## 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:*

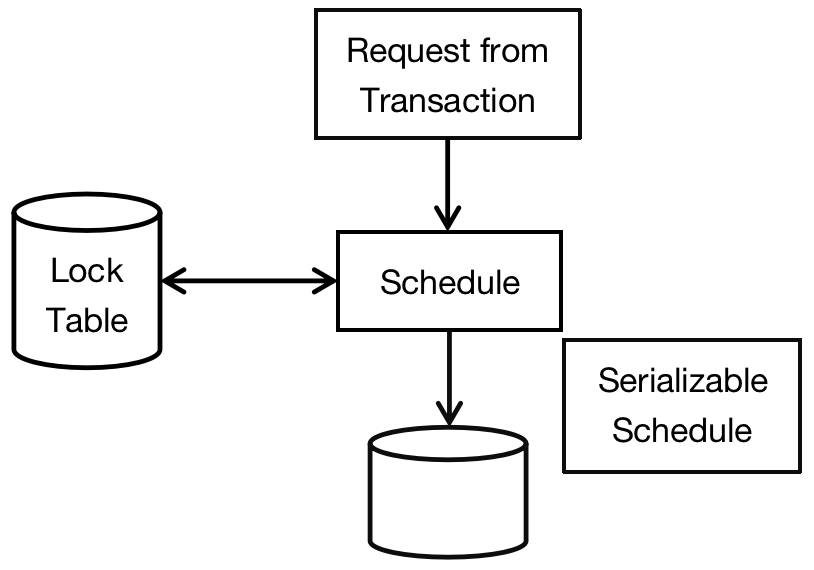
In this Chapter, we would introduce one simple Locking Module to introduce the concept of *Locking*. This module has only one Lock, it is a must when Transaction wants to do any Operations on these Database Elements.

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

Chapter 7.3.3 Two Phase Lock

Chapter 7.3.4 Reason Why Two Phase Lock Take Effect