Chapter 7.5 One Architecture of Lock Schedule

*Introduction:*

After knowing several different Locking Mechanism, then we need to learn how to operate on the Schedule. Here, we only consider one simple Schedule based on *principles* below:

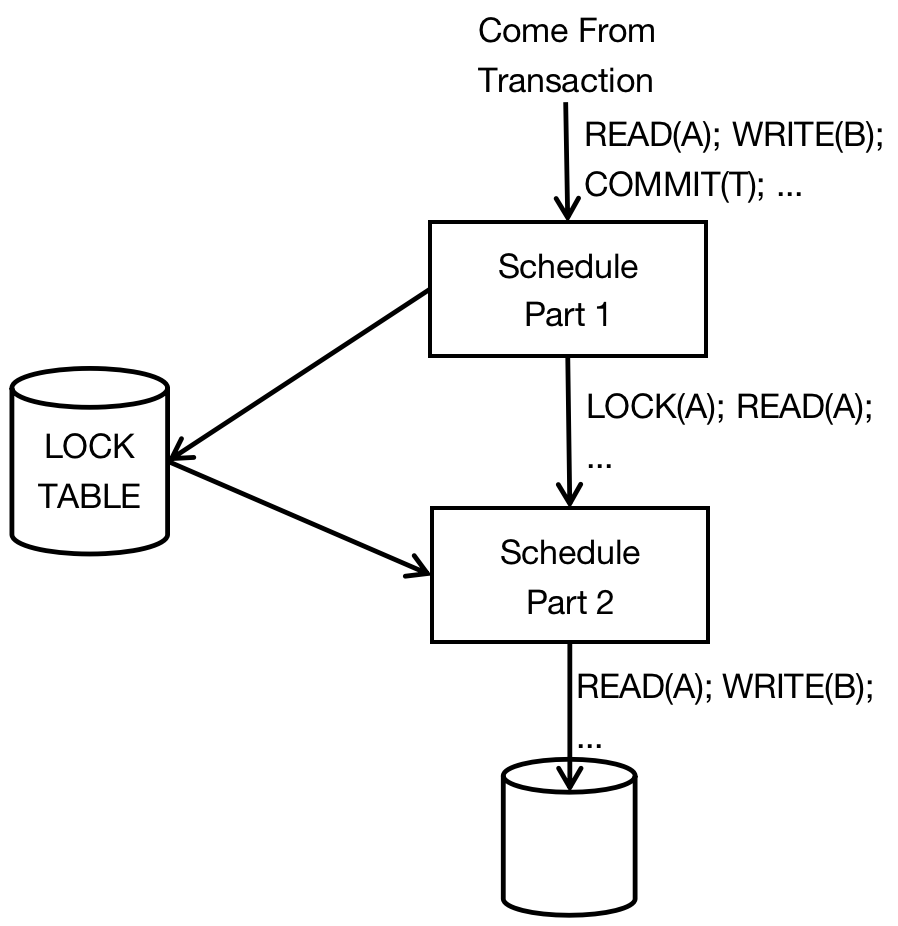
1. *Transaction itself would not apply Lock or we can rely Transaction to do this. The main Task of Schedule is to insert Lock to the Behavior when Transaction wants to read, write and other actions.*
2. *Transaction itself would not release Lock, but Schedule itself to release Lock when Transaction Management tells Schedule that Transaction will be committed or aborted.*

Chapter 7.5.1 Schedule that Insert Lock Behavior

*Introduction:*

The picture below depicts one Schedule which consists of two parts, and schedule accepts read / write and abortion requests from Transaction.

