Chapter 7.6 Level of Database Element

In this Chapter, we would focus on two problems when there has Tree Structure in Data.

1. *First Tree Structure is the Level Structure of Lockable Database Element.* We discuss how could Lock enables bigger element, just as Lock on Relation, and Lock that contains smaller element. (For example, the block that contains several blocks in Relation, or single tuple.)
2. *Another important Level in Currency Control is the Data that organized as Tree.* One important example is B - Tree Index. We can see the node in B - Tree as the Database Element, but if we do this, then the Lock Mode so far would be poor performance, and we need to use a newer method.

Chapter 7.6.1 Lock with Multi - Granularity

*Background:*

Since the different definition of Database Element in total different system, such as tuple, page, and Relation. Sometimes, the smaller Database Element would be better, while sometimes, the bigger Database Element would be better, which depends.  
*Example:*

Take the Database in Bank as an example.

* Take the whole Relation as Database Element, therefore there has only one Lock for the whole Relation, then the system only permits tiny concurrency. Since the most of Transactions need to add/minus Bank Balance, so most of Transaction would need one Exclusive Lock on the Bank Account. At one time, there only one deposit or draw money transaction can happen, no matter there has how many processor.
* The better method would to lock on the single page or data. Therefore, several accounts that corresponding to different block can be updated concurrently.
* The extreme method would choose to lock on the single tuple which may cost too much, although under this kind of situation, the whole account collection can solve them together.

*Example:*

Consider to take the File as Database. Since these file can not be accessed at the same time, and most of Transactions would just search the whole File. So the wise choice would to choose the whole File as the Database Element. Since most of Transactions can only be read, then Lock can be used to avoid reading the edited File.

If we use the much smaller Lock, such as picture, statement or word, then basically, lock can not bring any benefits but can add cost. The smaller granularity lock can support the only activity is that two people can modify two parts of the file together.

*Example:*

Some applications can use the bigger granularity lock and smaller granularity lock. For example, Database System in the Bank needs Lock for block or tuple, but it may be used to audit sometimes and need the Lock for the Whole Account Relation.

In order to calculate some aggregations for the Account Relation and get Shared Lock for the whole Relation, and in the same time, get Exclusive Lock for the single Account Tuple, this would cause the Non - Serializable Behavior. *The reason is that Aggregation Query may read the frozen Relation and at this time, the Relation can be changing now.*

Chapter 7.6.2 Warning Lock

*Definition:*

By using the Warning Lock, we can solve the different granularity problem. Such Lock is especially useful when we solve problem that Database Elements has Looped or Level Structure, just as the picture below. There have three kinds of Database Elements:

1. *Relation is the biggest Lockable Element.*
2. *Each Relation consists of one or more blocks or pages, and there have tuples in each Relation block or page.*
3. *Each block contains one or more tuples.*

*In the Level of Database Element, the Lock Management Rule consists of Warning Protocol, it includes ‘Normal’ Lock and ‘Warning’ Lock.*

* *Normal Lock Mode - The Lock Mode of S and X (The Shared and Exclusive Lock.)*
* *Warning Lock Mode - By adding prefix ‘I’ to represent ‘Intention’. For example, ‘IS’ represents the Intention to get the Shared Lock on the Sub - Element.*

*The Rule of Warning Protocol:*

1. If needs to add ‘S’ Lock or ‘X’ Lock on any Element, we must start from the root of Level Structure.
2. If we are at the Database Element that is waiting to be locked, then we do not need to search. We just ask for ‘S’ Lock or ‘X’ Lock on that Element.
3. If the element that we want to Lock is much more lower of the whole Level Structure, then we can add the Warning Lock on this Lock, which is to say, if we want to get the Shared Lock on the Sub - Element of the current Element, then we need to make request on the ‘IS’ Lock for the Element; If we want to get the Exclusive Lock on the Sub - Element of the current Element, then we need to make request on the ‘IX’ Lock for the Element. When all locks for the current Element has been granted, then we can go forward to the Sub - Element. Then we repeat the Step 2 and 3, till we reach the final node.

*The Corresponding Compatibility Matrix:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *IS* | *IX* | *S* | *X* |
| *IS* | *YES* | *YES* | *YES* | *NO* |
| *IX* | *YES* | *YES* | *NO* | *NO* |
| *S* | *YES* | *NO* | *YES* | *NO* |
| *X* | *NO* | *NO* | *NO* | *NO* |

Consider all table fields:

* *IS Column:*

When we want to request Lock IS on the current Node, then we intend to grant the Shared Lock on its following Sub - Node. When other Transactions have already acclaimed to write in the new Element for the Whole Database, there would some issues happen. So in the ‘X’ row, we can see that the corresponding field value equals to ‘NO’.

