Chapter 7.4 Locking System with Multi - Locking Mode

*Introduction:*

Main problem in Simple Lock System introduced before is that even Transaction T wants to read Database Element X but not read it, then it should request Lock on parameter X.

We can not escape getting the Lock. If not getting lock, then Non-Serializable Behavior would happen when another Transaction would write X when Transaction T is active.

*Outline:*

* *Two different complex Lock which is used to read (Shared Lock and Read Lock) and another is used to write (Exclusive Lock and Write Lock).*
* *Shared Lock can be upgrade into the Exclusive Lock.*
* *Increment Lock and can be used to write the Database Element incrementally. Increment Lock can be swapped.*
* *Compatibility Matrix is used to present what kind of Lock can be assigned on Database Element when there already has other locks already assigned on current Database Element.*

Chapter 7.4.1 Sharing Lock and Exclusive Lock

*Definition:*

When we write, Lock is much stronger than the Lock when we read, since the Database Element forbids reading and writing when we write. Let’s consider Lock Schedule which uses two different types Locks - Shared Lock and Exclusive Lock.

* *For any Database Element X, if there has one Exclusive Lock on X - which is used to Write on the Database Element X.*
* *For any Database Element X, there has no Exclusive Lock but there has random number Shared Lock on it - which is used to Read on Database Element X.*

*Representation:*

* *sli(X) is used to present ‘Transaction Ti applies one Shared Lock’ on Database Element X.*
* *xli(X) is used to present ‘Transaction Ti applies one Exclusive Lock’ on Database Element X.*
* *ui(X) is used to present ‘Transaction Ti releases the Lock’, which means Transaction release whatever lock it has on itself.*

Consistency of Transaction, 2PL of Transaction and Legality of Schedule have all been included in the respective Relation, here we can include all these as:

1. *Consistency of Transaction - Can not write only after has Exclusive Lock, and if has not got some locks, then can not read.*
2. *Before read behavior ri(X), there should have sli(X) or xli(X), and between them, there should has no ui(X).*
3. *Before write behavior wi(X), there should has xli(X), and between them, there should has no ui(X).*
4. *2PL of Transaction - Lock should before Release. More precisely, for random Database Element Y, uj(Y) should not appear right before any sli(X) or xli(X).*
5. *Legality of Transaction - One element can only be locked exclusively or can be shared locked, but can not have both. To be more precisely:*
6. *If there exists xli(X) in the Schedule, then for j != i, afterwords there should have not xlj(X) or slj(X), unless there has ui(X).*
7. *If there exists sli(X) in the Schedule, then for j != i, afterwords there should have not xli(X), unless there has uli(X).*

*( Attention, owning Shared Lock and Exclusive Lock is allowed on one Database Element, as long as the conflict would not happen with other Transactions. )*

*Example:*

Let’s consider to use Shared Lock and Exclusive Lock, below is the possible Schedule about two Schedules:

*T1: sl1(A); r1(A); xl1(B); r1(B); w1(B); u1(A); u1(B);*

*T2: sl2(A); r2(A); sl2(B); r2(B); u2(A); u2(B);*

Transaction T1 and T2 both read Database Element B, but only Transaction T1 writes B. Both do not write A. The Transaction Sequence is as table below.

|  |  |
| --- | --- |
| *Transaction T1* | *Transaction T2* |
| *sl1(A); r1(A);* |  |
|  | *sl2(A); r2(A);* |
|  | *sl2(B); r2(B);* |
| *xl1(B) is declined.* |  |
|  | *u2(A); u2(B);* |
| *xl1(B); r1(B); w1(B);* |  |
| *u1(A); u1(B);* |  |

*Explanation:*

Transaction T1 requests the Shared Lock on variable A, then Transaction T2 requests the Shared Lock on the variable A and B. Now Transaction T1 asks one Exclusive Lock on on variable B but this requirement would be declined, since Transaction T2 has already applied the Shared Lock in B. Transaction T1 needs to wait till the Shared Lock has been released. Then Transaction T1 can continue to proceed.

