28/2/20 Unit-4.

4. Transaction Management

Introduction to Transaction:

Collection of operations that perform a single logical unit of work are called transactions

→ A transaction is a unit of program execution that access and possibly update various data items.

→ Transaction is initiated by a user program written in a high level data manipulation language (or) programming language [SQL, C++ (or) Java] where it is delimited by stmts of the form begin-transaction and End-transaction.

→ The transaction consists of all operations executed between the begin and end transactions.

→ The transaction Management consists of 4 properties. these properties are often called as ACID properties.

1) Atomicity: Either all operations of the transaction are reflected property in the database (or) None

2) consistency: execution of the transactions in isolation preserves the consistency of database

3) Isolation: Even though multiple transaction may execute concurrently, the system guarenty is that each transaction is unaware of other transaction executing concurrently in the system

Scanned with CamScanner

successfully the changes it has made to the database persist even if there are system failures.

→ Transaction access the data using two operations:

i) read(x):- which transfers the data item x from database to a local buffer belonging to the transaction that executed the read operation.

ii) write(x): which transfers the data item x from the local buffer of the transaction that executed the write back to the database

Example:

let T_i be the transaction that transfers 50 rupees from Account A to account B. This transaction can be defined as

T_1 =

```
    read(A);
A := A - 500;
    write(A);
    read(B);
B := B + 50;
    write(B)
```

i) consistency:-

The consistency requirement here is that the sum of A & B be unchanged by the execution of the transaction.

Scanned with CamScanner

→ ensuring consistency for an individual transaction is the responsibility of the application programmer who codes the transaction.

2) Atomicity:

→ If the atomicity property is present all actions for the transactions are reflected in the database (or) none.

→ The Basic idea behind ensuring atomicity is the database system keeps track of the old values of any data on which a transaction perform a write, & if the transaction does not complete its execution, the database system restores the old values to make it appear as through the transaction never executed.

→ Ensuring atomicity is the responsibility of the database system itself and it is handled by the component called Transaction Management component.

Durability:-

The durability property guarantees that once a transaction completes successfully all the updates i.e, carried out on the database persist even if there is a system failure after the transaction completes execution.

→ we can guarantee durability by ensuring that either

Scanned with CamScanner

1) the updates carried out by the transaction have been written to the disk before the transaction complete.

2) Information about the updates carried out by the transaction and written to disk is sufficient to enable the database to reconstruct the updates when the database system is restarted after the failure.

→ Ensuring durability is the responsibility of a software component of the database system called the Recovery management component.

Isolation:

The Isolation property of a transaction ensures that the concurrent execution of transaction results in a system state i.e, equivalent to a state that could have been obtained had these transactions executed one at a time in some order.

→ Ensuring the isolation property is the responsibility of a component of the database system called concurrency control component.

Transaction states:

A transaction must be in one of the following states active. The initial state the transaction stays in this state while

Scanned with CamScanner

It is executing.

partially committed: After the final stmt has been executed.

failed: After the discovery that normal execution can go longer proceed.

Aborted: After the transaction have been rolled back and database has been restored, to its state previor to the start of the transaction.

· Committed :-

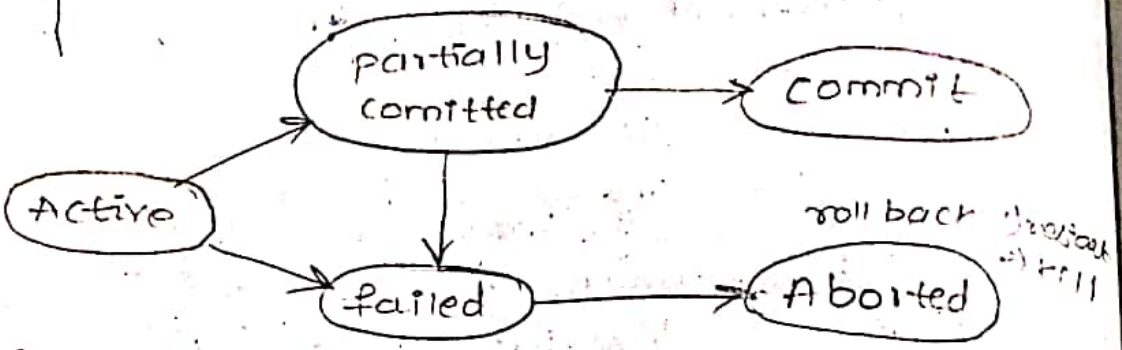
After successful completion at absorpted state the system has 2 option

1) It can restart the transaction but only if the transaction was aborted has a result of some hardware (or) software error that was not created to the internal logic of the transaction. A restarted transaction is consider to be a new transaction.