* *Schedule maintains one Lock Table, although the Lock Table is stored as Secondary Storage Data, but it may be located partially or as a whole in the Database System.*
* *The Secondary Storage Data that Lock Table used is not any part of Buffer Areas of Query Execution or Logging.*
* *Lock Table is another part of DBMS, and Operation System would assign the space for it just like other Code and Data in DBMS.*

**

*Principle:*

Normally, the Transaction Request is sent and executed in the Database System through Schedule. But sometimes, Transaction may need to wait one Lock and be delayed, and the behavior can not be sent to the Database. Two parts of Schedule work as below:

1. The first part of Schedule accepts Request generated from Transactions, and insert the appropriate Lock Actions before all Database System Access Operations such as Read, Write, Increment and Update Operations. After that, the Access Operations are sent to the second part of Schedule. No matter whatever Lock Module the Schedule uses, then the first part of Schedule must choose the appropriate Lock Method for the Transaction.
2. The second part of Schedule accepts Lock Operation and Database Access Operations, and execute each of them correctly. If Second Part of Schedule accepts one Lock or Database Access request, then it would decide whether Transaction T would be delayed because of the Transaction can not be granted some Lock. If the condition has been satisfied, then this behavior would be added into one Action List that belongs to Transaction T. If Transaction T would not be delayed, then:
   1. *If the behavior is Database Access, then this action would be sent to Database System and executed.*
   2. *If the second Part of Schedule receive one Lock Operation, then it will check the Lock Table to decide whether the Lock can be granted or not.*
      1. *If yes, modify Lock Table, and includes this Lock into Lock Table.*
      2. *If not, then there must has one tuple to signify that this Lock has been granted. So the Schedule would delay Transaction T till it get the Lock.*
3. When Transaction is being committed or aborted then, the Transaction Management would notify the first part of Schedule, and the first part of Schedule would then release all Locks that belongs to Transaction T. If any transactions is waiting for any Lock, then the First Part would wait to the Second Part.
4. When the Second Part of the Transaction is notified that the Lock can be granted, then it would decide one or multi - Transactions that can get this Lock. The Transaction that get the Lock would execute Behaviors as much as possible till reaches another un - granted Lock Behavior.

*Example:*

* When there has only one type of Lock, then the work of Schedule would be easy. As long as it sees the behavior of Database Element X, then it would insert Lock for Database Element X in Transaction if it has not been granted Lock for Database Element X.
* When Transaction is committed or aborted, then the first part of Schedule would forget all about Transaction after it releases the Lock, so main memory would not grow without limit.

*Example:*

When there have several types of Lock, then the Schedule would know what happen before hand. Consider the example before, however we have not granted any Lock to Behaviors.

*T1: r1(A); r1(B); w1(B);*

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

The message that is sent to the first part of Schedule not only includes write and read request, it also include indications that the same element would have about the following behaviors. Especially, when r1(B) is sent to the Schedule, then Schedule would need to know whether there has w1(B) behavior.

*Several methods to get such message:*

* *If Transaction is one Query, then we know it would not write anything.*
* *If Transaction is one SQL Database System update command, then Query Processor needs to make sure Database Element that may be wrote and read.*
* *If Transaction is the Program that uses the embedded SQL, then compiler can access all SQL statement, and also makes sure the Database Element that may be wrote later.*

|  |  |
| --- | --- |
| *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:*

Assume that Transaction Sequence would be like above, then Transaction T1 sends r1(A) first. Since in the future the Lock would not be upgraded, and the Schedule would insert sl1(A) before r1(A). The next step, the request about r2(A) and r2(B) comes to the Schedule. Since in the future, Lock would not be upgraded, then the first part of Schedule would send Behavior Sequence sl2(A); r2(A); sl2(B); r2(B); to the second part of Schedule.

Then behavior r1(B) and warning information reaches Schedule. The first part of Schedule send ul1(B); r1(B) to the Second Part of Schedule. And find that it can grant the Upgrade Lock to Transaction T1 for B, since there has only Shared Lock on Database Parameter B.

When behavior w1(B) reaches the Schedule, then the first part of Schedule sends xl1(B); w1(B) to the second part of Schedule. But, the second part of Schedule can not agree the request of xl1(B), since there has one Shared Lock on Database Parameter B for Transaction T2. This series of Transaction would be delayed, and the second part of Schedule would store them and wait to execute.

