Concurrency Control in DBMS

When more than one transactions are running simultaneously there are chances of a conflict to occur which can leave database to an inconsistent state. To handle these conflicts we need concurrency control in DBMS, which allows transactions to run simultaneously but handles them in such a way so that the integrity of data remains intact.

Concurrent Execution in DBMS

In a multi-user environment, multiple users are allowed to access data simultaneously. These users can send requests at the same time, which means the DBMS is serving multiple requests from different users simultaneously. This is called **concurrent execution in DBMS**.

In concurrent execution, there are several operations performed on the data items in database concurrently. There can be multiple read requests, write requests or combination of both. Serving a read request while another user is making change in the database can cause several issues .

The goal of the concurrent execution is to allow multiple transactions to execute simultaneously in such a way, that each executing transaction doesn’t affect the other transaction in any way.

Issues with the concurrent execution

Let’s take an example to understand what are the issues that can arise when transactions are executing concurrently.

Conflict Example

You and your brother have a joint bank account, from which you both can withdraw money. Now let’s say you both go to different branches of the same bank at the same time and try to withdraw 5000 INR, your joint account has only 6000 balance. Now if we don’t have concurrency control in place you both can get 5000 INR at the same time but once both the transactions finish the account balance would be -4000 which is not possible and leaves the database in inconsistent state.
We need something that controls the transactions in such a way that allows the transaction to run concurrently but maintaining the consistency of data to avoid such issues.

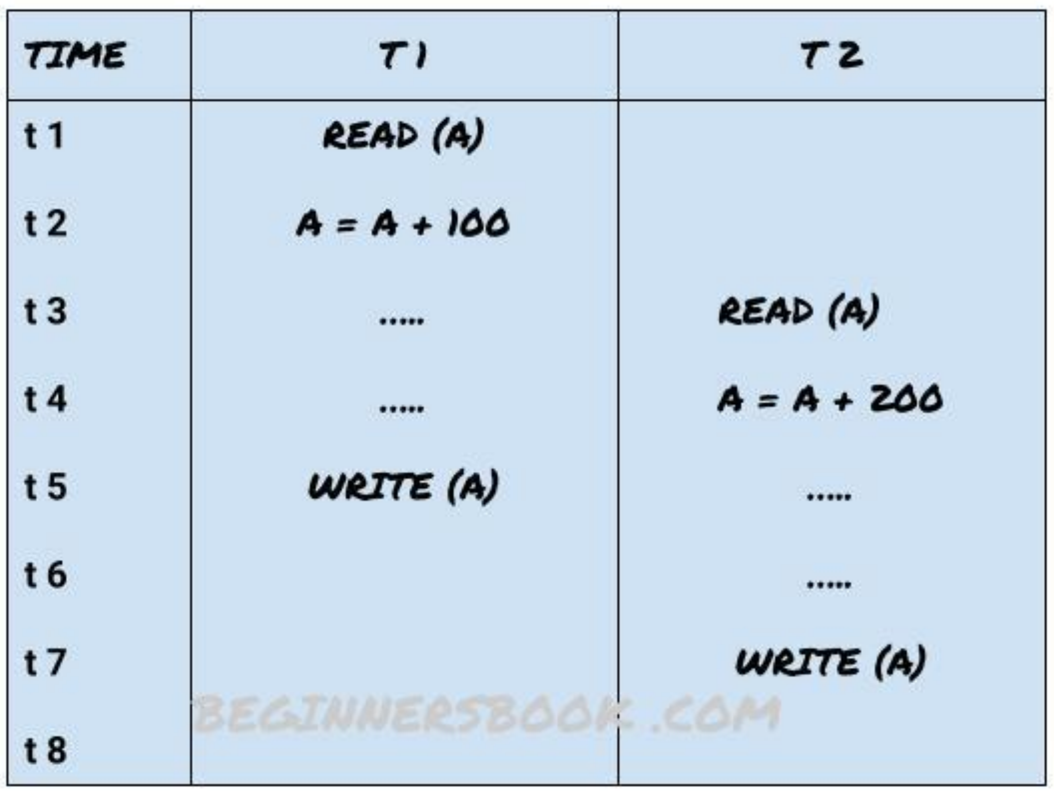
Problems with the concurrent execution

There are two main operations in the database, **read and write**. If we somehow manage these two operations on a data item in such a way, that the database always end up being in a consistent state, then we can say that this is a perfect concurrency control scenario.

