 2;* |  |  |
|  | *WRITE(A, s);* | *250* |  |
|  | *READ(B, s);* |  |  |
|  | *s := s \* 2;* |  |  |
|  | *WRITE(B, s);* |  | *250* |

For Transactions in 7 - 4, Transaction T2 is executed before T1, assume that the initialized status A = B = 25. Attention that, the Scheduler value of A and B are totally different.

|  |  |  |  |
| --- | --- | --- | --- |
| *T1* | *T2* | *A* | *B* |
|  |  | *25* | *25* |
|  | *READ(A, s);* |  |  |
|  | *s := s \* 2;* |  |  |
|  | *WRITE(A, s);* | *50* |  |
|  | *READ(B, s);* |  |  |
|  | *s := s \* 2;* |  |  |
|  | *WRITE(B, s);* |  | *50* |
| *READ(A, t);* |  |  |  |
| *t := t + 100;* |  |  |  |
| *WRITE(A, t);* |  | *150* |  |
| *READ(B, t);* |  |  |  |
| *t := t + 100;* |  |  |  |
| *WRITE(B, t);* |  |  | *150* |

A and B are done before Transaction T1 are all 150. *Normally, we can not expect that Database Status has no related with Sequence of Transaction.* We can list all Actions according to the Sequence to represent Serialized Schedule.

*In the first image, Scheduler represents (T1, T2), while the following Scheduler represents (T2, T3).*

Chapter 7.1.3 Serializable Schedule

*Definition:*

*The Rightness Principle of Transaction tells us, each Serializable Schedule can keep the Database System consistency.* There is another guarantee that ensures Consistent Schedule. If there exists Serializable Schedule S’, for any database initialized status, the effect of Schedule S and S’ are the same, and we can say that Schedule S is the Serializable Schedule.

*Analysis:*

*The Non - Serialized Serializable Schedule is as below.*

|  |  |  |  |
| --- | --- | --- | --- |
| *T1* | *T2* | *A* | *B* |
|  |  | *25* | *25* |
| *READ(A, t);* |  |  |  |
| *t := t + 100;* |  |  |  |
| *WRITE(A, t);* |  | *125* |  |
|  | *READ(A, s);* |  |  |
|  | *s := s \* 2;* |  |  |
|  | *WRITE(A, s);* | *250* |  |
| *READ(B, t);* |  |  |  |
| *t := t + 100;* |  |  |  |
| *WRITE(B, t);* |  |  | *125* |
|  | *READ(B, s);* |  |  |
|  | *s := s \* 2;* |  |  |
|  | *WRITE(B, s);* |  | *250* |

Tell from the table above, the result of Serializable Schedule is just the same as Serialized Schedule (T1, T2).

Since all consistent Database Status satisfy A = B = c, so the value of A and B should equals to 2 \* (c + 100), so starts from each consistent status, consistency can be ensured.

*Another way around, consider the Non - Serialized Serializable Schedule.*

|  |  |  |  |
| --- | --- | --- | --- |
| *T1* | *T2* | *A* | *B* |
|  |  | *25* | *25* |
| *READ(A, t);* |  |  |  |
| *t := t + 100;* |  |  |  |
| *WRITE(A, t);* |  | *125* |  |
|  | *READ(A, s);* |  |  |
|  | *s := s \* 2;* |  |  |
|  | *WRITE(A, s);* | *250* |  |
|  | *READ(B, s);* |  |  |
|  | *s := s \* 2;* |  |  |
|  | *WRITE(B, s);* | *50* |  |
| *READ(B, t);* |  |  |  |
| *t := t + 100;* |  |  |  |
| *WRITE(B, t);* |  |  | *150* |

We can ensure that it is not the Serialized Schedule because it starts from the consistent status A = B = 25, and at last, the Database Status turns to the Non - Consistent status A = 250 but B = 150. Actually, the Transaction T1 takes effect on A first, and Transaction T2 takes effect on B next. But, actually, they take totally different algebra algorithm, which means that A = 2 \* (A + 100), but B = 2 \* B + 100.

Chapter 7.1.4 Influence of Transaction Semantics

Chapter 7.1.5 Notation of Transaction and Scheduling