Chapter 7.9 Concurrency Control by Validation

*Definition:*

* *Validation Method is another Optimistic Concurrency Control*, among which, we enable Transactions do not need to lock Database Element, but when in the appropriate time, we need to check *whether Transactions can be ran in Serializable Way.*

*Difference:*

* *For Validation, Schedule needs to maintain all records which are used to check what all Active Transactions are doing.*
* *For Time Stamp, Schedule needs to save Write Time and Read Time of all Database Elements.*

*Process:*

When Transaction is trying to write value for Database Element, it needs to go through ‘*Validation Period*’, at this time, it *compares all Read and Written Database Element Collection with Write Collection of other Active Transactions. If there do exist Non - Realizable Behavior Risk, then the Transaction should be roll back.*

Chapter 7.9.1 Confirm Structure based on Validation

*Definition:*

When Validation is used on Concurrency Control Mechanism, then for each Transaction, *Schedule needs to be informed Database Element Collection which needs to be written and read in Transaction*, and they are *Read Collection RS(T)* and *Write Collection WS(T)*. The Transaction can be divided into three phases to execute:

1. *Read - In the first phase, Transaction reads all Database Elements Collection which are waiting to be read. Transaction needs to calculate all values which is waiting to be written to its Local Address Space.*
2. *Validation - In the second phase, Schedule needs to compare the Collection of all Database Elements which are waiting to be read and written with all other Transactions to confirm the effectiveness of the current Transaction.* This process would be described later. *If the validation process failed, then the Transaction needs to be Rolled back; Otherwise, the Transaction goes to the Third Step.*
3. *Write - In third phase, Transaction needs to write all Database Elements from Collection which are waiting to be wrote into the Database System.*

To be directly, each Transaction validates in just one moment. Therefore, the Schedule which is based on Validation would have an assumed Serializable Sequence for the Transaction, and it would go to check whether this Transaction Behavior is effective or not to decide whether this Transaction is effective or not.

*In order to support the decision about whether the Transaction is effective or not, the Schedule needs to maintain three collections:*

1. *START* - *The Transaction Collection which are already start but has not finished Validation.* For each Transaction in the Transaction Collection, Schedule needs to maintain *START(T)*, which is to say, *the start time of Transaction*.
2. *VAL (Validation)* - *The Transaction Collection which has already confirmed the effectiveness but has not stepped into third step.* For each Transaction T in the Collection, the Schedule needs to maintain *START(T) and VAL(T)*, which is to say, Validation Time of Transaction T. Attention that, *VAL(T) is the Execution Time of Transaction T* which has already assumed in the Serializable Sequence.
3. *FIN (Finish)* - *The Transaction has already finished the third step.* *For all Transactions, Schedule needs to record START(T), VAL(T) and FIN(T), which stands for the finishing time of Transaction T.* In principle, the Collection would increase, but just as we see, for random active Transaction U ( Which means that any Transaction U in the START or VAL. ), the Transaction T may satisfy the condition FIN(T) < START(T), then we do not need to remember the Transaction T. The Schedule can delete FIN Collection periodically, in case this may enlarge and exceed it’s size.

Chapter 7.9.2 Validation Rule

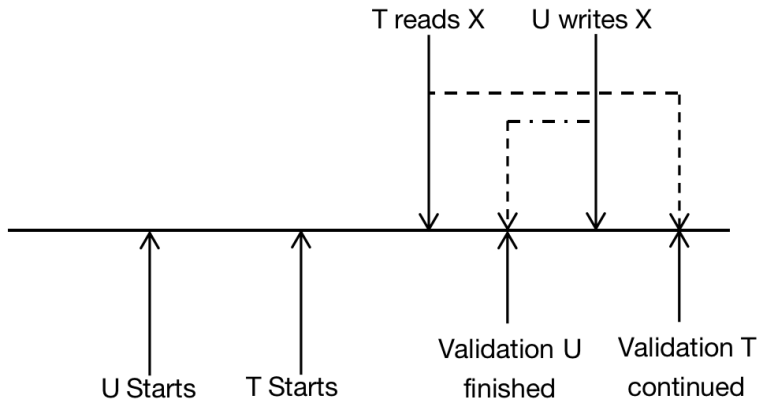
*Principle:*

The Schedule maintains all information above, these information enables it monitor all possible probabilities which may violate Serializable Sequence, assume that Transaction Serializable Sequence is the Validation Sequence of all Transactions. In order to understand these rules, let us consider what errors may happen when we want to make sure the validation of one Transaction.