Problem 1: W-W Conflict – Lost Update Problem

This conflict occurs when two transactions perform a write operation on a same data item in database in such a way that the database ends up in an inconsistent state. This problem is also known as Write- Write conflict or **W-W conflict** or **lost update problem**.

Let’s take an example to understand this problem:



TIME	T1	T2
t1	READ (A)	
t2	A = A + 100	
t3	.....	READ (A)
t4	.....	A = A + 200
t5	WRITE (A)	.....
t6		.....
t7		WRITE (A)
t8		

LOST UPDATE PROBLEM

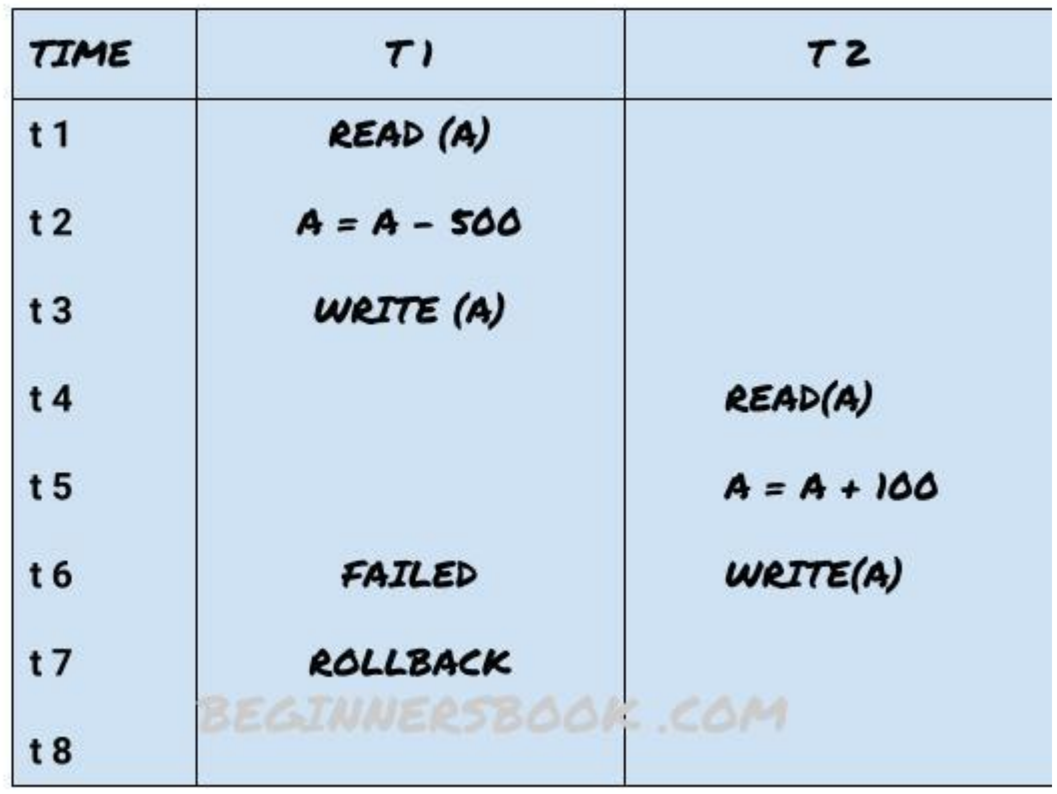
- In this example, at time t1, the transaction T1 reads the value of data item A. Let’s say the value of A is 1000.
- At time t2, transaction T1 adds 100 to the data item A, the value of A becomes 1100. This is value is yet to be updated in database as no write operation has been performed yet.
- At time t3, transaction T2 read the value of data item A, the value of A in database is still 1000 so transaction T2 reads the A value as 1000.
- AT time t4, transaction T2 adds 200 to data item A, in transaction T2 the value of A becomes 1200 but this value is yet to be updated in database.
- At time t5, transaction T1 writes the value of A in database, since the value of A is 1100 according to transaction T1, it updates the value of A in database to 1100.
- At time t7, transaction T2 writes the value of A in database, since the value of A is 1200 in T2, it updates the value of A to 1200 in database.

