UNIT IV - Transaction

Transaction Concepts – ACID Properties – Schedules – Serializability – **Concurrency Control – Need for Concurrency** – Locking Protocols – Two Phase Locking – Deadlock – Transaction Recovery – Save Points – Isolation Levels – SQL Facilities for Concurrency and Recovery.

Concurrency Control

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

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

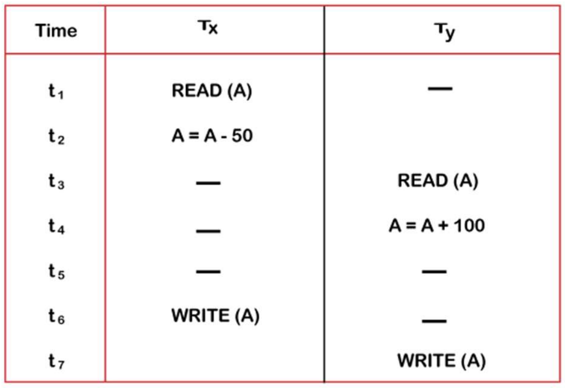
In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the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

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.



Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

- 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.

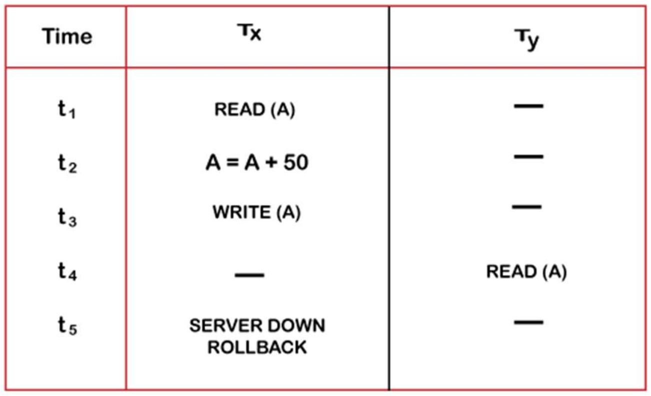
Dirty Read Problems (W-R Conflict)

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

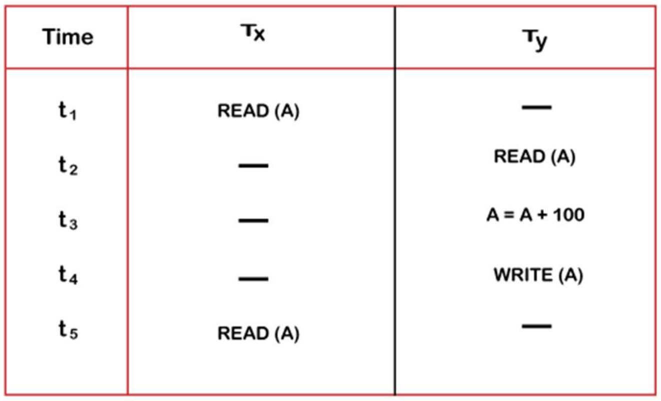
- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.
- 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.

Challenges in Concurrent Transactions

Isolation is one of the major issues which the community has to consider to achieve data consistency, integrity and good performance of a DBMS when it supports concurrent transactions. These are a few of the major difficulties:

Data Consistency:

Lost Updates: Happens when two or more transactions attempt to read the same record, modify it and then write it back, it would result in writing only one of the changes.

Temporary Inconsistencies: Raise when a transaction is reading data that another transaction is simultaneously processing, resulting in interim or unpredictable values.

Data Integrity:

Non-Repeatable Reads: Occur when one transaction retries a read operation and obtains a different value than before because another transaction has updated the value in between.

Phantom Reads: Happen when a transaction resubmit a query that returns a set of rows which meet a condition and discover that the set of rows has been changed by another transaction that was inserting or deleting rows.

Isolation Levels and Performance

Low Isolation Levels: Read Uncommitted and Read Committed for example, makes the query run faster but at the same time, bring about the possibility of dirty reads and non-repeatable reads.

High Isolation Levels: As such, we have techniques such as Repeatable Read and Serializable options that ensure data consistency but harm concurrency and system performance.

Deadlocks:

Detection and Resolution: After that, to identify deadlocks, it is necessary to monitor ongoing transactions, which are interested in resources and their allocation; the computation of these is computationally intensive, means that resolving deadlocks require the aborting of one or several transactions; this has negative impacts on the system throughput and users satisfaction.

Prevention and Avoidance: Policies or measures to address or minimize deadlocks in operating systems can reduce concurrency and hence underutilize resources.

Resource Contention

Lock Contention: If there are many transactions that require the use of locks, there is a potential to have small raw transaction rates, and long waits where there is a large transaction rate.

Hardware Resource Constraints: This generate contention issues on a limited number of CPUs, memory and I/O operation when there are a high number of concurrent transactions.

Concurrency Control Mechanisms

The use of locks is very important in DBMS concurrency control because it is used in the control of multiple transactions without allowing unsynchronized changes to the database. Some of the concepts that play an important role in these protocols are shared locks and exclusive locks in addition to two phase locking (2PL) and strict two phase locking.

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