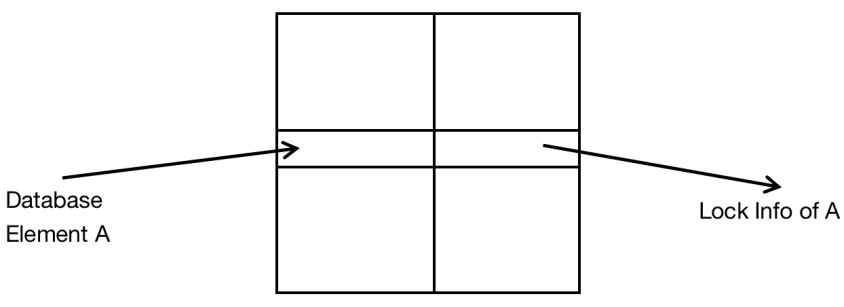
When second part of Schedule is informed this message, and it found that the Transaction T1 can get xl1(B) Lock. So it adds the Lock into Lock Table, and execute all actions that have been delayed of Transaction T1.

In the end, then Transaction T1 finishes.

Chapter 7.5.2 Lock Table

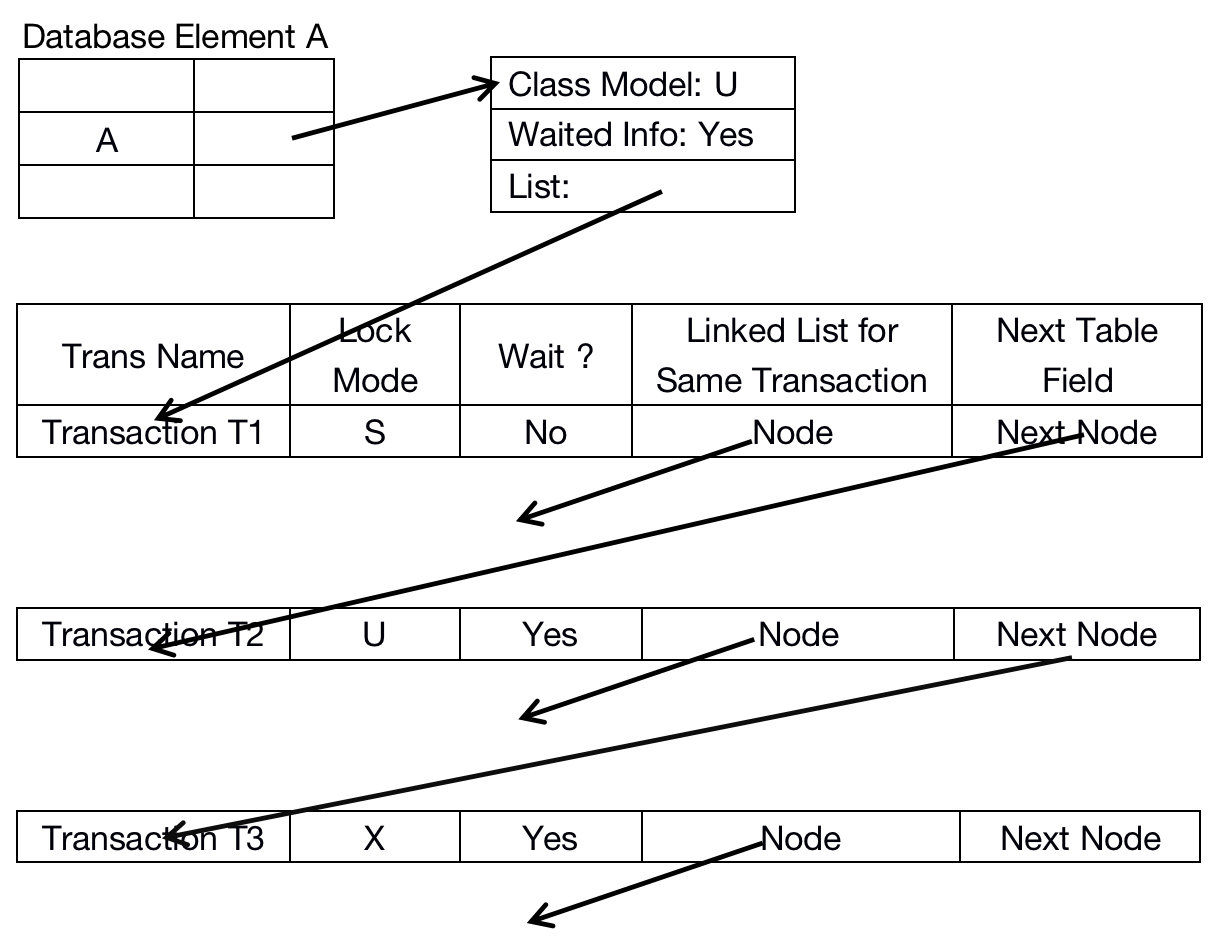
*Definition:*

*Lock Table is one Relation Table which is used to link the Database Element and Lock Information for this Database Element.*