*Assume that there exist Transaction U satisfy:*

1. *The Transaction U exists in VAL or FIN collection; which means the Transaction U has been validated.*
2. *FIN(U) > START(T); which means Transaction U has not finished before Transaction T started.*
3. *RS(T) Intersect with WS(U) equals to Non - Empty; Especially, this includes Database Element X.*

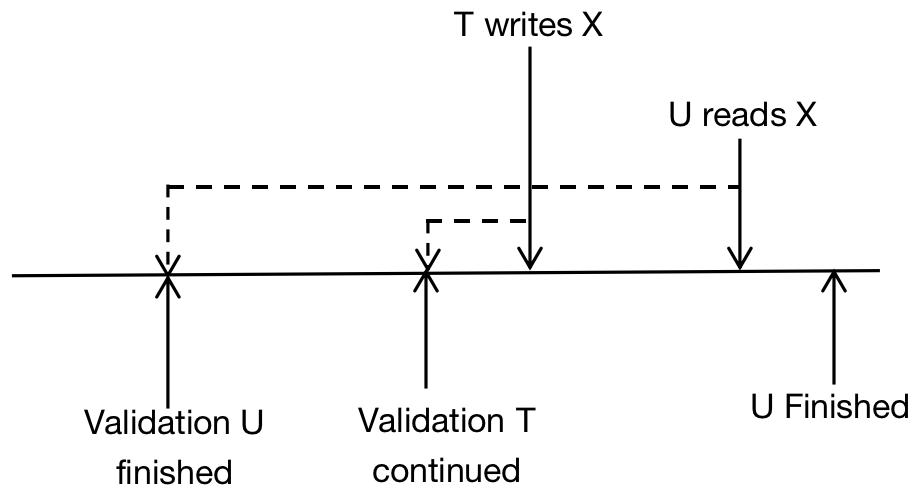
*There has possibility that Transaction U may write X after Transaction T reads X. Actually, Transaction U may has not wrote X.* Since we can not make sure whether there is any possibility that Transaction T has read X that wrote by Transaction U, therefore we must roll back Transaction T in case to escape the risk of Transaction Sequence of T and U may be different from Serializable Sequence.



*Assume that there exist Transaction U satisfy:*

1. *Transaction U stays in VAL collection; which is to say that the validation of Transaction U has been confirmed successfully.*
2. *FIN(U) > VAL(T); which is to say that Transaction U has not been finished before Transaction T enters the Validation Process.*
3. *WS(T) Intersect with WS(U) does not equal to empty; Especially, Database Element X exists in both Write Collection.*

The Transaction T and U all need to write Database Element X into the Transaction T and U and if we make sure the effectiveness of Transaction T, then it may writes Database X before Transaction U. *Since we can not make sure about this, then we can try to roll back Transaction T to ensure that the Serializable Sequence would not be violated.*