2) It can kill the transaction it usually thus so because of some internal logic error that can be connected only by rewritting the application program or because the input was bad (a) because the desired data were not found in database

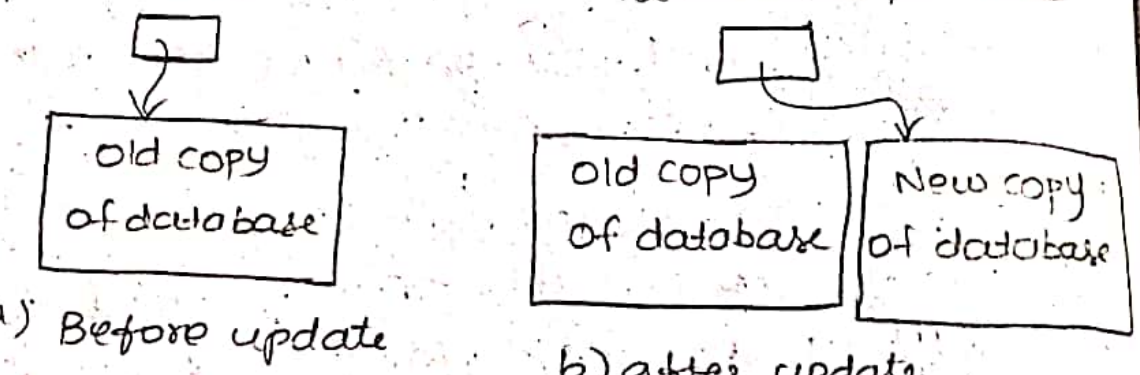
Scanned with CamScanner

State diagram of a transaction



29/2/20.

Implementation of Atomicity & Durability:



a) Before update

b) after update

fig :- shadow-copy technique of Atomicity & durability

→ The Recovery management component of db system can support atomicity & durability of a variety of schemes. A scheme called as the shadow-copy scheme. It is based on making copies of the database called shadow-copies.

→ A pointer called database pointer is maintain on disk it points to the current copy of the db

→ The shado
wants to up
copy of db
dp copy.
→ If at any
aborted the
the old cop
affected.
→ If the
committed
1) The OS [
sure tha
the db ha
2) The data
pointer t
The New
copy of
then de
have be
the up
Transac
If tran
point
the db
transa
copy o
syster
→ If s
Update

Scanned with CamScanner

→ The shadow-copy scheme a transaction that wants to update the db. first creates a complete copy of db all updates are done on the new db copy.

→ If at any point the transaction has to be aborted the system deletes the new copy. The old copy of the db has not been affected.

→ If the transaction completes it is committed as follows.

1) The OS [operating system] is asked to make sure that all the pages of the new copy of the db have been return out to disk.

2) The database system updates the db pointer to point to the new copy of the db. The New copy then becomes the current copy of the db. The old copy of the db is then deleted. The transaction is said to have been committed at the point where the updated db pointer in return to disk.

Transaction failure:

If transaction fail at any time before db pointer is updates, the old content of the db are not effected, about the transaction by just deleting the new copy of db.

System failure:

1) → If system fails at any time before the updated db pointer is return to disk.

Scanned with CamScanner

When the system restarts it will read db pointer from the original contents of the db and none of the effects of the transaction will be visible on the db.

2) If the system fails after the db pointer has been updated on disk once new copy is updated on the disk its contents will not be damaged even if there is a system failure.

Thus the atomicity & durability properties of transactions are ensured by the shadow_copy implementation of the recovery management component.

Concurrent executions :-

The two good reasons for allowing concurrency

1) Improved through put and resource utilization
2) Reduced waiting time

The database system must control the interaction among the concurrent transactions to prevent them from destroying the consistency of the db i.e; through a variety of mechanisms concurrency control scheme

Ex:- let T_1 & T_2 be two transactions that transfers funds from 1 account to another account.

Scanned with CamScanner

Transaction T_1 transfers 50 rupees from account A to account B

Suppose the current values of accounts A & B are 1000 & 2000 rupees respectively

```
T1: read(A);                    T2: read(A)  // 950
A: A - 50;                      temp = A * 0.1; // 95
write(A); // 950                A = A - temp; // 855
read(B);                        write(A); A: 855
B: B + 50;                      read(B); // 2050
write(B); // 2050               B: B + temp; // 2145
                                write(B); // 2145
```

Serial Schedule :-

A serial schedule T_2 is followed by T_1

T_1	T_2

T_1 : read(A);
A : A - 50;
write(A);
read(B))
B : B + 50;
write(B);
	T_2 : read(A)
	temp = A * 0.1;
	A : A - temp; // 855
	write(A);
	read(B);
	B : B + temp)
	write(B)); // 2145

Scanned with CamScanner

Schedule-2 A serial schedule in which T_1 followed by T_2

T_1	T_2

	T_2: read (A) // 1000
	temp: A * 0.1; // 100
	A : A − temp ; // 900
	write (A); // 900
	read (B); // 2000
	B: B + temp ; // 2100
	write (B) ; // 2100

