DBMS Concurrency Control

Concurrency Control is the management procedure that is required for controlling concurrent execution of the operations that take place on a database.

But before knowing about concurrency control, we should know about concurrent execution.

Concurrent Execution in DBMS

* In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.
* While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.
* The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database. Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

Problems with Concurrent Execution

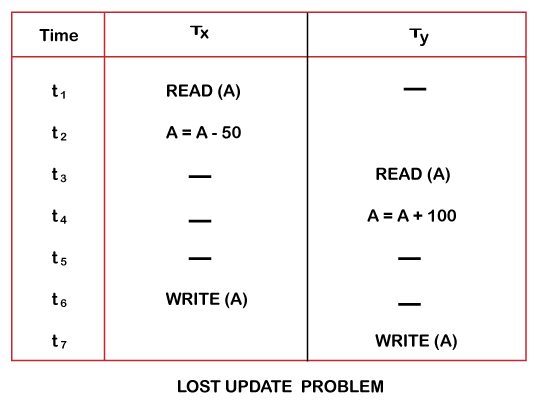
In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

Problem 1: Lost Update Problems (W - W Conflict)

The problem occurs *when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent*.

**For example:**

**Consider the below diagram where two transactions TX and TY, are performed on the same account A where the balance of account A is $300.**



* At time t1, transaction TX reads the value of account A, i.e., $300 (only read).
* At time t2, transaction TX deducts $50 from account A that becomes $250 (only deducted and not updated/write).
* Alternately, at time t3, transaction TY reads the value of account A that will be $300 only because TX didn't update the value yet.
* At time t4, transaction TY adds $100 to account A that becomes $400 (only added but not updated/write).
* At time t6, transaction TX writes the value of account A that will be updated as $250 only, as TY didn't update the value yet.
* Similarly, at time t7, transaction TY writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by TX is lost, i.e., $250 is lost.

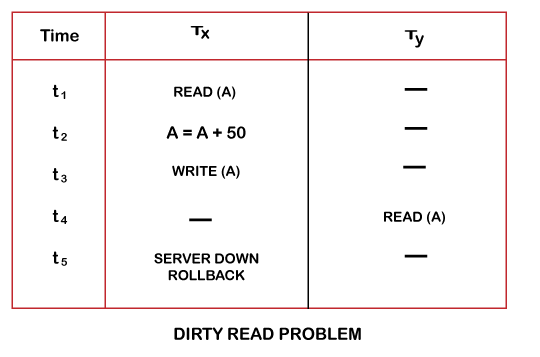
Hence data becomes incorrect, and database sets to inconsistent.

Dirty Read Problems (W-R Conflict)

The dirty read problem occurs *when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.*

**For example:**

**Consider two transactions TX and TY in the below diagram performing read/write operations on account A where the available balance in account A is $300:**



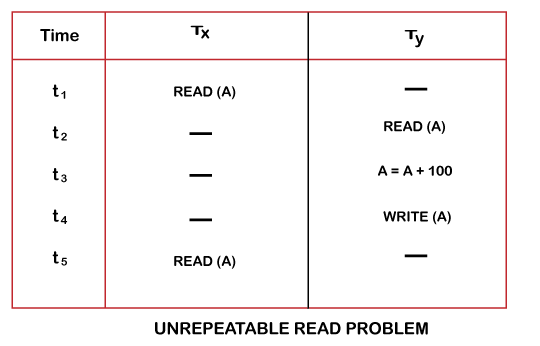
* At time t1, transaction TX reads the value of account A, i.e., $300.
* At time t2, transaction TX adds $50 to account A that becomes $350.
* At time t3, transaction TX writes the updated value in account A, i.e., $350.
* Then at time t4, transaction TY reads account A that will be read as $350.
* Then at time t5, transaction TX rollbacks due to server problem, and the value changes back to $300 (as initially).
* But the value for account A remains $350 for transaction TY as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Unrepeatable Read Problem (W-R Conflict)

*Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.*

**For example:**

**Consider two transactions, TX and TY, performing the read/write operations on account A, having an available balance = $300. The diagram is shown below:**



* At time t1, transaction TX reads the value from account A, i.e., $300.
* At time t2, transaction TY reads the value from account A, i.e., $300.
* At time t3, transaction TY updates the value of account A by adding $100 to the available balance, and then it becomes $400.
* At time t4, transaction TY writes the updated value, i.e., $400.
* After that, at time t5, transaction TX reads the available value of account A, and that will be read as $400.
* It means that within the same transaction TX, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction TY, it reads $400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Thus, in order to maintain consistency in the database and avoid such problems that take place in concurrent execution, management is needed, and that is where the concept of Concurrency Control comes into role.

Concurrency Control

Concurrency Control is the working concept that is required for controlling and managing the concurrent execution of database operations and thus avoiding the inconsistencies in the database. Thus, for maintaining the concurrency of the database, we have the concurrency control protocols.

Concurrency Control Protocols

The concurrency control protocols ensure the *atomicity, consistency, isolation, durability* and *serializability* of the concurrent execution of the database transactions. Therefore, these protocols are categorized as:

* Lock Based Concurrency Control Protocol
* Time Stamp Concurrency Control Protocol

We will understand and discuss each protocol one by one in our next sections.

Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

**1. Shared lock:**

* 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:**

* 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.

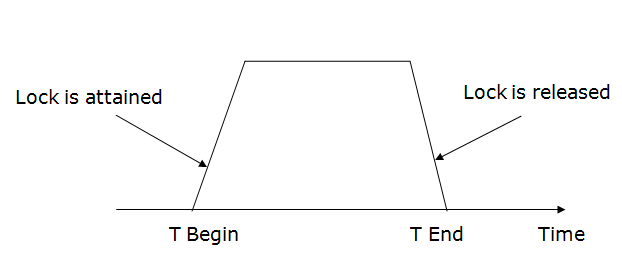
There are four types of lock protocols available:

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. 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:

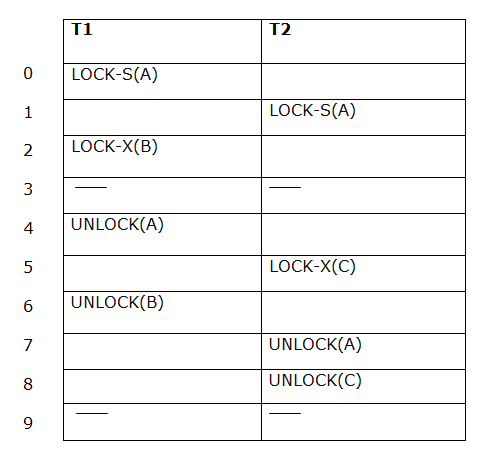
**Growing phase:** In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

**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.

**Example:**



The following way shows how unlocking and locking work with 2-PL.

**Transaction T1:**

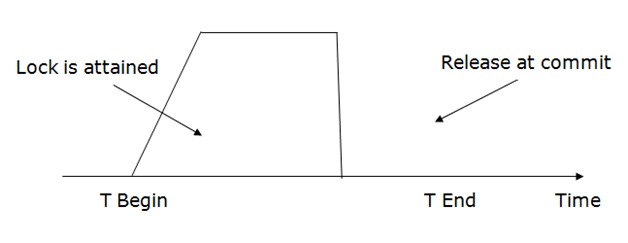
* **Growing phase:** from step 1-3
* **Shrinking phase:** from step 5-7
* **Lock point:** at 3

**Transaction T2:**

* **Growing phase:** from step 2-6
* **Shrinking phase:** from step 8-9
* **Lock point:** at 6

3. Strict Two-phase locking (Strict-2PL)

* The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
* The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
* Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
* Strict-2PL protocol does not have shrinking phase of lock release.



It does not have cascading abort as 2PL does.

Timestamp Ordering Protocol

* 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.