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 T_X and T_Y, are performed on the same account A where the balance of account A is $300.

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).
- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.
- Similarly, at time t7, transaction T_Y 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 T_X 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 T_X and T_Y in the below diagram performing read/write operations on account A where the available balance in account A is $300:

Time	T_X	T_Y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.
- Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).
- But the value for account A remains $350 for transaction T_Y 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, T_X and T_Y, performing the read/write operations on account A, having an available balance = $300. The diagram is shown below:

Time	T_X	T_Y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

- At time t1, transaction T_X reads the value from account A, i.e., $300.
- At time t2, transaction T_Y reads the value from account A, i.e., $300.
- At time t3, transaction T_Y updates the value of account A by adding $100 to the available balance, and then it becomes $400.
- At time t4, transaction T_Y writes the updated value, i.e., $400.
- After that, at time t5, transaction T_X reads the available value of account A, and that will be read as $400.
- It means that within the same transaction T_X, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction T_Y, 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
- Validation Based Concurrency Control Protocol

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

1. Two-Phase Locking Protocol: Locking is an operation which secures: permission to read, OR permission to write a data item. Two phase locking is a process used to gain ownership of shared resources without creating the possibility of deadlock. The 3 activities taking place in the two phase update algorithm are:
(i). Lock Acquisition
(ii). Modification of Data
(iii). Release Lock
Two phase locking prevents deadlock from occurring in distributed systems by releasing all the resources it has acquired, if it is not possible to acquire all the resources required without waiting for another process to finish using a lock. This means that no process is ever in a state where it is holding some shared resources, and waiting for another process to release a shared resource which it requires. This means that deadlock cannot occur due to resource contention. A transaction in the Two Phase Locking Protocol can assume one of the 2 phases:

- **(i) Growing Phase:** In this phase a transaction can only acquire locks but cannot release any lock. The point when a transaction acquires all the locks it needs is called the Lock Point.
- **(ii) Shrinking Phase:** In this phase a transaction can only release locks but cannot acquire any.

2. Time Stamp Ordering Protocol: A timestamp is a tag that can be attached to any transaction or any data item, which denotes a specific time on which the transaction or the data item had been used in any way. A timestamp can be implemented in 2 ways. One is to directly assign the current value of the clock to the transaction or data item. The other is to attach the value of a logical counter that keeps increment as new timestamps are required. The timestamp of a data item can be of 2 types:

- **(i) W-timestamp(X):** This means the latest time when the data item X has been written into.
- **(ii) R-timestamp(X):** This means the latest time when the data item X has been read from. These 2 timestamps are updated each time a successful read/write operation is performed on the data item X.
    - **Validation Based Protocol** is also called Optimistic Concurrency Control Technique. This protocol is used in DBMS (Database

Management System) for avoiding concurrency in transactions. It is called optimistic because of the assumption it makes, i.e. very less interference occurs, therefore, there is no need for checking while the transaction is executed.

- In this technique, no checking is done while the transaction is been executed. Until the transaction end is reached updates in the transaction are not applied directly to the database. All updates are applied to local copies of data items kept for the transaction. At the end of transaction execution, while execution of the transaction, a **validation phase** checks whether any of transaction updates violate serializability. If there is no violation of serializability the transaction is committed and the database is updated; or else, the transaction is updated and then restarted.
- Optimistic Concurrency Control is a three-phase protocol. The three phases for validation based protocol:
- **Read Phase:**
  Values of committed data items from the database can be read by a transaction. Updates are only applied to local data versions.

- **Validation Phase:**
  Checking is performed to make sure that there is no violation of serializability when the transaction updates are applied to the database.

- **Write Phase:**
  On the success of the validation phase, the transaction updates are applied to the database, otherwise, the updates are discarded and the transaction is slowed down.

The idea behind optimistic concurrency is to do all the checks at once; hence transaction execution proceeds with a minimum of overhead until the validation phase is reached. If there is not much interference among transactions most of them will have successful validation, otherwise, results will be discarded and restarted later. These circumstances are not much favourable for optimization techniques, since, the assumption of less interference is not satisfied.

Validation based protocol is useful for rare conflicts. Since only local copies of data are included in rollbacks, cascading rollbacks are avoided. This method is not favourable for longer transactions because they are more likely to have conflicts and might be repeatedly rolled back due to conflicts with short transactions.

In order to perform the Validation test, each transaction should go through the various phases as described above. Then, we must know about the

following three time-stamps that we assigned to transaction T_i, to check its validity:

1. Start(T_i): It is the time when T_i started its execution.

2. Validation(T_i): It is the time when T_i just finished its read phase and begin its validation phase.

3. Finish(T_i): the time when T_i end it's all writing operations in the database under write-phase.

Two more terms that we need to know are:

1. Write_set: of a transaction contains all the write operations that T_i performs.

2. Read_set: of a transaction contains all the read operations that T_i performs.

In the Validation phase for transaction T_i the protocol inspect that T_i doesn't overlap or intervene with any other transactions currently in their validation phase or in committed. The validation phase for T_i checks that for all transaction T_j one of the following below conditions must hold to being validated or pass validation phase:

Finish(T_j)<Starts(T_i), since T_j finishes its execution means completes its write-phase before T_i started its execution(read-phase). Then the serializability indeed maintained.

2. T_i begins its write phase after T_j completes its write phase, and the read_set of T_i should be disjoint with write_set of T_j.

3. T_j completes its read phase before T_i completes its read phase and both read_set and write_set of T_i are disjoint with the write_set of T_j.

Ex: Here two Transactions T_i and T_j are given, since TS(T_j)<TS(T_i) so the validation phase succeeds in the Schedule-A. It's noteworthy that the final write operations to the database are performed only after the validation of both Ti and T_j. Since T_i reads the old values of **x(12)** and **y(15)** while **print(x+y)** operation unless final write operation take place.

Schedule-A

T_j	T_i
r(x) // x=12	
	r(x)
	x=x-10
	r(y) //y=15
	y=y+10
	r(x)

T_j T_i

<validate>
print(x+y)

<validate>

w(x)
w(y)

Schedule-A is a validated schedule

Advantages:

1. Avoid Cascading-rollbacks: This validation based scheme avoid cascading rollbacks since the final write operations to the database are performed only after the transaction passes the validation phase. If the transaction fails then no updation operation is performed in the database. So no dirty read will happen hence possibilities cascading-rollback would be null.

2. Avoid deadlock: Since a strict time-stamping based technique is used to maintain the specific order of transactions. Hence deadlock isn't possible in this scheme.

Disadvantages:

1. Starvation: There might be a possibility of starvation for long-term transactions, due to a sequence of conflicting short-term transactions that cause the repeated sequence of restarts of the long-term transactions so on and so forth. To avoid starvation, conflicting transactions must be temporarily blocked for some time, to let the long-term transactions to finish.