Concurrency Control in DBMS

Concurrency control concept comes under the Transaction in database management system (DBMS). It is a procedure in DBMS which helps us for the management of two simultaneous processes to execute without conflicts between each other, these conflicts occur in multi user systems.

Concurrency can simply be said to be executing multiple transactions at a time. It is required to increase time efficiency. If many transactions try to access the same data, then inconsistency arises. Concurrency control required to maintain consistency data.

For example, if we take ATM machines and do not use concurrency, multiple persons cannot draw money at a time in different places. This is where we need concurrency.

Advantages

The advantages of concurrency control are as follows −

- Waiting time will be decreased.
- Response time will decrease.
- Resource utilization will increase.
- System performance & Efficiency is increased.

Control concurrency

The simultaneous execution of transactions over shared databases can create several data integrity and consistency problems.

For example, if too many people are logging in the ATM machines, serial updates and synchronization in the bank servers should happen whenever the transaction is done, if not it gives wrong information and wrong data in the database.

Main problems in using Concurrency

The problems which arise while using concurrency are as follows −

- **Updates will be lost** – One transaction does some changes and another transaction deletes that change. One transaction nullifies the updates of another transaction.

- **Uncommitted Dependency or dirty read problem** – On variable has updated in one transaction, at the same time another transaction has started and deleted the value of the variable there the variable is not getting updated or committed that has been done on the first transaction this gives us false values or the previous values of the variables this is a major problem.
- **Inconsistent retrievals** – One transaction is updating multiple different variables, another transaction is in a process to update those variables, and the problem occurs is inconsistency of the same variable in different instances.

Concurrency control techniques

The concurrency control techniques are as follows –

Locking

Lock guaranties exclusive use of data items to a current transaction. It first accesses the data items by acquiring a lock, after completion of the transaction it releases the lock.

Types of Locks

The types of locks are as follows −

1. **Shared Lock** [Transaction can read only the data item values]

   It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.

   It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

2. **Exclusive Lock** [Used for both read and write data item values]

   In the exclusive lock, the data item can be both reads as well as written by the transaction.

   This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

There are four types of lock protocols available:

1. Simplistic lock protocol

It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

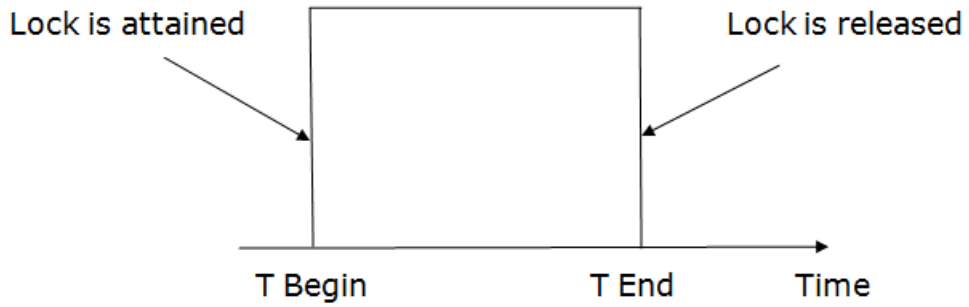
2. Pre-claiming Lock Protocol

Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.

Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.

If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.

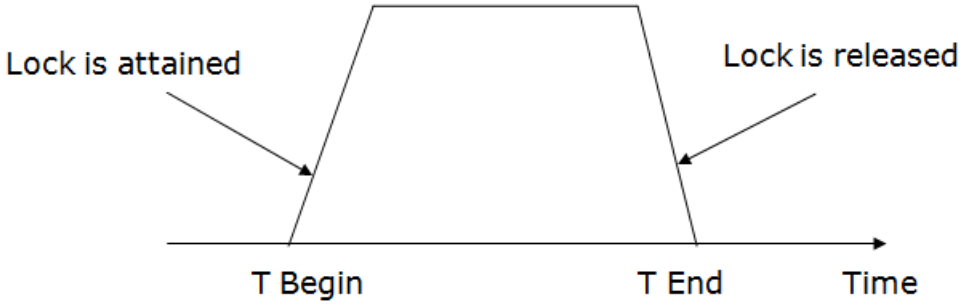
If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.



3. Two-phase locking (2PL)

- The two-phase locking protocol divides the execution phase of the transaction into three parts.
- In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.

- In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
- In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



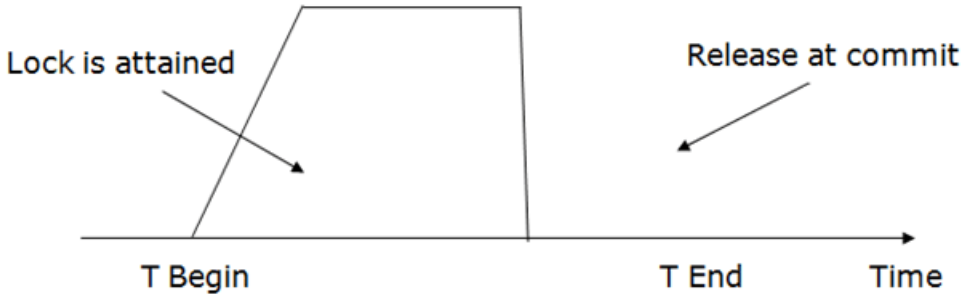
There are two phases of 2PL:

Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

1. Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
2. Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.



It does not have cascading abort as 2PL does.

Time Stamping

Time stamp is a unique identifier created by DBMS that indicates relative starting time of a transaction. Whatever transaction we are doing it stores the starting time of the transaction and denotes a specific time.

This can be generated using a system clock or logical counter. This can be started whenever a transaction is started. Here, the logical counter is incremented after a new timestamp has been assigned.

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.
- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction Ti issues a Read (X) operation:

- If W_TS(X) >TS(Ti) then the operation is rejected.
- If W_TS(X) <= TS(Ti) then the operation is executed.
- Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction Ti issues a Write(X) operation:

If TS(Ti) < R_TS(X) then the operation is rejected.

If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Where

TS(TI) denotes the timestamp of the transaction Ti.

R_TS(X) denotes the Read time-stamp of data-item X.

W_TS(X) denotes the Write time-stamp of data-item X.

Advantages and Disadvantages of TO protocol:

- TO protocol ensures serializability since the precedence graph is as follows:

Image: Precedence Graph for TS ordering

- TS protocol ensures freedom from deadlock that means no transaction ever waits.
- But the schedule may not be recoverable and may not even be cascade- free.