## Chapter 7.3 Serializable Realization by Using Lock

In this Chapter, we consider the architecture that used most in the Schedule, and this architecture maintain ‘*Lock*’ on the Database Element to prevent Non - Serializable Behavior.

More directly, it means that *Transaction gets Lock on the Required Database Element, to prevent other Transactions accessing the Database Element at the same time, which can be used to prevent causing any possibilities of Non - Serializable situation.*

*Introduction:*

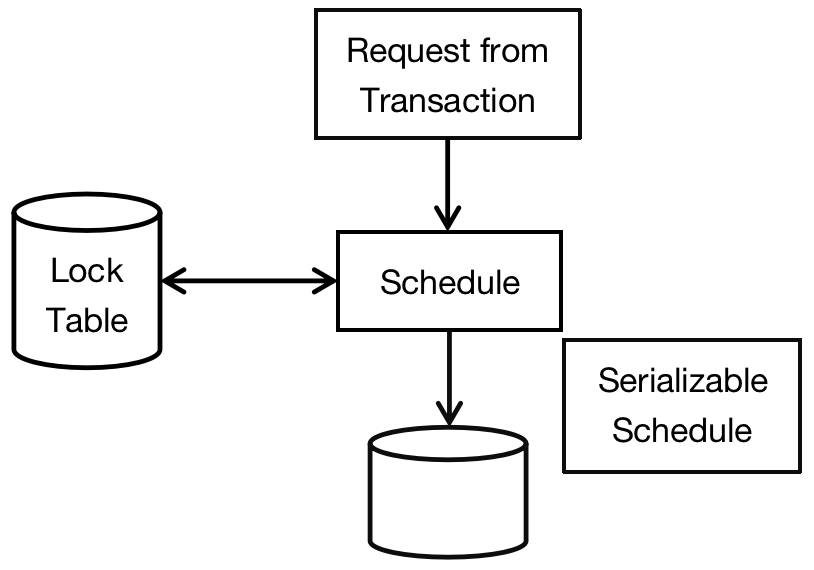
In this Chapter, we would introduce one simple Locking Module to introduce the concept of *Locking*. This module has only one Lock, it is a must when Transaction wants to do any Operations on these Database Elements.

In Chapter 7.4 would introduce much more *Realistic Locking Mode*, in which the Locking Mode can use multi - Lock, which includes *Sharing Lock/Exclusive Lock which correspond to Write Authority and Read Authority.*

Chapter 7.3.1 Lock

*Introduction:*

In the picture, the schedule would choose to use one Lock Table to help finished Transaction. *The responsibility of Schedule is to accept the request from Transaction, or enable them work on the database element, or push them back until it is safe to continue to work. The Lock Table is used to instruct this decision.*



In ideal situation, when Schedule starts to forward any request, only when this request would not make Database System under the inconsistent situation when the request is executed after all active Transactions have been committed or aborted. *The Lock Schedule is actually like the most of Scheduler, and it is Conflict Serializable.*

*Principle:*

When the Schedule is using the lock, beside read and write Database Element, it would request and release Lock. The usage of Lock must be meaningful in two below situations: The first one is Structure of Transaction, and the second one adapts to Schedule Structure.

*Consistency of Transaction - Behavior and Lock must be connected by using the anticipated method:*

1. *Transaction can read or write Database Elements, only when the Lock has been granted to the Database Element but has not been released.*
2. *If one Transaction has locked some Database Element, then it must release for this Element.*

*The Legality of Schedule - Lock must have it’s anticipated meaning:*

*Random two Transactions can not lock the same Database Element, unless one of the Transaction has released its Lock first.*

We would enlarge the Symbolize of Actions, to add Lock and Release Action:

*li(X): Transaction Ti requests Lock on Database Element X.*

*ui(X): Transaction Ti releases Lock on Database Element X.*

*Extension:*

Therefore, we can simplify two rules about the Consistency of Transaction and the Legality of Schedule:

*Consistency of Transaction:*

*As long as the Transaction Ti has the action ri(X) and wi(X), then before these two actions there must exist li(X), and among li(X), ri(X) and wi(X), there has no ui(X), and after the transaction finishes visiting X, there would be ui(X).*

*Legality of Schedule:*

*If there has lj(X) after the behavior li(X) in the Schedule, then there must has one behavior ui(X) somewhere between the behavior lj(X) and li(X).*

*Example:*

Let’s consider the two Transactions T1 and T2.