Lost Update: Initial value of A was 1000, if T1 is adding 100 and T2 is adding 200, the value of A in database should be 1300 at the end of execution of both of these transactions. However as you can see the value of A is 1200 in this case. **This is because the update made by transaction T1 is lost**.

Problem 2: W-R Conflict – Dirty Read Problem or temporary update problem.

This conflict occurs when a transaction makes changes to a data item in the database and the transaction fails after making the change. However before the failed transaction is rolled back, another transaction reads the value updated by failed transaction. This is called **Dirty Read** in DBMS or **W-R (Write – Read)** Conflict.

Let's take an example to understand this conflict:



TIME	T1	T2
t 1	READ (A)	
t 2	A = A – 500	
t 3	WRITE (A)	
t 4		READ(A)
t 5		A = A + 100
t 6	FAILED	WRITE(A)
t 7	ROLLBACK	
t 8		

DIRTY READ PROBLEM

- At time t1, transaction T1 reads the value of A, let's say the value of A is 1000. T1 read A as 1000.
- At time t2, T1 deducts 500 from A, the value of A becomes 500.
- At time t3, T1 make the changes in database by writing the value of A. The database value of A gets updated from 1000 to 500.

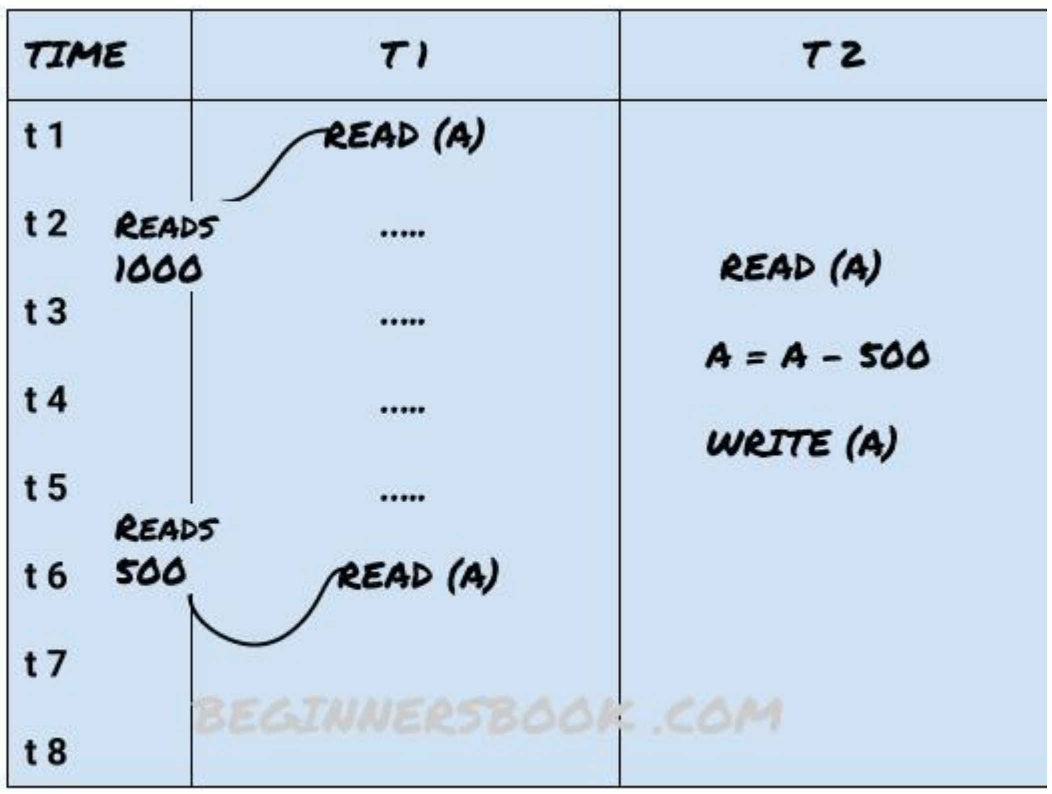
- At time t4, another transaction T2 reads the value of A which is 500 now in database.
- At time t5, T2 adds 100 and value of A is now 600 but in database it is still 500
- At time t6, Transaction T1 fails and T2 writes the value of A in database, value of A gets updated to 600 in database.
- At time t7, transaction T1 is rolled back because it is failed in previous step.

