## Chapter 7.2 Serializable Conflicts

In Business Model, *Scheduler usually executes the condition which is called ‘Serializable Conflict’, it based on concept which is called ‘Conflict’, for series of actions, they satisfies: (definition)If their sequence changed, then at least one behavior among all involved Transactions would be changed.*

Chapter 7.2.1 Conflicts

*At first, we should know that most of behaviors are not conflict.* Then in the following content, we assume that Ti and Tj would be totally different Transactions, which means i doesn’t equal to j. *Under the conditions below, the conflict would not happen:*

1. *ri(X); rj(Y); would not conflict, even X = Y. The reason is that these steps would not change any Database Elements.*
2. *ri(X); wj(Y); would not conflict, as long as X != Y. The reason is that X and Y are totally different variables and when read the variable X first would not influence the process that Transaction j write into the variable Y. Here, Transaction i and Transaction j are totally different Transactions.*
3. *wi(X); rj(Y); would not conflict, as long as X != Y. The reason is the same as the second one.*
4. *wi(X); rj(Y); would not conflict, as long as X != Y. The reason is the same as the one before.*

Conversely, there also have three situations, *under which we can not exchange the behavior sequence, which means that exchange the sequence of behavior would have no problem:*

1. *Two behaviors in one Transaction, just as ri(X); wi(Y);* Their sequence in the Transaction has been fixed, therefore exchange their sequence is not permitted. They can not be sorted again.
2. *Different Transaction write on the same Database Element.* This sequence in the Transaction could not be changed also. Which is to say, wi(X); wj(X); can not be exchanged, and they are always be the conflict. If we exchange the sequence of wj(X) and wi(X); then at last, we would use the value of X calculated by wj(X).
3. *Different Transaction read and write operation on the same Database Element.* Which is to say the sequence of ri(X) and wj(X) and so is wi(X) and rj(X). If we move wj(X) before ri(X), then the value the reading behavior reads is the value after X has been written by Transaction Tj. Otherwise, the value the writing value of X is the value that Transaction Ti reads. Therefore, if we exchange the execution sequence of reading and writing, then it would have influence on the values of X that Ti reads.

*Conclusion - Random behavior of different Transactions can be exchanged, except below:*

1. *They are mainly on the same Database Element.*
2. *At least one behavior is write.*

So if we enlarge this thought, then we can accept random schedule, and to proceed the random Non-Conflict Exchange, the aim is to convert the Schedule to the Serialized Schedule, if we can do this, then the initial schedule is Serializable, since it would not change when encountering each Non - Conflict Exchange.

*Key Points:*

* *If one Schedule can be used a series of neighbor behaviors to convert to another, then we can say that two Schedules are Conflicts Equivalence.*
* *If one Schedule Conflict equals to one Serialized Schedule, then this Schedule is called Conflict Serializable.*

*(If one Schedule Conflict equals to one Serialized Schedule, then we need to say that this Schedule Conflict is Conflict Serializable.)*

*Example:*

Take the Schedule below:

r1(A);w1(A); r2(A);w2(A); r1(B);w1(B); r2(B);w2(B) as an example:

|  |  |
| --- | --- |
| Transaction T1 | Transaction T2 |
| r1(A) |  |
| w1(A) |  |
|  | r2(A) |
|  | w2(A) |
| r1(B) |  |
| w1(B) |  |
|  | r2(B) |
|  | w2(B) |

We can say that this Schedule is Conflict Serializable. We give one series of exchange and to make it final Serialized Schedule (It means Transaction T1 are all before Transaction T2.) Also, we add the underline to each steps that needs to be exchanged.