* The Lock Table can be realized by Hash Table, and address of the Database Element is used as Hash Key.
* The Database Element that has not been locked would not appear in the Table, therefore the size of Lock Table only compared with the Locked Database Element, but not the whole Database System.

In Lock Table, we can find all these information:



Structure of Lock Table

1. Class Mode generalizes the most serious condition when the Transaction applies a new Lock for Database Parameter A. We are not use the requirement of Lock compared with other Transaction of the same Database Element, but just compare with the Class Mode to simplify whether to grant or deny the decision. In the Shared - Exclusive - Update Lock Model, the rule is simple:

*Class Mode represents:*

* 1. *Lock S presents that the only Lock that has been granted is shared Lock.*
  2. *Lock U presents one Update Lock but maybe one or more Shared Lock.*
  3. *Lock X presents one Exclusive Lock and has no other kinds of Lock.*

1. The waited Byte presents that at least one Transaction is waiting for the Lock on Database Element A.
2. One List describes all Locks or Locks that granted on Database Element A, or those Transactions that are waiting for the Locks that granted on Database Element A. Each list contains the following useful information:
   1. The name of Transaction that has Locks or waiting for the Locks.
   2. The Lock Mode.
   3. Whether the Transaction gets the Lock or just waits for the Lock.
3. Two Link Nodes. One is used to link the List itself while another one is to Link all transactions that belongs to one specific Transaction. The latter Lock is useful when the Transaction is aborted or committed, and make us easier to find all locks that need to be released.

*Deal with Locking Request:*

Assume that Transaction T makes request on Database Element A. If there has no Lock Table Field on A, then the corresponding Table Field can be created and the request is granted. If there do has the Lock field on Database Element A, then we need to use it to instruct us to make decision on whether to grant the Lock or not.

* *Find the Class Mode, which is ‘U’ or ‘Update’ in the Table. Once there has Update Lock on the table, then all other Locks can not be granted. So this request is denied, and add the corresponding list in the Transaction List and present that Transaction T applies the Lock, and the value of ‘Wait ?’ equals to YES.*
* *If Class Mode equals to ‘X’, then the same thing would happen.*
* *If Class Mode equals to ‘S’, then the other Shared Lock or Update Lock can be granted. Under this kind of situation, the value of ‘Wait ?’ equals to ‘No’, and if it is the Update Lock, then the value of Class Mode equals to ‘U’, otherwise, the value of Class Mode keeps to ‘S’.*

*No matter whether the Lock is granted or not, then they can be linked by two types of Node.*

*Deal with Release:*

Assume that the Transaction T releases the Database Value A, then everything about Database Element A would be deleted in the Transaction List.

* *If Transaction T keeps the type of Lock with totally different type with Class Mode (For example, Transaction T keeps Shared Lock and Class Mode is U.), then we do not need to change the type of Class Mode.*
* *Otherwise, if the Lock of Transaction T is under Class Mode, then we have to check the whole list to find the new Class Mode.*

*If the value of ‘Waiting’ equals to ‘yes’, then we need to grant one or more Locks in the Lock List, there have several strategy which has its own advantages:*

1. *First come First Service* - Grant the Lock Request for the Transaction that waiting the longest time. This strategy can ensure that starve to death would not happen, which means that one Transaction would wait for the Request Lock forever.
2. *Shared Lock First* - Grant all waiting Shared Locks. Then if there has waiting Updated Locks, then grant a more newer Lock. Only when there has no other Lock waiting then grant Exclusive Lock. This strategy enables the Transaction which is waiting for ‘U’ or ‘X’ Lock to starve to death.
3. *Upgrade Lock First* - If there has one Transaction which is keeping the U lock and waiting to upgrade to X Lock, then grant the X Lock first. Otherwise, use the three strategies above first.