*The Schedule below is Conflict Serializable, the equivalent schedule is (T2, T1), since Transaction T2 releases Lock before Transaction T1.*

|  |  |
| --- | --- |
| *Transaction T1* | *Transaction T2* |
|  | *sl2(A); r2(A);* |
|  | *sl2(B); r2(B);* |
|  | *u2(A); u2(B);* |
| *sl1(A); r1(A);* |  |
| *xl1(B) is declined.* |  |
| *xl1(B); r1(B); w1(B);* |  |
| *u1(A); u1(B);* |  |

Chapter 7.4.2 Compatibility Matrix

*Definition:*

If we use several kinds of Lock Type, then Schedule may need one known Strategy about when to grant Lock for Transaction when already have other types of Locks. *Compatibility Matrix is one simple method which describes Lock - Management Strategy. Each Lock method has one line and one row. The row means that Lock which another Transaction already has been granted for Database Element X, while column means the application Transaction Type for Database Element X.* The rule can be described as:

* *The C type Lock is only granted when each types of Lock that granted on other Transactions on Database Element X has corresponding column R, the value of which on the C type row is ‘YES’.*

*Example:*

Table 7 - 16 is the Compatibility Matrix which includes Shared Lock and Exclusive Lock. About the Column of Shared Lock means that if there has only Read Lock on the Database Element, then we can grant Shared Lock on it. Only when there has no other Locks that has been granted, then we can grant Exclusive Lock on it.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Applied Lock | |
| Shared Lock | Exclusive Lock |
| Locks that already have been granted | Shared Lock | YES | NO |
| Exclusive Lock | NO | NO |

Chapter 7.4.3 Upgrade Lock

*Background:*

Transaction that has Shared Lock on Variable X is friend to other Transactions, since other Transactions also can apply Shared Lock on Variable X. *We need to make sure whether it is much more friendly to upgrade Shared Lock to Exclusive Lock - Upgrade Lock.*

*Example:*

Transaction T1 and T2 can execute and calculate currently, but it is impossible for Transaction T1 get the Exclusive Lock.

Two Transactions are:

*T1: sl1(A); r1(A); sl1(B); r1(B); xl1(B); w1(B); u1(A); u1(B);*

*T2: sl2(A); r2(A); sl2(B); r2(B); u2(A); u2(B);*

*Analysis:*

Transaction T1 reads A and B and execute some calculation on them, and at the end, it uses the result to write new value for B. Attention, here Transaction T1 gets the Shared Lock for B, and after that, it finishes all calculation of A and B, then it applies the Exclusive Lock for B. Transaction T2 just reads and writes on A and B, but do not writing.

Table below gave one possible schedule:

|  |  |
| --- | --- |
| *Transaction T1* | *Transaction T2* |
| *sl1(A); r1(A);* |  |
|  | *sl2(A); r2(A);* |
|  | *sl2(B); r2(B);* |
| *sl1(B); r1(B);* |  |
| *xl1(B) is declined;* |  |
|  | *u2(A);u2(B)* |
| *xl1(B); w1(B);* |  |
| *u1(A); u2(B);* |  |

*Analysis:*

At first, Transaction T2 applies Shared Lock on variable B before Transaction T1, but after that, Transaction T1 gets Shared Lock on variable B which is permitted. After that, when Transaction T1 tries to applies the Exclusive Lock on B, then it is declined. And Transaction Schedule forced Transaction T1 to wait till the Shared Lock is released by Transaction T2, then Transaction Schedule would permit Transaction T1 to get Exclusive Lock on B. Then this series of Transactions have been done.

*Example:*

*Unluckily, upgrade the Type of Lock without difference would introduce new and more serious situation ‘Dead Lock’.*

*Background:*

Assume that Transaction T1 and T2 would read Database Element A, and write for Database Element A. If these two transactions gets the Shared Lock first and upgrade to the Exclusive Lock after that. Then as long as Transaction T1 and T2 start at the same time, then the Transaction Sequence would happen as below.

|  |  |
| --- | --- |
| *Transaction T1* | *Transaction T2* |
| *sl1(A)* |  |
|  | *sl2(A)* |
| *xl1(A)* |  |
|  | *xl2(A)* |

*Analysis:*

Actually, Transaction T1 and T2 can get the Shared Lock on Variable A and then they tried to upgrade from Shared Lock to Exclusive Lock, but since there already has Shared Lock on Variable A, therefore Schedule forces Transaction T1 and T2 to wait. However, since both can not get any progress, they may wait forever till the system finds the existence of Dead Lock and abort one of the Transaction first, and grant the Exclusive Lock for another Transaction.

