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:

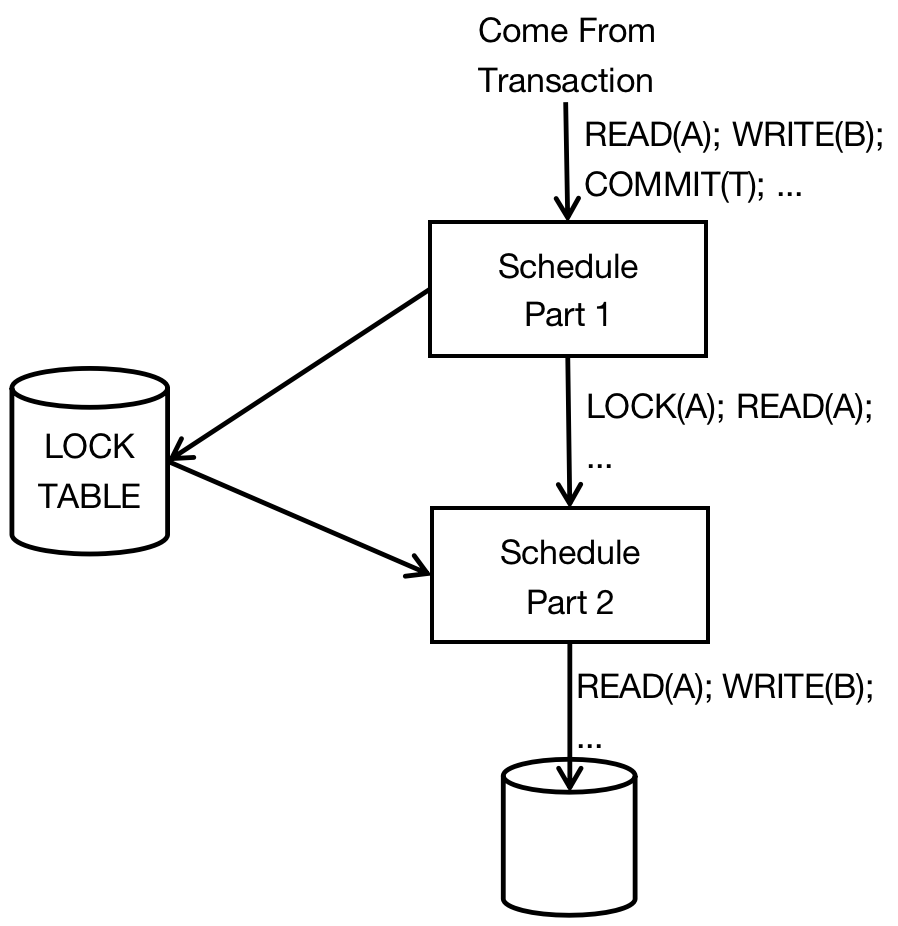
1. *Transaction itself would not apply Lock or we can rely Transaction to do this. The main Task of Schedule is to insert Lock to the Behavior when Transaction wants to read, write and other actions.*
2. *Transaction itself would not release Lock, but Schedule itself to release Lock when Transaction 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 the 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 ( r2(A) and r2(B) ) comes to the Schedule. Since in the future, the Lock would not be upgraded, then the first part of Schedule would send the Behavior Sequence sl2(A); r2(A); sl2(B); r2(B);

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

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

Chapter 7.5.2 Lock Table