T_1 : read (A); // 900
A: A − 50; // 850
write (A); // 850
read (B); // 2100
B: B + 50; // 2150
write (B);	

Schedule - 3 :- A concurrent schedule equivalent to schedule-1

T_1	T_2

Read (A) // 1000
A: = A − 50; // 950
write (A); // 950	

	Read (A) // 950
	temp: = A * 0.1 // 95
	A: A − temp // 855
	write (A) // 855

Read (B) B: = B + 50 write (B)	

	Read (B) // 2050
	B: = B + temp // 2145
	Write (B) // 2145

Scanned with CamScanner

Schedule 4: A concurrent schedule (Inconsistent)

T_1	T_2
Read(A) // 1000	
A := A - 50 // 950	
	Read(A) // 1000
	temp := A * 0.1; // 100
	A := A - temp // 900
	write(A) // 900
write(A) // 950	
Read(B) // 2000	
B := B + 50 // 2050	
	Read(B) // 2000
	B := B + temp // 2100
	write(B) // 2100

Schedule - 5: A concurrent schedule (Inconsistent)

T_1	T_2
Read(A)	
A := A - 50	
	Read(A) // 1000
	temp := A * 0.1 // 100
	A := A - temp // 900
	write(A) // 900
	Read(B) // 2000
write(A) // 900	
Read(B) // 2000	
B := B + 50 // 2050	
write(B) // 2050	
	2000 + 100
	B := B + temp // 2100
	write(B) // 2100

Scanned with CamScanner

system carries out this task.
concurrent schedule has to be always equivalent to a serial schedule.

** Serializability:

The database system must control concurrent execution of transactions to ensure that the database state remains consistent.

2 types of serializability.

1) conflict serializability
2) view serializability.

1) conflict serializability:

T_1	T_2
Read(A)	
write(A)	
	Read(A)
	write(A)
Read(B)	
write(B)	
	Read(B)
	write(B)

Let us consider a schedule A in which there are 2 consecutive Instructions I_i and I_j of transactions T_i & T_j where $i \neq j$.

Scanned with CamScanner

→ If I_i & I_j referred to different data items then we can swap I_i & I_j without effecting the results.

→ But I_i & I_j referred to the same data item (Q) then the order of the 2 steps may matter

4 cases to consider.

case-1: I_i = Read(Q), I_j = Read(Q)

The order of I_i and I_j doesnot matter

case-2: I_i = Read(Q), I_j = write(Q)

If I_i comes before I_j then T_i does not read the value of Q. i.e, written by T_j in instruction I_j. If I_j comes before I_i then T_i reads the values of Q i.e written by T_j. thus the order of I_i & I_j matters

case-3: If I_i = write(Q) & I_j = Read(Q)

The order I_i & I_j matters for reason similar to that of previous case.

case-4: If I_i = write(Q), I_j = write(Q)

The order of these instructions does not effect either T_i or T_j but it will effect the db which resides in disk so we said that I_i and I_j conflict. If they are operations by different transactions on the data item.

Scanned with CamScanner

9/3/20

i)

EOC:-

T_1	T_2
R(A)	
W(A)	
	R(B)
	W(B)
R(B)	
W(B)	
	R(B)
	W(B)

ii)

T_1	T_2
R(A)	
W(A)	
	R(A)
R(B)	
	W(A)
W(B)	
	R(B)
	W(B)

iii)

T_1	T_2
R(A)	
W(A)	
	R(A)
R(B)	
W(B)	
	W(A)
	R(B)
	W(B)

iv)

T_1	T_2
R(A)	
W(A)	
R(B)	
	R(A)
W(B)	
	W(A)
	R(B)
	W(B)

v)

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

the c
to
schedu
is con
thus
since
seria
conf
view
consi
same
bot
are
the
i) f
T_i
Sch
Gr
co
2)
ex
If-
acy
t
c

Scanned with CamScanner

The concept of conflict equivalence leads to the concept of conflict serializability schedule S is conflict serializable if it is conflict equivalent to a serial schedule thus schedule-3 is equivalent serializable since it is conflict equivalent to the serial schedule-1 that is called as conflict serializability.

View serializability:

consider two schedules S & S' where the same set of transactions participates in both schedules. the schedules S & S' are said to be view equivalent if three conditions are met.

1) for each data item 'Q' if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule S', also read the initial value of Q.

2) For each data item Q, if transaction T_i executes read Q in schedule S, and if that value was produced by a ~~write~~ write Q operation executed by transaction T_j, then the read Q operation of transaction T_i must in

Scanned with CamScanner

schedule S', also read the value of Q that was produced by the write Q operation of transaction Tj.

3) For each data item Q, the transaction that performs the final write Q operation in schedule S, must perform the final write Q operation in schedule S'.