Chapter 7.4.4 Update Lock

*Definition:*

*Update Lock is the third Lock with which we can used to avoid Dead Lock. Update Lock uli(X) only gives the right to read but not the right to write, but only Update Lock can be updated to Write Lock and Read Lock can not be updated.*

When there has Shared Lock on Variable X, then we can granted Update Lock on Variable X, but only when Variable X has Update Lock, then no more any other kinds of type of Locks can be granted on Variable X. The reason is that if we can not decline such Lock, then the Update Lock could never be updated to the Exclusive Lock because of other types of Lock.

*When Transaction T1 applies Update Lock, it acts like Shared Lock while when Transaction T1 already has Update Lock, then it acts like Exclusive Lock.* Below is the Compatible Matrix with Shared Lock, Exclusive Lock and Update Lock.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Shared Lock | Exclusive Lock | Update Lock |
| Shared Lock | YES | NO | YES |
| Exclusive Lock | NO | NO | NO |
| Update Lock | NO | NO | NO |

*Example:*

Two Transactions are using the Update Lock would not have any influence. For third action, Transaction T1 would apply Update Lock on Variable B, but not Shared Lock. *Under this kind of situation, Upgrade Lock can be granted, since there has only Shared Lock on Variable B.*

|  |  |
| --- | --- |
| *Transaction T1* | *Transaction T2* |
| *sl1(A); r1(A);* |  |
|  | *sl2(A); r2(A);* |
|  | *sl2(B); r2(B);* |
| *~~sl1(B);~~ ul1(B); r1(B);* |  |
| *xl1(B) is declined;* |  |
|  | *u2(A);u2(B)* |
| *xl1(B); w1(B);* |  |
| *u1(A); u2(B);* |  |

*Example:*

The Upgrade Lock helps solve the Dead Lock Question above. Now, Transaction T1 and Transaction T2 all first apply the Upgrade Lock for Variable A but only that they get the Exclusive Lock. Transaction T1 and T2 can be described as:

*T1: ul1(A); r1(A); xl1(A); w1(A); u1(A);*

*T2: ul2(A); r2(A); xl2(A); w2(A); u2(A);*

One Transaction Sequence can be described as below:

|  |  |
| --- | --- |
| *Transaction T1* | *Transaction T2* |
| *ul1(A); r1(A);* |  |
|  | *ul2(A) is declined;* |
| *xl1(A); w1(A); u1(A);* |  |
|  | *ul2(A); r2(A);* |
|  | *xl2(A); w2(A); u2(A);* |

Transaction T2 that applies Update Lock for Variable A is declined. Transaction T1 is granted to be executed, after it is finished, then Transaction T2 can be executed.

Then, the Locking System would block Concurrency Execution of Transaction T1 and T2 but actually in this Sequence, random number Concurrency Executions would cause Dead Lock or inconsistent Database Status.