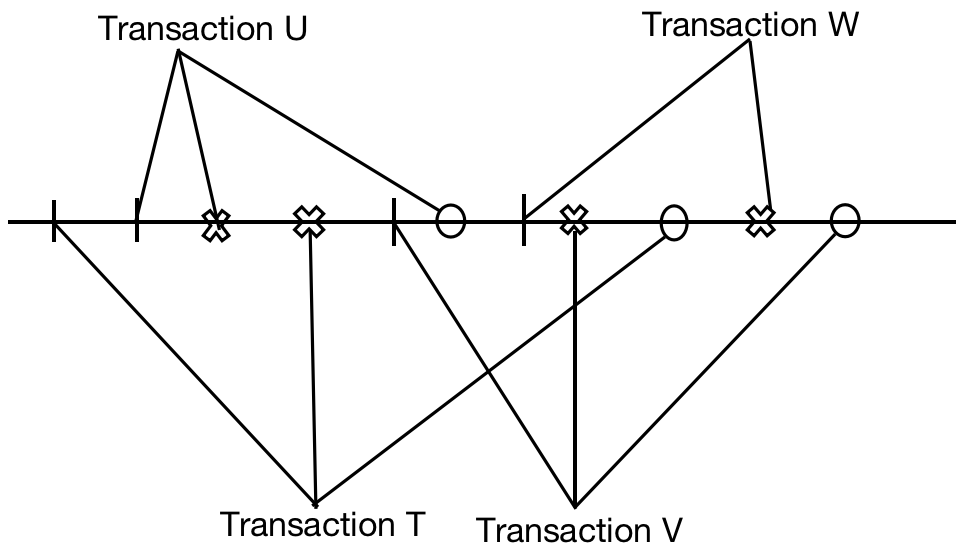


*The rule is about Validation of Transaction can be generalized:*

* *For all U that has been validated but not finished before start of Transaction T, which is to say, for Transaction U that satisfies FIN(U) > START(T), we need to check RS(T) intersect with WS(U) = Empty Collection.*
* *For all U that has been validated but not finished before Validation of Transaction T, which is to say, for Transaction U that satisfies FIN(U) > VAL(T), we need to check WS(T) intersect with WS(U) = Empty Collection.*

*Example:*

* Transaction U, RS(U) = { B } and WS(U) = { D }.
* Transaction W, RS(W) = { A, D } and WS(W) = { A, C }.
* Transaction T, RS(T) = { A, B } and WS(T) = { A, C }.
* Transaction V, RS(V) = { B } and WS(V) = { D, E }.



* *sta stands for start time of Transaction.*
* *val stands for validation time of Transaction.*
* *fin stands for finish time of Transaction.*

*Remember that, for the matched Transaction, we only focus on WS(Transaction T).*

*Analysis:*

1. *When validating Transaction U, we need to find Transaction Z which satisfies:* 
   1. *Transaction Z has finished validation, and it has not finished before start of Transaction U. Also, RS(U) intersect with WS(Z) = Empty Collection.*
2. *Transaction Z appears in VAL collection. -> Transaction Z has finished validation.*
3. *Zfin > Usta. -> Transaction Z has not finished before start of Transaction U.*
4. *RS(U) intersect with WS(Z) = Empty Collection.*
   1. *Transaction Z has finished validation, and it has not finished before validation of Transaction U. Also, WS(U) intersect with WS(Z) = Empty Collection.*
      1. *Transaction Z appears in VAL collection. -> Transaction Z has finished validation.*
      2. *Zfin > Uval. -> Transaction Z has not finished before validation of Transaction U.*
      3. *WS(U) intersect with WS(Z) = Empty Collection.*

*Conclusion - If we want to check validating Transaction T, then we need to find Transaction Z which satisfies the equality condition below:*

*START = { Usta, Tsta };*

*VAL = { Uval };*

*FIN = { Empty };*

*We can not find any other Transaction Node in VAL Collection, so we ignore validation of Transaction U. We already proved the validation of Transaction T.*

1. *When validating Transaction T, we need to find Transaction Z in the collection VAL:*

*START = { Usta, Tsta };*

*VAL = { Uval, Tval };*

*FIN = { Empty };*

*In this Example, we discuss the conditions of Transaction U to check whether it matches two below conditions.*

*Condition 1:*

* Check VAL collection, and there only has Transaction U. - *Satisfy*.
* Ufin > Tsta - *Satisfy*.
* WS(U) intersect with RS(T) = {D} intersect with {A, B} = Empty - *Satisfy*.

*Condition 2:*

* Check VAL collection, and there only has Transaction U. - *Satisfy*.
* Ufin > Tval - *Satisfy*.
* WS(U) intersect with WS(T) = {D} intersect with {A, C} = Empty - *Satisfy*.

*We can not find the Transaction Z that violates all conditions of 1 and 2. Therefore, we proved the validation of Transaction T successfully.*

1. *When validating Transaction W, we need to find Transaction Z in the Collection VAL:*

START = { Usta, Tsta, Vsta, Wsta };

VAL = { Uval, Tval, Vval };

FIN = { Ufin, Tfin };

*In this Example, we discuss the conditions of Transaction U, T, and V, and to check whether they match two below conditions.*

Transaction W - RS(W) = { A, D } and WS(W) = { A, C }.

Transaction U - RS(U) = { B } and WS(U) = { D }.