*Step 1:*

|  |  |
| --- | --- |
| Transaction T1 | Transaction T2 |
| r1(A) |  |
| w1(A) |  |
|  | r2(A) |
| r1(B) |  |
|  | w2(A) |
| w1(B) |  |
|  | r2(B) |
|  | w2(B) |

*Step 2:*

|  |  |
| --- | --- |
| Transaction T1 | Transaction T2 |
| r1(A) |  |
| w1(A) |  |
| r1(B) |  |
|  | r2(A) |
|  | w2(A) |
| w1(B) |  |
|  | r2(B) |
|  | w2(B) |

*Step 3:*

|  |  |
| --- | --- |
| Transaction T1 | Transaction T2 |
| r1(A) |  |
| w1(A) |  |
| r1(B) |  |
|  | r2(A) |
| w1(B) |  |
|  | w2(A) |
|  | r2(B) |
|  | w2(B) |

*Step 4:*

|  |  |
| --- | --- |
| Transaction T1 | Transaction T2 |
| r1(A) |  |
| w1(A) |  |
| r1(B) |  |
| w1(B) |  |
|  | r2(A) |
|  | w2(A) |
|  | r2(B) |
|  | w2(B) |

So, through a series of behavior exchange, then we got the Serialized Schedule.

Chapter 7.2.2 Estimation on Priority Picture and Serializable Conflict

It’s fairly easy to check whether Schedule S is the Serializable Conflict. Since no matter where the conflict happens, then as long as the sequence is totally the same as the sequence in the Schedule S, under this situation, the sequence can be thought as the Serializable Conflict. So the conflict behaviors add some conflicts on the behavior sequence. If these conflicts do not contradict mutually, then we can find one Serialized Schedule. Otherwise, there do not exist such Serialized Schedule.

For known Schedule S, which relates with Transaction T1 and T2, and maybe some other transactions, we can say that *Transaction T1 is pretty much prioritized than T2, which is written as T1 < s T2. If behavior A1 and A2 in Transaction T1, that satisfied:*

1. *In Serializable Sequence S, behavior A1 is before behavior A2.*
2. *Behavior A1 and A2 involve the same Database Element.*
3. *At least there has one Write behavior in A1 and A2.*

Attention that, this is just the situation that we can not exchange the Sequence of A1 and A2. Since random Schedule Conflict that equals to Schedule S, there has the situation that A1 appears before A2. So the Serialized Schedule of Schedule Conflict must make Transaction T1 right before T2.

*Priority Picture:*

Generalize the Sequence in the Priority Picture. The node in the Priority Picture is the Transaction in Schedule S. When these Transactions are different Transactions with different i, then we will use the integrity i to signal the node of Ti. If Ti < s Tj, then it would exist one arch that starts from node i to node j.

*Example:*

Below is the Schedule S and three Transactions T1, T2, T3:

*S: r2(A);r1(B);w2(A);r3(A);w1(B);w3(A);r2(B);w2(B);*

|  |  |  |
| --- | --- | --- |
| Transaction T1 | Transaction T2 | Transaction T3 |
|  | r2(A) |  |
| r1(B) |  |  |
|  | w2(A) |  |
|  |  | r3(A) |
| w1(B) |  |  |
|  |  | w3(A) |
|  | r2(B) |  |
|  | w2(B) |  |

*Here we watch the table above, and find that:*

* *r1(B); w1(B); are right before r2(B); w2(B);*
* *r1, w1, r2, w2 all proceed the parameter B;*
* *r1(B) right before w2(B) and w1(B) is right before r2(B);*

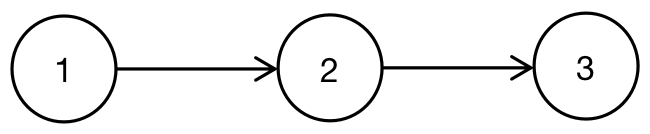
According to the conditions above, we can tell that T1 < s T2, Transaction T1 is right before Transaction T2. Similarly, we can also tell that T2 < s T3, which is to say Transaction T2 is right before T3.