→ In previous example schedule-1 is not view equivalent to schedule-2 but schedule-1 is view equivalent to schedule-3.

→ Schedule-S is said to be view serializable if it is view equivalent to serial schedule.

→ every confict serializable schedule is also view serializable but there are view serializable schedules that are not conflict serializable.

Blind writes:- In any transaction if they perform write Q operation without having read Q operation, writes of this sort are called Blind writes. It occurs only in view-serializable schedule

Implementation of Isolation:

In Isolation property schedule must have to leave the database in consistent state & allow transaction failure to be handled in a save manner.

→ To control the concurrency scheme a transaction acquires a lock on entire db before it starts and release the lock after it committed.

→ No other transactions are allowed to accept the db until the lock db has been released.

→ This locking policy executes only one transaction at a time which means only serial schedules are generated.

- the goal of concurrency control scheme is to provide a high degree of concurrency which ensuring that all schedules that can be generated are conflict or view serializable and cascadeless.

Scanned with CamScanner

4/3/20

Recoverability:

Schedules are acceptable from the view point of consistency of the db if there are no transaction

If T_i fails we need to undo the effects of transaction T_i to ensure the atomicity property. If a system allows concurrent execution it is necessary to ensure any Transaction T_j depended on T_i where T_j read data written by T_i is also aborted.

1) Recoverable schedule:

T_8	T_9
R(A)	
W(A)	
commit	
	R(A)
	commit
R(B)	

fig:- schedule

→ If T_9 commit immediately after (executing) reading (A) instruction before T_8 commits. If T_8 fails then we have to abort T_9 to ensure atomicity but T_9 has already comitted. where it

es imp
failu
are tan
→ NO N
→ A rec
each pa
that
writte
T_i app
2) Casc
Tic
R(A
R(B
W(A
→ each
in thi
transa
tran
recov
failu
roll

Scanned with CamScanner

is impossible to recover correctly from the failure of T_8. These type of schedules are known as Non-recoverable schedule.

→ NO Non-recoverable is allow in the db

→ A recoverable schedule is one where for each pair of transaction T_i & T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the commit operation of T_j

2) Cascadeless Schedule:

T_{10}	T_{11}	T_{12}
R(A)		
R(B)		
W(A)		
	R(A)	
	W(A)	R(A)

→ each and every transaction is dependent in this schedule. to recover correctly from transaction T_i we have to roll back several transaction then only the schedule is recoverable in which a single transaction failure leads to a series of transaction roll backs is called cascading roll back.

Scanned with CamScanner

→ To ignore cascading roll back occurs in the database cascadeless schedules are taken place.

→ A cascadeless schedule is one where, for each pair of Transaction T_i & T_j such that T_j reads a data item written by T_i then the commit operation of T_i must appears before read operation of T_j.

→ So every cascadeless schedule is also recoverable.

part-II

concurrency control:

When several transaction execute concurrently in the db the isolation property may no longer be preserved so interaction is necessary b/w the transaction this can be achieved by : concurrency control scheme

** Lock based protocols:

To ensure serializability a transaction

i) Shared lock [S]: If a transaction T_i has obtained a shared mode lock S on data item Q, then T_i can read, but cannot write.

ii) Exclusive lock [X]: If a transaction T_i has obtained a exclusive lock X on data item Q, then T_i can read & write

lock	S	X
S	T	F
X	F	F

lock compatability matrix component [compatability function]

Ex:-

T_1 :- Transfer 50 Rs from B to A.

T_2 :-> Read A & B value & display the sum of A & B values.

T_1	T_2
lock_X(B)	lock_S(A)
Read(B)	Read(A)
B := B-50	unlock(A)
write(B)	lock_S(B)
unlock(B)	R(B)
lock_X(A)	unlock(B)
R(A) A := A+50 W(A)	display (A+B)
unlock(A)	

Scanned with CamScanner

T_1	T_2
lock_X(B)	
R(B)	
B := B-50	
W(B)	
unlock_(B)	
	lock_S(B)
	R(B)
	unlock_(B)
lock_X(A)	
R(A)	
A := A+50	
W(A)	
unlock_(A)	
	lock_S(A)
	R(A)
	unlock_(S) → display(A+B)

→ Every transaction require a lock in an appropriate mode on data item(Q) the requirement is made to the concurrency control manager [CCM].
The transaction can proceed operation only after the CCM grants the lock to the transaction.

→ To access a dataitem transaction T_i must first lock that item. If the data item is already locked by another transaction in an incompatable mode, CCM will not grant the lock until all incompatable locks held by other transactions have been released. So T_i has to wait.

