(3) Validation Based protocol

Validation Based protocol in DBMS

Validation Based protocol is also called optimistic concurrency control Technique.

This protocol is used in DBMS for avoiding concurrency in transactions.

we have three ~~phases~~ phases'.

(1) Read phase: In this phase, the transaction T is read and executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.

② validation phase:- In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability. In this validation test is performed to determine whether changes in Actual Database can be made.

③ write phase: If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

Time Stamp is used to determine when to start validation Test

Every Transaction Ti is Associated with three Time stamps which are

Start (Ti) - It gives time when Ti start execution.

Validation (Ti)! - ~~when Ti~~ It gives Time when Ti Finishes its Read phase and starts its validation phase

Finish (Ti) ! It Gives Time when Ti Finished its execution or write phase.

If Any transaction failed in validati Test then its is ABORTED and ROLLB

T	S	V	F
	10	10-10	10.15

Timestamp odering protocol

→ The Timestamp odering 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.

→ Timestamp based protocol start working as soon as a transaction is Created

→ Let's assume there are two transactions T_1 and T_2. Suppose the transaction T_1 has entered the system at 007 time and Transaction T_2 has entered the system at 009 time.

T_1 has the higher priority, so it executes first as it is entered the system first.

→ The time stamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Basic Timestamp ordering protocol works as follows

check the following condition whenever a transaction T_i issues a Read(x) operation.

if $W\_TS(x) > TS(T_i)$ rejected

if $W\_TS(x) <= TS(T_i)$ then operation is executed.

→ Timestamps of all data items are updated.

2. Check the following condition whenever a transaction T_i issues a write (X) operation.

-> If $TS(T_i) < R\_TS(X)$ then the operation is rejected.

-> If $TS(T_i) < W\_TS(X)$ then the operation is rejected, T_i is rollback otherwise the operation is executed.

where $TS(T_i)$ denotes the timestamp of the transaction T_i

$R\_TS(X)$ denotes the read time stamp of dataitem X

$W\_TS(X)$ denotes the write time-stamp of data item X.

Timestamp

(a)

100 T_1	200 T_2
	R(A)
W(A)	

T_1 Timestamp - 100, T_2 timestamp - 200

Here T_2 transaction wants to read (A) First. Later T_1 transaction wants to write (A) which is older transaction

Ex A reads 10 in T_2 transaction and Then T_1 transaction updates A value to 20. But T_1 transaction is the older transaction. Here we reject the operation because the lastest transaction t_2 ~~read~~ is reading the old value A=10

(2)

100 T_1	T_2
	W(A)
R(A)	

Timestamp T_1 = 100
" T_2 - 200

let ~~A be 10~~ A=10 (2)

suppose T_2 has updated the value to A=20 and then T_1 transaction has read A=20 and completed the transaction. If suppose T_2 transaction due to server failure, It has roll back the transaction then the original value of A is 10 is retained But T_1 has transaction has read A=20. This is Dirty read problem. we have to ~~roo~~ roll back T_1 transaction

(3)

T_1	T_2
	W(A)
W(A)	

A=10

Timestamp $T_1 = 100$
" $T_2 = 200$

Here T_2 transaction is first updating the value of A=20 and Then T_1 the ~~the~~ is updating the value from A=20 to A=30 so this is lost update problem.

Recoverability

Sometimes a

So we have to roll back transaction to to prevent lost update.

Recoverability

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.

Table 1

T_1	T_1's buffer space	T_2	T_2's buffer space	Database
				A = 6500
Read(A)	A = 6500			A=6500
A = A - 500;	A = 6000			A = 6500
write A	A = 6000			6000
		Read(A)	A = 6000	A = 6000
		A = A+1000	A = 7000	A = 6000
		write(A)	A = 7000	A = 7000
		commit;		
Failure point				
commit				

The above table shows a schedule which has two transactions T_1 reads and writes the value of A and that value is read and written by T_2. T_2 commits but later on, T_1 fails. Due to the failure, we have to rollback T_1. T_2 should also be rollback because it reads the value written by T_1 but T_2 can't be rollback because it is already committed. So this type of schedule is known as irrecoverable schedule.

Irrecoverable Schedule: The schedule will be irrecoverable If T_j reads the updated value of T_i and T_j committed before T_i commit.

T_1	T_1's buffer space	T_2	T_2's buffer space	Database
				A=6500
				A=6500
Read(A)	A=6500			A=6500
A=A-500	A=6000			A=6000
write(A)	A=6000			
		Read(A)	A=6000	A=6000
		A=A+1000	A=7000	A=6000
		write(A)	A=7000	A=7000
Failure point				
Commit;				
		commit;		

The above table 2 shows a schedule with two transactions. Transaction T_1 reads and writes A, and that value is read and written by transaction T_2. But later on, T_1 fails. Due to this we have to rollback T_1. T_2 should be rollback because T2 has read the value written by T_1. As it has not committed before T_1 commits. So we can rollback transaction

T_2 as well. so it is recoverable with cascade rollback.