Attention that, if there has another Transaction wants to write on Sub - Element, and this indicates by using ‘IX’ Lock on Database Element N, then we can withstand to grant ‘IS’ Lock on Database Element N. If these Transactions relates on the same Element, then we allow to solve the conflict in the Low Level.

* *IX Column:*

If we want to write the Sub - Element for the Node N, then we must avoid writing and reading on Database Element N. So, we can tell that in the row of ‘S’ and ‘X’, the corresponding value equals to ‘NO’.

We can solve conflict problems in Low Level of Tree Structure, so granting the Lock of ‘IS’ and ‘IX’ are all enabled.

* *S Row:*

Since ‘S’ Lock would not conflict with other Read Operations on the Database Element N, or read on the Sub - Element of N, so granting the Lock of ‘S’ is allowed. In other two situations ‘X’ and ‘IX’ represent that some Transaction may need to write on the Database Element N, so the corresponding value of ‘S’ equals to ‘NO’.

* *X Row:*

For ‘X’ row, it only has ‘NO’. If other Transactions has the right to read or write on Database Element N, then the Database Element N can not be allowed to write or read.

*Example:*

Consider the Relation:

*Movie(title, year, length, studioName)*

Let’s consider there has Lock on the whole Relation or the single tuple. From Query:

*SELECT \**

*FROM Movie*

*WHERE title = ‘King Kong’;*

Starts from that the Transaction T1 gets IS Lock for whole Relation, then this Transaction turns to the single tuple ( There have three film tuples with the name of ‘King Kong’ ), and for each of them grant ‘S’ Lock.

Assume that there has another Transaction starts when Transaction T1 still continues, it changes the year field of one tuple:

*UPDATE Movie*

*SET year = 1939*

*WHERE title = ‘Gone With the Wind’;*

Now T2 needs ‘IX’ Lock for that Relations, since it intends to write a new value of this tuple; ‘IX’ Lock is compatible with ‘IS’ Lock in the Transaction T1, so it can be granted seen from Compatible Matrix. Transaction T2 comes to ‘Gone With the Wind’ tuple, find there has no Lock, and then it gets ‘X’ Lock and rewrite the tuple.

If Transaction T2 tired to write new value in one tuple among ‘King Kong’ films, it must wait for Transaction T1 to release ‘S’ Lock, since Lock ‘S’ and ‘X’ are not compatible.

*The Collection of Lock is as below:*

|  |  |  |  |
| --- | --- | --- | --- |
| Movies | | | |
| King Kong | King Kong | King Kong | King Kong |
| T1 - S | T1 - S | T1 - S | T2 - X |

Chapter 7.6.3 Phantom and Insertion Correctly

*Introduction:*

1. When Transaction tries to create one new Sub - Element of the Lockable Element, sometimes it may goes wrong. The question is that we can only lock on any existent Tuple. There has not any other simple methods to lock any in-existent Database Element which would be inserted later. Below is one simple example:

*Example:*

Relation Movie and the Transaction T3 is the first transaction to be executed, it queries:

*SELECT SUM(length)*

*FROM Movie*

*WHERE studioName = ‘Disney’*

Transaction T3 needs to read all tuples whose studioName equals to ‘Disney’, so it mostly gets ‘IS’ Lock first on the Movie, and for each tuple in the File it requires ‘S’ Lock.

Now, Transaction T4 appears and insert one new Disney Film. It looks like Transaction T4 does not need Lock, but it has made the result different.

To be more precise, assume that D1 and D2 films are already exist as Disney Film, but D3 is newly inserted Disney File. L is the total length of Disney Film which is executed by Transaction T3, and there also exists one consistent constraint which is used to indicate that the variable L is the value that last time calculated. Then below is the event sequence:

*r3(D1); r3(D2); w4(D3); w4(X); w3(L); w3(X);*

Here, we use w4(D3) to represent that the Transaction T4 creates Database Tuple D3. Especially, the value of L equals to sum of D1, D2 and D3, and these three films are the current Disney Film.

The question here is that new Disney Film has one phantom file, and the film has not been locked, because the film does not exist when it gets Lock. But there has one simple method to avoid the happening of Phantom.

*We must take the insertion and deletion of tuples as write on the whole Relation. Transaction T4 has already got ‘X’ Lock for Movie and Transaction T3 has already locked the whole Relation by using ‘IS’ Lock, and this Lock type is not compatible with ‘X’ Lock, so Transaction T4 must wait for the end of Transaction T3.*