Scanned with CamScanner

T1 T2

example:

T_1	T_2	CCM
lock-X(B)		→ G-X(B, T_1)
Read(B)		
B := B-50		
W(B)		
Unlock(B)		
	lock-S(A)	→ G-S(A, T2)
	Read(A)	
	Unlock(A)	
	LOCK-S(A)	
		→ G-S(A, T2)
	Read(B)	
	Unlock(B)	
	Display(A+B)	
lock-X(A)		
Read(A)		G-X(A, T_1)
A : A+50		
write(A)		
Unlock(A)		

T_1	T_2	CCM
LOCK-X(B)		→ G-X(B, T_1)
Read(B)		
B := B-50		
W(B)		
Unlock(B)	LOCK-S(B)	→ G-S(B, T2)
	Read(B)	
	Unlock(B)	
LOCK(X(A)		→ G-X(A, T_1)
Read(A)		
A: A+50		
write(A)		
unlock(A)		
	LOCK-S(A)	→ G-S(A, T_1)
	Read(A)	
	Unlock(A)	
	display(A+B)	

Scanned with CamScanner

10.03.20

Deadlock:

Sometimes locking can be lead to an undesirable situation i.e, which leads to deadlock where T_1 is waiting for T_2 to release a lock on A and also T_2 is waiting for T_2 to release a lock on B where the Transaction is stopped i.e, waiting for each other request is called as deadlock.

T_1	T_2
L-X(B)	
R(B)	
B_i=B-30	
W(B)	
	L-S(A)
	R(A)
	L-S(B) X
X L-X(A)	

Fig: Deadlock Schedule.

To Overcome deadlocks the system must rollback one of the two Transaction.

Rollback means release locks from transactions so that other transactions can execute there operations.

→ If we don't use locking (or) if we unlock data item's as soon as possible after reading (or) writing then the result is inconsistent data.

→ If we don't unlock a data item before request a lock on another data item deadlocks are going to occur.

→ To Handle the deadlocks we are rolling back Transactions but the inconsistent states are not handled by the database system.

for all situations each transaction in system must follow a set of rules called locking protocal which indicates when to lock and unlock on data items

② Granting of locks:

When a transaction request a lock on a dataitem in a particular mode and no other transaction has a lock on the same data item in a conflicting mode then the lock will be granted.

If T_2 has a shared lock on Q, T_1 requests an exclusive mode lock on Q then T_1 has to wait until T_2 releases the lock on Q. If in the same time if T_3 requests a shared mode lock on Q then the request is compatible. so, the lock is granted to T_3 by T_2 instead of T_1 if the same procedures continues the T_1 never gets the exclusive lock mode Q. so, T_1 is never Executed. This process is known as starvation.

Avoid starvation of transaction by granting locks in the following manner:

When a transaction T_i request a lock on data-item Q in a particular mode M the CCM grants the lock as

i) There is no other transaction holding a lock Q in a mode that conflicts with M.

ii) There is no other transaction that is waiting for a lock on Q and that made its lock request before T_i.

iii) While performing the above manner a lock request never get blocked by a lock request that is made lock.

⑧ Two-phase locking protocal:

One protocal that ensures serialisability is the two-phase locking protocal.

Each transaction issue lock and unlock request in two phases.

i) Growing phase:

A Transaction may obtain locks, but may not release any lock is called a growing phase.

ii) Shrinking phase:

A Transaction may release locks but may not obtain any locks is called as shrinking phase.

iii) locking point (lock point):

The point in the schedule where the transaction has obtained its final lock i.e, end of growing phase.

→ The transaction has to be ordered according to their locking points.

→ To be serializable the schedules have to be cascadeless but cascading rollback can occur under two-phase locking protocal.