Dirty Read: Since T1 failed and rolled back, the changes made by T1 should be reverted. T2 should have read original value of A which is 1000 and at the end of T2 the value of A in database should be 1100. However as we have seen above, the value of A in database is 600. This is because T2 read an updated value by failed transaction. **This is called dirty read** and it left the database in an inconsistent state.

Problem 3: W-R Conflict – Non-Repeatable Read Problem

This conflict occurs when a transaction reads different values for the same data-item. This is also known as inconsistent retrieval or **Non-repeatable read problem**.

Let's take an example to understand this:



TIME	T1	T2
t1	READ (A)	
t2	.....	READ (A)
t3	.....	A = A - 500
t4	.....	WRITE (A)
t5	.....	
t6	READ (A)	
t7		
t8		

READS 1000
READS 500

NON-REPEATABLE READ PROBLEM

- At time t1, the transaction T1 reads the value of A as 1000.

- At time t6, the same transaction T1 reads reads the different value of A as 500.
- This is because between time t1 and t6, an another transaction made the changes in the database and updated the value of A from 1000 to 500.
- This is an issue as the transaction reads two different values for same data item, thus it is called non-repeatable read problem. This leaves the database in an inconsistent state.

Concurrency Control

Concurrency control is the technique that ensures that the the above three conflicts don't occur in the database. There are certain rules to avoid problems in concurrently running transactions and these rules are defined as the concurrency control protocols.

Concurrency control protocols

Concurrency control protocols ensure that the database remain in a consistent state after the execution of transactions. There are three concurrency control protocols:

1. **Lock Based Concurrency Control Protocol**
2. **Time Stamp Concurrency Control Protocol**
3. **Validation Based Concurrency Control Protocol**

Lock based Protocol in DBMS

A lock is kind of a mechanism that ensures that the integrity of data is maintained. It does that, **by locking the data** while a transaction is running, any transaction cannot read or write the data until it acquires the appropriate lock. There are two types of a lock that can be placed while accessing the data so that the concurrent transaction can not alter the data while we are processing it.

1.SharedLock(S)
2. Exclusive Lock(X)

Shared Lock(S): Shared lock is placed when we are reading the data, multiple shared locks can be placed on the data but when a shared lock is placed no exclusive lock can be placed.

To understand the lock mechanism let's take an **example of conflict**:

You and your brother have a joint bank account, from which you both can withdraw money. Now let's say you both go to different branches of the same bank at the same time and try to withdraw 5000 INR, your joint account has only 6000 balance.

Now if we don't have concurrency control in place you both can get 5000 INR at the same time but once both the transactions finish the account balance would be -4000 which is not possible and leaves the database in inconsistent state.

We need something that controls the transactions in such a way that allows the transaction to run concurrently but maintaining the consistency of data to avoid such issues.

Solution of the above problem using Shared lock:

For example, when two transactions are reading Steve’s account balance, let them read by placing shared lock but at the same time if another transaction wants to update the Steve’s account balance by placing Exclusive lock, do not allow it until reading is finished.

2. Exclusive Lock(X): Exclusive lock is placed when we want to read and write the data. This lock allows both the read and write operation, Once this lock is placed on the data no other lock (shared or Exclusive) can be placed on the data until Exclusive lock is released.

For example, when a transaction wants to update the Steve’s account balance, let it do by placing X lock on it but if a second transaction wants to read the data(S lock) don’t allow it, if another transaction wants to write the data(X lock) don’t allow that either.

