## 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

During learning Serializable Schedule, we consider the execution of Transaction clearly, to make sure whether is there any possibility to make sure that one Schedule is Serializable. We do know that Transaction Detail do has the Relation just as we see the example below.

*Example:*

Just as the example below, the only different in the example is the Operation that Transaction T2 operates, which is to say that Transaction T2 adds 200 on A and B each separately, but not multiple each by 2. Here, it is easily to verify that no matter what the initial status is, the final status is the result that generated by Serialized Schedule.

|  |  |  |  |
| --- | --- | --- | --- |
| T1 | T2 | A | B |
|  |  | 25 | 25 |
| READ(A, t) |  |  |  |
| t := t + 100; |  |  |  |
| WRITE(A, t) |  | 125 |  |
|  | READ(A, s) |  |  |
|  | s := s + 200; |  |  |
|  | WRITE(A, t) | 325 |  |
|  | READ(B, s) |  |  |
|  | s := s + 200; |  |  |
|  | WRITE(B, s) |  | 225 |
| READ(B, t) |  |  |  |
| t := t + 100; |  |  |  |
| WRITE(B, t) |  |  | 325 |

This is one Serializable Schedule that because of the Transaction Detail.

However, it can not be realized to make the Schedule to consider all calculation details. Since there would includes SQL, other code that programmed by Senior Language and include the code realized by other common Programming Language, which can not tell what the exact things the Transaction does. But, the Schedule does can see the Read and Write command from the Transactions, then it can know what kind of Database Elements have read and it can know which Elements it modified. In order to simplify the work of schedule, we assume that:

*Random Database Element A written by Transaction T depends on the Database Status without any Algebra Coincidence.*

*Example:*

The example above is a very coincident example, although two operations operates different operations on two variables, because A + 100 + 200 = B + 200 + 100, so A = B. To change the words, If there has something that Transaction T would do to Database System and make the Database System totally different, then Transaction T would do this.

Chapter 7.1.5 Notation of Transaction and Scheduling

Assume that there has no coincidence, then only when the transaction executes write and read, then we can take into consideration but not relates to the truth value. So, her we take one simpler method to represent the Transaction and Schedule, among which the actions has rT(X) and wT(X), which are used to represent the Transaction Write and Read Database Element X. On other words, since we can use T1, T2, ..., to represent Transaction, normally we use ri(X) and wi(X) which are the synonymy of r(Ti)(X) and w(Ti)(X), here i is the synonymy of Ti.

*Example:*

The Transaction can be written as the below:

*T1: r1(A); w1(A); r1(B); w1(B);*

*T2: r2(A); w2(A); r2(B); w2(B);*

Take another example as below:

*r1(A); w1(A); r2(A); w2(A); r1(B); w1(B); r2(B); w2(B);*

is just the Serializable Schedule in 7 - 5.

To be more precise:

1. *Action just as the representation ri(X) or wi(X), which are represent Transaction Ti read or write Database Element X.*
2. *Transaction Ti has the action sequence with the subscript of i.*
3. *The Schedule S of the Transaction Collection T is one of the Action Sequence, among which each Transaction Ti in the Transaction T, the sequence that actions (Read/Write) in Transaction Ti is just the same as the sequence that it appears in the Schedule S. Then we say that the Sequence S is one of the Intersection of its original Transaction Sequence.*

*Example:*

All actions with the sequence subscripts 1 appears in the Transaction T1 with the same sequence, but the all other actions with the sequence subscripts 2 appears just the same as in the Transaction T2.