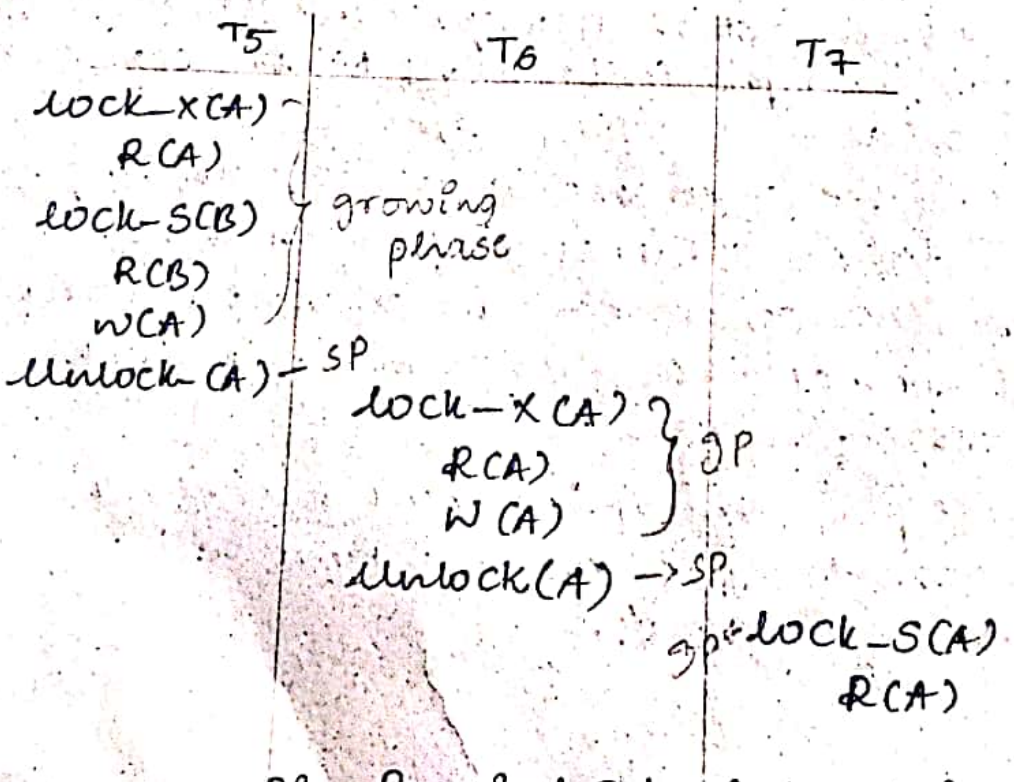


Fig. Partial Schedule using 2phase locking.

Scanned with CamScanner

If T5 fails after T6 reads data item(A) then the cascading rollback of T6 & T7 occurs. Cascading rollbacks can be avoided by a modification of two phase locking protocal called the Strict two phase locking protocal.

This strict two-phase locking protocal requires not only that locking be two phase but also the all exclusive mode locks taken by a transaction be held until the txn. commits.

→ Another variant of 2phase locking is the rigorous two phase locking protocal which requires that all locks be held until the transaction commits.

→ Most database system implements either strict (or) rigorous 2-phase locking.

⇒ Refinement of two phase locking protocal with the lock Conversion:

lock conversions are of two types

i) Upgrade
ii) Downgrade

i) Upgrade: A shared lock to an exclusive lock.
ii) Downgrade: An exclusive lock to a shared lock.

Upgrading can take place in only the growing phase, whereas downgrading can take place in only the shrinking phase.

A scheme automatically generates the appropriate lock and unlock instruction for a transaction, on the basis of read and write request from the transaction.

→ When a transaction T_i issues a read. Operation on Q, the system issues a lock-S(Q) followed by the read(Q) instruction.

→ When T_i issues a write(Q) operation, the system checks to see whether T_i already.

holds a shared lock on a, if it does, then the system issues an upgrade(a) instruction followed by the write(a) instruction. Otherwise the system issues a lock-x(a) instruction followed by the write(a) instruction.

→ All locks obtained by a transaction are unlocked after the transactions commits (or) aborts. (roll back).

④ Implementation of locking:-

→ A lock manager can be implemented as a process that receives messages from transaction and send messages in reply.

→ lock Manager uses data structure i.e linked list of records to maintain the request send by transaction.

→ Maintaining each and every request and order of a transaction with its name along with data item is a hash table called lock table.

→ This algorithm gusantees that there is no starvation and ~~gives~~ uses shared memory instead of message passing for lock request (or) grant.