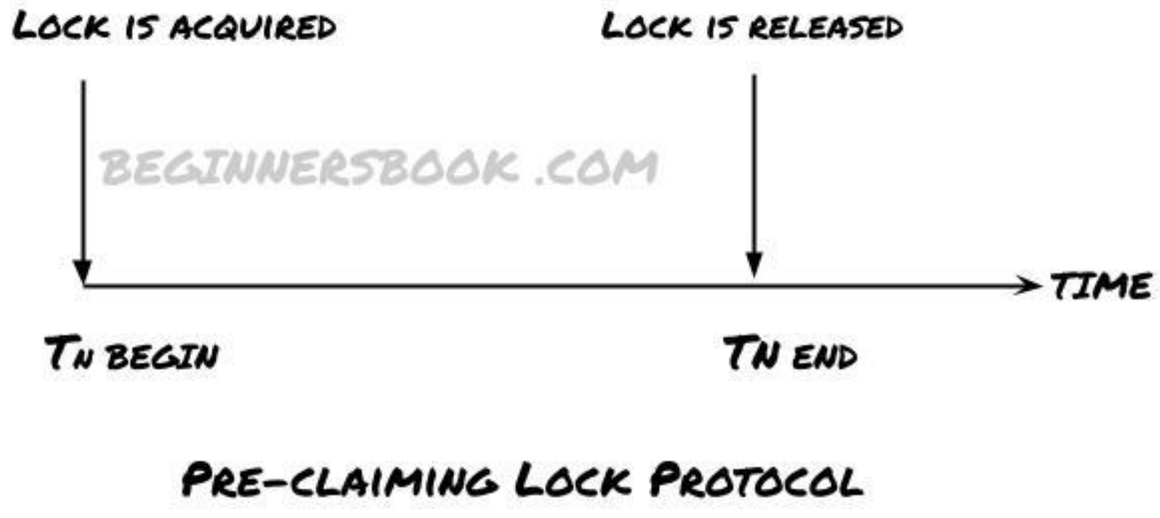
Types of Lock Protocols

1. Simplistic lock protocol

This protocol is simplest form of locking the data while a transaction is running. As per simplistic lock protocol any transaction needs to acquire the lock on the data before performing any insert, update or delete operation. The transaction releases the lock as soon as it is done performing the operation. This prevents other transactions to read the data while its being updated.

2. Pre-claiming lock protocol

- As the name suggests, this protocol checks the the transaction to see **what all locks it requires before it begins**.
- Before the transaction begins, it **places the request to acquire all the locks on data items**.
- If all the locks are granted, the transaction begins execution and **releases all the locks once it’s done execution.**
- If all the locks are not granted this **transaction waits** until the required locks are granted.



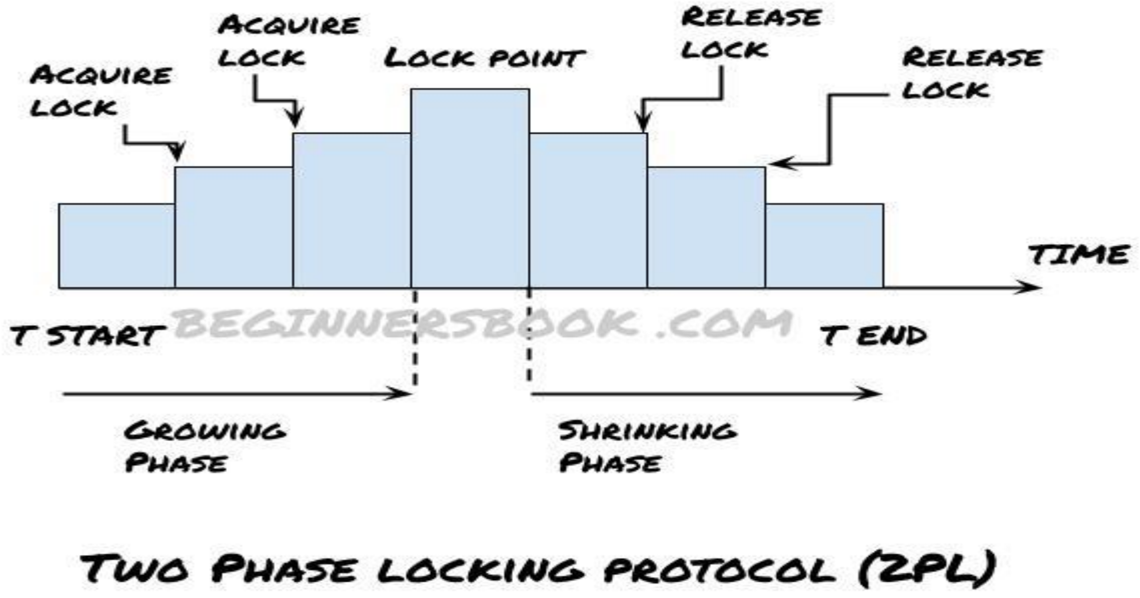
Two Phase locking protocol (2PL)

