## 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

Chapter 7.2.3 Reason Why Priority Picture Testing Take Effect