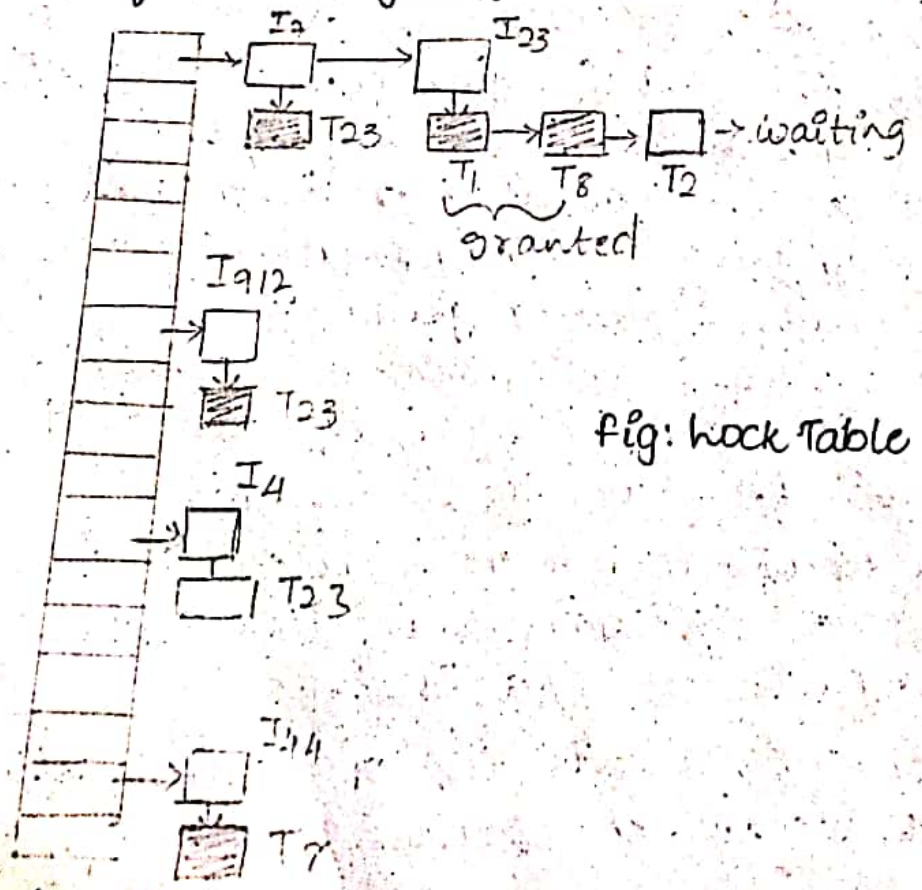


fig: lock Table

The lock
1. when
a record
data ite
it creat
to the
if the
item
grant
reques
request
2. when
mess
record
3. If a t
delet
tran
itake
tra
the

⑤ Gra
To
2 pha
→ The p
may
grapl
→ In
allo
can
clos
i) the
ii) Suk
by
do

Scanned with CamScanner

The lock manager processes requests this way:
1. When a lock request message arrives, it adds a record to the end of the linked list for the data item, if the linked list is present otherwise it creates a new linked list. It always grants the first lock request on a data item but if the transaction request a lock on a data item on which a lock has already been granted, the lock manager grants the request only if it is compatible with earlier requests otherwise the request has to wait.
2. When the lock manager receives an unlock messages from a transaction, it deletes the record for that data item in the linked list.
3. If a transaction aborts, the lock manager deletes any waiting requests made by the transaction. Once the database system has taken appropriate actions to undo the transactions, it releases all locks held by the aborted transaction.

(5) Graph Based Protocali-

To construct locking protocals that are not 2 phase but ensure conflict serializability.

→ The partial ordering implies that the set 'D' may not be viewed as a directed acyclic graph called a database graph.

→ In the tree protocal, the only lock instruction allowed is lock exclusive each Transaction T_i can lock a data item atmost once, and must observe the following rules.

i) The first lock by T_i may be on any data item.
ii) Subsequently a data item Q can be locked by T_i only if the parent of Q is currently locked by T_i.

Scanned with CamScanner

(iii) Data items may be unlocked at any time.
(iv) A Data item that has been locked and unlocked by T_i cannot subsequently be relocked by T_i.

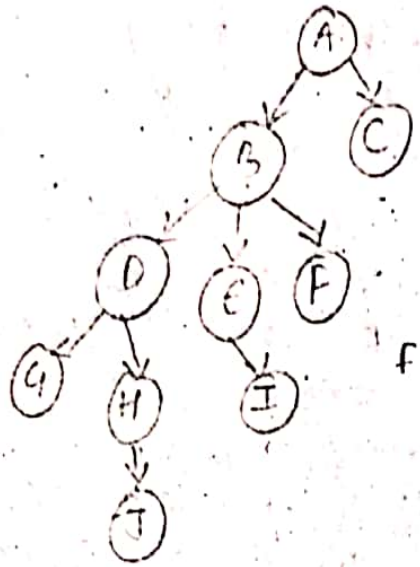


Fig: Tree Structured database graph.

→ This tree protocal doesnot ensure recoverability and cascadeless.

→ TO ensure, the protocal can be modified not permit release of Exclusive lock until the end the Transaction i.e; reduces Concurrency.

** Timestamp-Based Protocols:

In locking protocols the order b/w every pair of conflicting transaction is determined at execution time. By the first lock that they both request (i.e, incomptable modes).

An Ordering among transactions can be done in advance by using a common method that is timestamp-ordering scheme.

→ Timestamp (TS):

The TS for each transaction T_i is assigned by the database system before the transaction T_i starts execution.

If T_j enters after TS assigned for T_i then $TS(T_i) < TS(T_j)$.

* Two Methods for Implementing this Scheme.
i) Use the value of the system clock as the time stamp.
→ TS(Ti) = value of the clock when the txn. enters the system.
ii) Use a logical counter.
→ Incremented after a new time stamp has been assigned i.e, TS(Ti) = value of the counter.

13/3/20
→ To implement this scheme, associated with each data item two timestamp values.
i) Write Time stamp (w-Timestamp)(Q): denotes the largest time stamp of any transaction stamp that executed write(Q) successfully.
ii) R_Timestamp(Q): that denotes the largest time stamp of any transaction that executed Read(Q) successfully.

→ The timestamps are updated whenever a new Read(Q) or write(Q) instruction is executed.