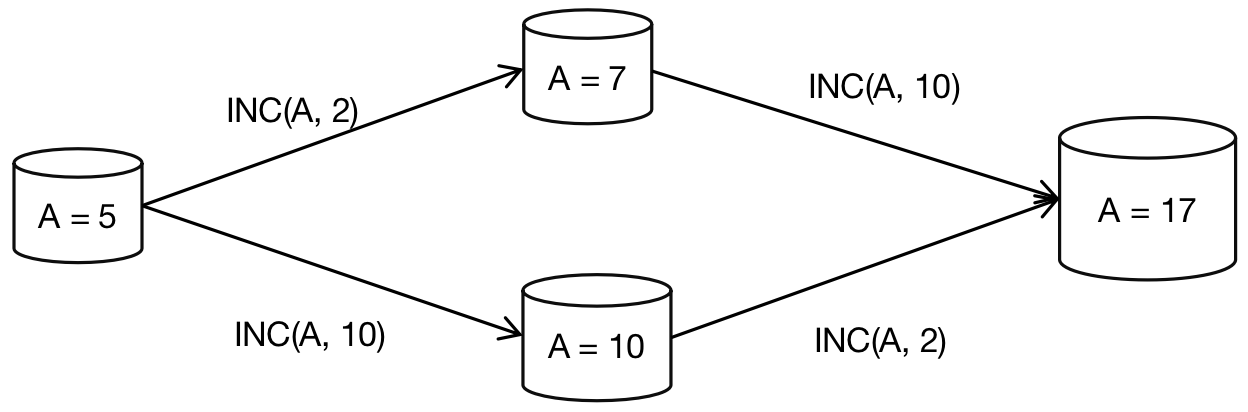
Chapter 7.4.5 Increment Lock

*Introduction:*

Another interesting Type of Lock is Increment Lock. Many Transactions just increment or decrease the storage value to operate on the Database System. For example, one Transaction that just transfer money from one Bank Account to another Bank Account.

*Property of Increment Lock:*

* *Increment Operations can be exchanged mutually, if two Transactions just add constant value for one common Database Element, then there has no relation about Operation Sequence, just shown as the Transition Diagram.*

**

* *Another side is that, Increment and Write or Read can not be exchanged. If you want to read it before increment A, then the value you get is different, but if you increment A before or after writing the new value in other Transactions, then you will get the totally different value A.*

Here, we would introduce one possible action, which is INC(A, c) and this action would add constant c to the Database Element A, also we assume that Database Element A is the single number. Attention that, the number c could be minus, and under this situation, we actually decrease Database Element A.

*Representation:*

* *Practically, we normally apply INC on one of Element of tuple, and the whole tuple is the lockable field but not one field of it.*
* *To be more formal, we use INC(A, c) to represent the following Atomic Operation: READ(A, t); t := t + c; WRITE(A, t);*

For Transaction Ti add some constant value on Database Element X, we can easily record it as inci(X), here it has nothing to do with constant c.

*Increment Lock:*

Corresponding to Increment Action, we need a Increment Lock. We use ili(X) to present that Transaction Ti applies Increment Lock on variable X.

*The existence of Increment Active and Increment Lock do need to do some modifications about the consistency of Transaction, conflict and legal Schedule. These modifications includes:*

1. *Consistency Transaction can do increment operation on Database Element X only when it has Increment Lock on Database Element X.*
2. *In the Legal Schedule, at any time, it can have random Transactions to get Increment Lock on Database Element X. But, if some Transaction has the Increment Lock on Database Element X, then other Transaction can not get Shared Lock and Exclusive Lock on Database Element X.*
3. *For j != i, inci(X) action may cause conflict with rj(X) and wj(X), but it definitely not has conflict with incj(X).*

*Example:*

Consider two Transactions, each one reads Database Element A and then add B on A.

*T1: sl1(A); r1(A); il1(B); inc1(B); u1(A); u1(B);*

*T2: sl2(A); r2(A); il2(B); inc2(B); u2(A); u2(B);*

The transaction Sequence above is consistent, since only when they owe Increment Lock, then they can do Increment Operation, and also when they has the Shared Lock, then they execute Read Operation. The table below gives one possible transaction sequence for T1 and T2.

|  |  |
| --- | --- |
| *Transaction T1* | *Transaction T2* |
| *sl1(A); r1(A);* |  |
|  | *sl2(A); r2(A);* |
|  | *il2(B); inc2(B);* |
| *il1(B); inc1(B);* |  |
|  | *u2(A); u2(B)* |
| *u1(A); u1(B);* |  |

*Analysis:*

At first, Transaction T1 reads Parameter A first, then Transaction T2 reads A and increments B on it. But, at this time, Transaction B is allowed to get the Increment Lock on B and increment B on Database Element A.

Attention, in the Transaction Sequence above, we do not need delay any Transaction. Addition Operation does no relation with the Result of Transaction. Put it in another word, we can see the Transaction Sequence as the Serialized Transaction as without Lock Operation:

*S: r1(A); r2(A); inc2(B); inc1(B); ->S’: r1(A); inc1(B); r2(A); inc2(B); -> S’’: r2(A); inc2(B); r1(A); inc1(A);*

We can exchange the last action inc1(B) to the second location, since there has no conflict with another Increment Operation and the Read Operation. Also in Transaction Sequence above, we can exchange sequence and make Transaction T2 just before Transaction T1.