In two phase locking protocol the locking and unlocking of data items is done in two phases.

Growing Phase: In this phase, the locks are acquired on the data items but none of the acquired locks can be released in this phase.

Shrinking Phase: The existing locks can be released in this phase but no new locks can be acquired in this phase.

Note: The point at which the transaction acquires final lock and the growing phase ends is called **lock point**.



2 PL Example: Let's take an example to understand how two phase locking protocol works: In the following example there are two transaction T1 and T2 running concurrently.

Transaction T1: In this example, growing phase of T1 is from **Step 1 to Step 5**. Shrinking phase is from **Step 7 to Step 9**. **Lock point is at step 5**.

Transaction T2: Growing phase of T2 is from **Step 2 to Step 10**. Shrinking phase is from **Step 11 to Step 13**. **Lock point is at step 10**.

```
   T1          T2
       ----        ----
Step 1 lock-S(A)
Step 2 ..            lock-S(A)
Step 3 lock-S(B)
Step 4 ...           lock-S(B)
Step 5 lock-X(C)
Step 6 ..
Step 7 Unlock(A)
Step 8 Unlock(B)
Step 9 Unlock(C)
Step 10              lock-S(C)
Step 11              Unblock(A)
Step 12              Unblock(B)
Step 13              Unblock(C)
```

Strict Two Phase Locking Protocol (Strict – 2PL)

It is somewhat similar to 2PL except that it doesn't have a shrinking phase. **This protocol releases all the locks only after the transaction is completed successfully** and used the commit statement to make the changes permanent in the database.

It doesn't release locks after performing an operation on data items. It releases all the locks at the same time once the transaction commit successfully.

Validation Based Protocol

Validation based protocol avoids the concurrency of the transactions and works based on the assumption that if no transactions are running concurrently then no interference occurs. This is why it is also called **Optimistic Concurrency Control Technique**.

In this protocol, a transaction doesn't make any changes to the database directly, instead it performs all the changes on the local copies of the data items that are maintained in the transaction itself. At the end of the transaction, a validation is performed on the transaction. If it doesn’t violate any serializability rule, the transaction commit the changes to the database else it is updated and restarted.

Three phases of Validation based Protocol

1. **Read phase:** In this phase, a transaction reads the value of data items from database and store their values into the temporary local variables. Transaction then starts executing but it doesn’t update the data items in the database, instead it performs all the operations on temporary local variables.
2. **Validation phase:** In this phase, a validation check is done on the temporary variables to see if it violates the rules of serializability.
3. **Write phase:** This is the final phase of validation based protocol. In this phase, if the validation of the transaction is successful then the values of temporary local variables is written to the database and the transaction is committed. If the validation is failed in second phase then the updates are discarded and transaction is slowed down to be restarted later.

Let’s look at the timestamps of each phase of a transaction:

Start(Tn): It represents the timestamp when the transaction Tn starts the execution.

Validation(Tn): It represents the timestamp when the transaction Tn finishes the read phase and starts the validation phase.

Finish(Tn): It represents the timestamp when the transaction Tn finishes all the write operations.

This protocol uses the Validation(Tn) as the timestamp of the transaction Tn because this is actual phase of the transaction where all the checks happen. So it is safe to say that TS(Tn) = Validation(Tn).

If there are two transactions T1 & T2 managed by validation based protocol and if Finish(T1) < Start(T2) then the validation will be successful as the serializability is maintained because T1 finished the execution well before the transaction T2 started the read phase.