Recoverable with cascading roll back:-

The schedule will be recoverable with cascading rollback if T_j reads the updated value of T_i, commit of T_j is delayed till commit of T_i.

T_1	T_1's buffer space	T_2	T_2's buffer space	Database
				A=6500
Read(A)	A=6500			A=6500
A=A-500	A=6000			A=6500
write(A)	A=6000			A=6000
commit;		Read(A)	A=6000	A=6000
		A=A+1000	A=7000	A=6000
		write(A)	A=7000	A=7000
		commit;		

The above Table 3 shows a schedule with two transactions. Transaction T_1 reads and write A and commits, and that value is read and written by T_2. So this is a cascade less recoverable schedule.

Crash recovery

Crash recovery is the process by which the database is moved back to a consistent and ustable state. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred.

Failure classification

To see where the problem has accured, we generalize a failure into various categories as follows.

(a) Transaction failure! - There are two types types of errors that may cause a transaction to fail

(i) Logical error! - The

transaction can no longer continue with its normal execution because of some internal condition, such as bad Input, data not found, overflow or resource limit exceeded.

(ii) ~~system~~ error :- when the database system itself terminates an active transaction because the DBMS is not able to execute it.

(3) Disk failures :- A disk block loses it content as a result of either a head crash or failure during a data-transfer operation.

Recovery and Atomicity

when a system crashes, it may have several transactions being executed and various files opened for them to modify the data items.

Transactions are made of various operations which are atomic in nature. But acc to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is either all the operations are executed or none.

when DBMS recovers from a crash. It should maintain the following

→ It should check the states of all the transactions, which are being executed.

→ A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.

→ It should check whether the transaction can be completed now or it needs to be rolled back.

→ No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques which can help a DBMS in recovering as well as maintaining the atomicity of a transaction.

Maintaining the logs :- maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.

Maintaining shadow paging :- where the changes are done on a volatile memory and later, the actual Database is updated.

a) Log-based recovery

Log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the ~~value~~ logs are written prior to the actual modification and stored on a stable storage media, which is fail safe.

-> The log file is kept on a stable storage media

-> when a transaction enters the system and starts execution, it writes a log about it.

< T_n, start >

-> when the transaction modifies an item x, it writes log as follows

< T_n, x, v_1, v_2 >

It reads T_n has changed the value of x from v_1 to v_2.

-> when the transaction finishes, it logs < T_n, commit >

The Database can be modification :-

using two approaches.

(a) Deffered database Modification

All logs are written on to the stable storage and the database is updated when a transactions commits

let T_0 be a transaction that transfers 50 accountA to account B

$A = 1000$, $B = 2000$

$C = 700$

T_0 read (A);

A := A - 50

write (A);

read (B);

B := B + 50;

write (B)

let T_1 transaction that withdraws 100 Rs from account C

T_1

read (C)

C = C - 100

write (C).

Portion of the database log corresponding to T_0 and T_1

	Database (after committing)
$\langle T_0 \text{ start} \rangle$	
$\langle T_0, A, 950 \rangle$	$A = 950$
$\langle T_0, B, 2050 \rangle$	$B = 2050$
$\langle T_0 \text{ commit} \rangle$	
$\langle T_1 \text{ start} \rangle$	
$\langle T_1, C, 600 \rangle$	
$\langle T_1 \text{ commit} \rangle$	$C = 600$

Immediate Database modification

Each log follows an actual database modification. That is, the database is modified immediately after every operation.

Log

<T_0 start>

<T_0, A, 1000, 950>

<T_0, B, 2000, 2050>

<T_0 commit>

<T_1 start>

<T_1, C, 700, 600>

<T_1 commit>;

Database

A = 950

B = 2050

C = 600

Check point

Keeping and maintaining logs in real time and in real environment may fill out ~~...~~ all the memory space available in the system. As time passes, the log file may grow too big to be handled at all.

Check point is a mechanism where all the previous logs are removed from the system. and stored permanently in a storage disk.

Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions ~~...~~ were committed.

Recovery:- when a system with concurrent transactions crashes and recovers, It behaves in the following manner.

→ The recovery ~~stem~~ system reads the logs backwards from the end to the last checkpoint.

→ It maintains two lists, an undo-list and a redo-list

→ If the recovery ~~stem~~ system sees a log with $\langle T_n, start \rangle$, and $\langle T_n, commit \rangle$ or just $\langle T_n, commit \rangle$ it puts the transaction in redo-list.

→ If the recovery systems sees a log with $\langle T_n, start \rangle$ but no commit or abort log found, it puts the transaction in undo list.

All the transactions in the undo-list are then ~~d~~ undone and their logs are removed. All the transactions in the redolist and their previous logs are removed and then redone before saving their logs.