*We can construct the Priority Picture of S, and to check if there exists circle to check whether the Sequence S can be Conflict Serializable. If the picture has no circle, then the sequence S is Serialized and each Topological Order of each node has Serialized Sequence.*

The picture with no circle, so the schedule S is Conflict Serializable. *The only Node Sequence or Transaction Sequence is: (T1, T2, T3).* Convert the Schedule S and make the sequence as below is possible:

*S’: r1(B); w1(B);r2(A);w2(A);r2(B);w2(B);r3(A);w3(B);*

In order to get S’ from S by exchanging the neighboring Transaction, which is completely the same sequence as the analysis before.



Img 7 - 9 The Priority Picture

*Example:*

Consider the Schedule:

*S1: r2(A); r1(B); w2(A); r2(B); r3(A); w1(B); w3(A);w2(B);*

The difference between S is that the Sequence of r2(B) move forward for three locations.

|  |  |  |
| --- | --- | --- |
| Transaction T1 | Transaction T2 | Transaction T3 |
|  | r2(A) |  |
| r1(B) |  |  |
|  | w2(A) |  |
|  | r2(B) |  |
|  |  | r3(A) |
| w1(B) |  |  |
|  |  | w3(A) |
|  | w2(B) |  |

Analysis r2(A); w2(A) and r3(A); w3(A); then we can tell that:

* *r2(A); w2(A); are right before r3(A); w3(A);*
* *r2, w2, r3, w3 all proceed the parameter A;*
* *r2(A) right before w3(A) and w2(A) is right before r3(A); The sequence includes Writing Behavior;*

*Therefore, we can tell that, Transaction T2 < s T3.*

Analysis r1(B); w2(B); then we can tell that:

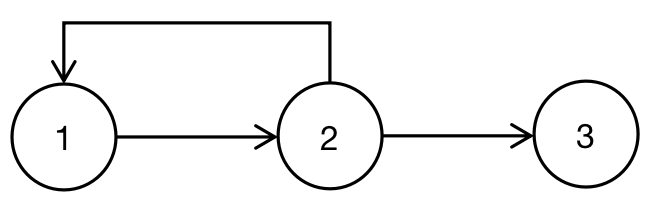
* *r1(B) is right before w2(B);*
* *r1(B) and w2(B) all proceed the parameter B;*
* *The sequence includes Writing Behavior;*

*Therefore, we can tell that, Transaction T1 < s T2.*

Also, Analysis r2(B); w1(B); then we can tell that:

* *r2(B) is right before w1(B);*
* *r2(B) and w1(B) all proceed the parameter B;*
* *The sequence includes Writing Behavior;*

*Therefore, we can tell that, Transaction T2 < s T1.*



Img 7 - 10 The Priority Picture; It’s Schedule is not Conflict Serializable.

We can tell from the priority picture that S1 is not Serializable since there has the cycle in the picture. To make it more direct, there has no such Schedule that enable Transaction T1 runs before T2 and enable Transaction T2 runs before T1.

Chapter 7.2.3 Reason Why Priority Picture Testing Take Effect

*Analysis:*

If in a series of Transactions, which is from T1 -> T2 -> T3 -> ...-> Tn -> T1, then during Serialized Schedule, T1 is before T2, T2 is before T3, T3 is before T4, ... till Tn, but Tn is asked to be before T1, therefore, we can conclude that if there has a cycle in the Priority Picture, then this Schedule would not be Serializable Conflict.

*Reversely, as long as there has no cycle in the Priority Picture, then we can legally exchange the action sequence to change the Schedule to the Serializable Schedule.*

*Condition that we need to add one arc from Node Ti to Node Tj:*

1. *Two Behaviors separately on Transaction Ti and Tj, Ti is right before Tj.*
2. *Two Behaviors operates on the same Database Element.*
3. *There has one Write Behavior in Transaction A1 and A2.*

As long as these three conditions are not satisfied, then there should not exist such arc to identify that an Arc exists between node Tj and Tj.

*Conclusion:*

*Each Schedule with Non - Cycle Priority Picture is Conflict Serializable.*