*Condition 1:*

* Check VAL collection, and Transaction U exists. - Satisfy.
* Ufin < Wsta - Not Satisfy.
* RS(W) intersect with WS(U) = { A, D } intersect with { D } = { D }. - Not Satisfy.

*Transaction U does not satisfy the second condition. Skip checking this part.*

*Condition 2:*

* Check VAL collection, and Transaction U exists. - Satisfy.
* Ufin < Wval - Not Satisfy.
* WS(W) intersect with WS(U) = { A, C } intersect with { D } = Empty Collection. - Satisfy.

*Transaction U does not satisfy the second condition. Skip checking this part.*

*To Conclude, Transaction U satisfies the Serializable Sequence.*

Transaction W - RS(W) = { A, D } and WS(W) = { A, C }.

Transaction T - RS(T) = {A, B} and WS(T) = {A, C}.

*Condition 1:*

* Check VAL collection, and Transaction T exists. - Satisfy.
* Tfin > Wsta - Satisfy.
* RS(W) intersect with WS(T) = { A, D } intersect with { A, C } = { A }. - Not Satisfy.

*There has Common Database Element in the RS(W) and WS(T), therefore the Serializable Sequence can not be ensured. We need to roll back Transaction W.*

*Condition 2:*

* Check VAL collection, and Transaction T exists. - Satisfy.
* Tfin < Wval - Not Satisfy.
* WS(T) intersect with WS(U) = { A, C } intersect with { A, C } = { A, C }. - Not Satisfy.

*Transaction T does not satisfy the second condition. Skip checking this part.*

*To Conclude, Transaction T skipped.*

Transaction W - RS(W) = { A, D } and WS(W) = { A, C }.

Transaction V - RS(V) = { B } and WS(V) = { D, E }.

*Condition 1:*

* Check VAL condition, and Transaction V exists. - Satisfy.
* Vfin > Wsta. - Satisfy.
* RS(W) intersect with WS(V) = { A, D } and { D, E } = Empty. - Satisfy.

*Transaction T satisfies the first condition. For Transaction W and V satisfy Serializable Sequence.*

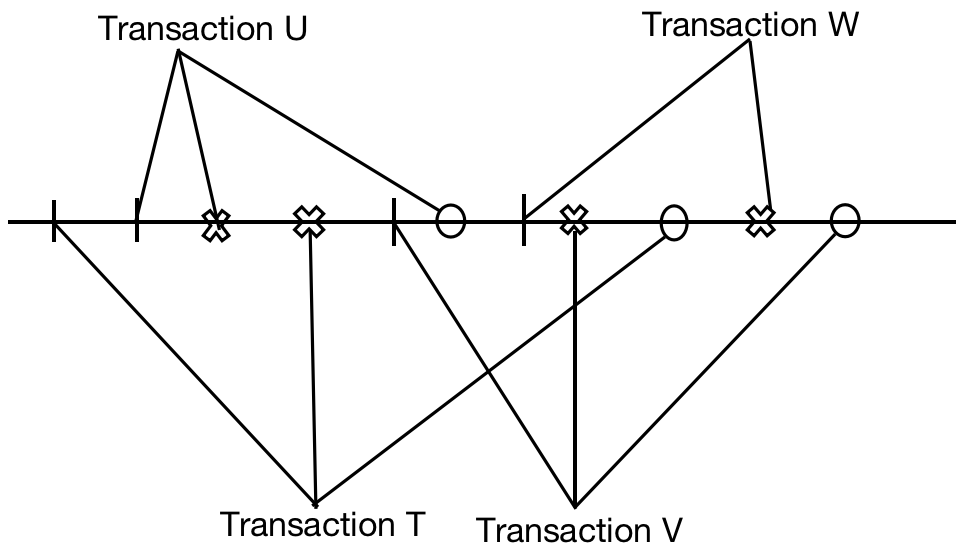
*Condition 2:*

* Check VAL collection, and Transaction V exists. - Satisfy.
* Vfin > Wval. - Satisfy.
* WS(V) intersect with WS(W) = { A, C } intersect with { D, E } = Empty. - Satisfy.