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

In the Transaction Sequence below, we add Lock behavior to it:

*T1: l1(A); r1(A); A := A + 100; w1(A); u1(A);* ***|*** *l1(B); r1(B); B := B +100; w1(B); u1(B);*

*T2: l2(A); r2(A); A := A \* 2; w2(A); u2(A);* ***|*** *l2(B); r2(B); B := B \* 2; w2(B); u2(B);*

These Transactions released all Locks they had before on parameter A and B. Also, only after these Transactions have request Locks on variables A and B, then they start to read and write on variables A and B. The table below gives the right sequence:

|  |  |  |  |
| --- | --- | --- | --- |
| Transaction T1 | Transaction T2 | A | B |
|  |  | 25 | 25 |
| l1(A); |  |  |  |
| r1(A); |  |  |  |
| A := A + 100 |  |  |  |
| w1(A); |  |  |  |
| u1(A); |  | 125 |  |
|  | l2(A); |  |  |
|  | r2(A); |  |  |
|  | A := A \* 2; |  |  |
|  | w2(A); |  |  |
|  | u2(A); | 250 |  |
|  | l2(B); |  |  |
|  | r2(B); |  |  |
|  | B := B \* 2; |  |  |
|  | w2(B); |  |  |
|  | u2(B); |  | 50 |
| l1(B); |  |  |  |
| r1(B); |  |  |  |
| B := B + 100; |  |  |  |
| w2(B); |  |  |  |
| u2(B); |  |  | 150 |

*Analysis:*

The table above gives one legal schedule on two Transactions. This schedule is legal since two Transaction T1 and T2 have never kept lock on the same parameter on A and B. To put it more precisely, Transaction T2 starts to execute l2(A) after Transaction T1 has released its lock on parameter A, which is to say u1(A). Just the same way, the Transaction T1 just waits till Transaction T2 release it lock on parameter B which is u2(B), then Transaction T1 goes to ask the lock on parameter B, which is l1(B).

Seen from the result, we can check that the final result A is not equal to B, then this Schedule is not Serializable, although it is legal. In the following chapter, would introduce ‘Two Phase Lock’ technology to make ensure Schedule Conflict Serializable.

Chapter 7.3.2 Lock Schedule

*Definition:*

*The task about Lock Schedule is that only when the request would generate the legal Schedule, then Schedule would agree to execution.*

*Process:*

If the schedule has not been approved, then the Transaction Request would be delayed, until the Schedule agrees the Transaction Request. For help to generate Decision, then there would be one *Lock Table for the Schedule, for each Database Element, if there has one Lock, then in the Lock Table would indicate Transaction of the current Lock.* If there has one Lock, then *Lock Table would has been seen as Relation Locks(element[Database Element], transaction), which consists of Transaction that satisfies the Lock Pair (X, T) on Database Element X.* Schedule only needs to visit and modify this Relation Table.

*Example:*

The Transaction T1 and T2 lock the parameter B before release A.

*T1: l1(A); r1(A); A := A + 100; w1(A); l1(B); u1(A); | r1(B); B := B + 100; w1(B); u1(B);*

*T2: l2(A); r2(A); A := A \* 2; w2(A); l2(B); u2(A); | r2(B); B := B \* 2; w2(B); u2(B);*

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

*Analysis:*

In table 7 - 13 above, when Transaction T2 requests the Lock on parameter B, then the schedule refuse the request, since the lock that Transaction T1 makes request is still on B. So the Transaction T2 was delayed, then Transaction T1 continues the behavior. At last, Transaction T1 executes the u1(B), which will release the lock on parameter B. Then at that time, Transaction T2 can get the lock on variable B. Attention that, since the Transaction T2 was forced to wait, so it was forced to multiple by 2 after the Transaction T1 add 100 on parameter B. This process ensures the consistency of Database Status.

Chapter 7.3.3 Two Phase Lock

*Definition:*

*One condition is called Two - Phase Locking (2PL), under this kind of situation, we can ensure that Legal Schedule on Consistent Transaction is Conflict Serializable.*

In each Transaction, all Locking request is prior to all Release Request. Here, 2PL means two phases in which the first phase makes request Lock and the second phase release Lock. The Two Phase Locking is just like the consistency, it adds the constraint condition on one Transaction Behavior Sequence. The Transactions that obeys the condition of Two Phase Locking is called Two Phase Locking Transaction, or 2PL Transaction.

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

*Transaction Sequence above is called Two - Phase Locking Transaction. Transaction T1 request Lock on parameters A and B in the first five Steps, and in the next five steps, it release the lock in the next five steps.* The same step in the Transaction T2.

After comparing two tables in 7.3.2 and 7.3.3, then we can know that the Two - Phase Lock Transaction can ensure the consistency when interacts with Schedule, but the Non - 2PL Transactions does not permit the inconsistent behavior.

Chapter 7.3.4 Reason Why Two Phase Lock Take Effect

*Explanation:*

Each Two - Phase Locking Transaction can be considered as that it was finished execution when first release request has been made. The Serialized Schedule of Conflict Equivalence of Schedule S in 2PL Transaction is the Serialized Schedule which has the same Release Sequence. Of course, we can use the *Conflict Equivalent Serialized Schedule to substitute the consistent Two - Phase Lock Transaction*. However, this process can be finished by using the conclusion to prove.

The Conflict Equivalent Question is only key to the behavior of Read and Write. Once we sort on the Read and Write, then we can add a series of Lock and Release behavior around them. Since each Transaction would release its Lock before it’s end, so we know it’s legal for the Serialized Schedule.

*Example:*

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

In the table above, we can exchange the execution sequence of transaction T1 to make it *Serialized Schedule.*

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

Prove - omit.

*Conclusion:*

*It is possible to move all actions of Transaction Ti forward till the start of the Schedule without any read and write conflicts.*