* The timestamp ordering protocol:
It ensures that any conflicting read & write operations are executed in TS order.
This protocol operates as follows:
1) Suppose that transaction Ti issues read(Q)
a) If TS(Ti) < W_TS(Q) then Ti needs to read a value of Q that was already over written. Hence the read operation is rejected. and Ti is roll back

b) If $TS(T_i) > W\_TS(Q)$ then the read operation is executed and $R\_TS(Q)$ is set to the maximum of $R\_TS(Q)$ & $TS(T_i)$

2) Suppose the transaction T_i issues write(Q)

a) If $TS(T_i) < R\_TS(Q)$ then the value of Q that T_i is producing was needed previously the value never be produced. Hence the system rejects the write operation and rolls T_i back.

b) If $TS(T_i) < R\_TS(Q)$ then T_i is attempting to write an obsolute value of Q. Hense the system rejects its write operation and rolls T_i back.

c) Otherwise, $\left[TS(T_i) \geq R\_TS(Q) \text{ or } TS(T_i) \geq W\_TS(Q) \right]$ then the system executes write operation and set $W\_TS(Q)$ to $TS(T_i)$

Ex:- If there are two transactions T_{14} & T_{15} where T_{14} is used to display the sum of acc A & acc B values where T_{15} is used to transfer 50 Rs from B to A & display sum of A&B

display(A+B);

T15 : R(B)
B:= B-50
W(B)
R(A)
A:=A+50
W(A)
display(A+B)

	R(B)
	R(B)
	B:B-50
	W(B)
R(A)	
	R(A)
display(A+B)	
	A:=A+50
	W(A)
	display(A+B)

fig Schedule-3

Ex:-

T16	T17
R(Q)	
	W(Q)
W(Q)	

fig:- Schedule-4

* Thomas write Rule:

The modification to the time-stamp ordering protocol is called as Thomas write Rule.

1) Suppose the transaction T_i issues Read(Q)

a) If $TS(T_i) < W\_TS(Q)$ then T_i needs to read a value of Q that was already over written. Hence the read operation is rejected and T_i is roll back.

b) If $TS(T_i) > W\_TS(Q)$ then the read operation is executed and $R\_TS(Q)$ is set to the minimum of $R\_TS(Q)$ & $TS(T_i)$

2) Suppose the T_i issues Write(Q)

a) If $TS(T_i) < R\_TS(Q)$ then the value of Q that T_i is producing was needed previously the value never be produced. Hence the system rejects the write operation and rolls T_i back.

b) If $TS(T_i) < W\_TS(Q)$ then T_i is attempting to write an obsolute value of Q hence these write operation is rejected.

c) otherwise $\left(TS(T_i) \geq W\_TS(Q)\right]$ the system executes the write operation and set $W\_TS(Q)$ to $TS(T_i)$

NOTE:- $TS(T_i) \geq R\_TS(Q)$ thomas write rule: ignore write operation.

Validation based protocol:

To minimize the overhead of concurrency control scheme and to gain the knowledge of which transaction leads to conflict & whichnot, A scheme is required to monitor the system.

There are 3 phases.
When the transaction T_i executes by system:

1) Read : The values of various data items are read, and are stored in variables local to T_i, All write operations are performed on temporary local variables. Without update on the actual database

2) Validation phase: It performs a validation test to determine whether the temporary local variables that holds the result of write operations can copy to database or not.

3) Write phase: If T_i succeeds in validation then the actual updates are applied to the database otherwise T_i is roll back

To perform the validation test, details of various phases of transaction T_i has to know.

1) Start(T_i) :- The time when T_i started its execution

2) Validation(T_i) :- The time when T_i finished it-read phase and started its validation phase

3) finish(T_i) :- The time when T_i finished its write phase

Scanned with CamScanner

Let $TS(T_i)$ = Validation(T_i) and $TS(T_i) < TS(T_j)$ where it means T_i appears before T_j.

→ The validation test for transaction T_j requires that ~~if~~ transaction T_i with $TS(T_i) < TS(T_j)$ one of the following two conditions must hold

i) finish(T_i) < start(T_j)

Since T_i completes its execution before T_j started.

→ The serializability order is indeed maintained.

ii) The set of data items written by T_i doesnot intersect with the set of data items read by T_j, and ~~it~~ T_i complete its write phase before T_j start its validation phase.

i.e start(T_j) < finish(T_i) <

Validation(T_j)

This condition ensures that the write(T_i) & T_j donot Overlap.

Scanned with CamScanner

R(B)		
	R(B)	fig:- Schedule produced by using validation
	B:=B-50	
	R(A)	
	A:=A+50	
R(A)		
<validate>		
display(A+B)	<validate>	
	write(B)	
	write(A)	

→ The validation scheme automatically guards against cascading roll back but possibility of starvation.

→ To avoid starvation, that validation phase is called Optimistic concurrency control scheme, since transactions execute optimistically assume to finish execution and validate at end.