*Transaction T satisfies the second condition. For Transaction W and V satisfy Serializable Sequence.*

*To Conclude, We roll back Transaction W.*

1. *When validating Transaction V, we need to find Transaction Z in the Collection VAL:*



START = { Tsta, Usta, Wsta, Vsta };

VAL = { Uval, Tval };

FIN = { Ufin };

In this Example, we need to match Transaction U with Transaction V and Transaction T with Transaction V.

Transaction U - RS(U) = { B } and WS(U) = { D }.

Transaction V - RS(V) = { B } and WS(V) = { D, E }.

*Condition 1:*

* Check VAL collection, and Transaction U exists. - Satisfy.
* Ufin > Vsta - Satisfy.
* RS(V) intersect with WS(U) = { B } intersect with { D } = Empty Collection.

*Condition 2:*

* Check VAL collection, and Transaction U exists. - Satisfy.
* Ufin < Vval - Not Satisfy.

*To Conclude, Transaction U and V satisfy Serializable Sequence.*

Transaction T - RS(T) = { A, B } and WS(T) = { A, C }.

Transaction V - RS(V) = { B } and WS(V) = { D, E }.

*Condition 1:*

* Check VAL collection, and Transaction T exists. - Satisfy.
* Tfin > Vsta - Satisfy.
* RS(T) intersect with WS(V) = { A, B } intersect with { D, E } = Empty Collection.

*Condition 2:*

* Check VAL collection, and Transaction T exists. - Satisfy.
* Tfin > Vval - Satisfy.
* WS(T) intersect with WS(V) = { A, C } intersect with { D, E } = Empty Collection.

*To Conclude, Transaction V and T satisfy Serializable Sequence.*

*To Sum up, Transaction V satisfy the Serializable Sequence.*

*Supplement：*

The action validation is the Transient Behavior which happens in one moment or un - divided moment.

* If we run Validation on one *Single Processor System*, and it has only one Schedule Processor, we can confirm that Schedule Action happens in just one moment. The reason is that, if Schedule confirms Transaction T, then it would be impossible to make sure another Transaction U, so during Confirmation Process of Transaction T, then the validation status of Transaction U would not be changed.
* If we run Validation on *Several Multi - Processor System*, and there have multi - schedule Processor, then there has possibility that one of them are validating the Transaction T, and the other one is validating Transaction U. If so, we need to depend on some kind of Synchronous Mechanism to validate the Atomic Behavior.

Chapter 7.9.3 Comparative among three kinds of Concurrency Control

In this Chapter, we would take all three methods into consideration ( Locking, Time Stamp and Validation ), and compare all their advantages.

1. *Compare the utility of storage:*

* *Locking Mechanism: Locking Space is in proportion to the number of Locked Database Elements.*
* *Time Stamp: In the less maturity reality, the read and write time of each Database Element need space, no matter the Database Element would be visited.* But, the more preciser realization would see all Time Stamp of all earliest Transactions as ‘Negative Infinity’ and do not record them. Under this kind of situation, we can record Write Time and Read Time into one Table just as the Lock Table, of course, in which we only record all Database Elements that has been visited recently.
* *Validation: The space is used for record Time Stamp, Read, and Write Collections of each active Transactions and several few finished Transactions which starts after the current Transaction.*

Therefore, the space size used by each method is normally proportion to the total number of Database Elements of all Active Transactions. The Space for Time Stamp and Validation may be a bit more, since they record some visits from the Transactions Committed recently, while this has not been recorded by Lock Table. One potential problem of Validation is that, Writing Collection should be gotten before the Transaction has happened.

1. *Compare their completion ability if the Transaction would not be delayed.*

The performance of these three methods depends on whether Interaction of Transactions ( *The Interactions of Transactions means that one Transaction may visit the Database Element which would be visited by its Concurrency Transaction. )* is high or low:

* *Lock pushed back the Transaction but it can be used to avoid rollback, even the interaction is super high.* The Time Stamp and Validation have never pushed back transactions, but it would cause the rollback, but this is one more serious format of delay, and it would waste the resource.
* *If Interactions of each other Transaction is less, Time Stamp and Validation would not cause too much rollback*, and because they cost less than lock schedule therefore they are much more popular.
* *When rollback is necessary, Time Stamp can get problem much earlier than Validation, or by using Time Stamp, the Transaction can be finished its inner work before roll back.*