Unit

4 Transaction Processing Concepts

4.1 TRANSACTION CONCEPT

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

A transaction is a unit of program execution that accesses and possibly updates various data items.

We can say that the transaction consist of all operations executed between the begin transaction & end transaction.

The transaction have following four properties.

(1) Atomicity : Either all operations of the transactions are reflected properly in the database or none are, *i.e.*, if everything works correctly without any errors, then everything gets committed to the database. If any one part of the transaction fails, the entire transaction gets rolled back.

(2) Consistency : The consistency property implies that if the database was in a consistent state before the start of a transaction, then after the execution of a transaction, the database will also be in consistence state.

(3) Isolation : Even though multiple transactions may execute concurrently, the system guarant 's that, for every pair of transactions Ti & Tj, it appears to Ti that either Tj finished execution before Ti started or Tj started execution after Ti finished.

(4) Durability : Durability ensures that, once a transaction has been committed, that transaction's update do not get losts, even if there is a system failure.

These four properties are called the ACID properties of Transactions.

4.2 TRANSACTION ACCESS DATA

Transaction access data using two operations.

(i) Read (X) : Read (X) operation transfers the data item X from the 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_1 be a transaction that transfers $ 500 from account A to account B.

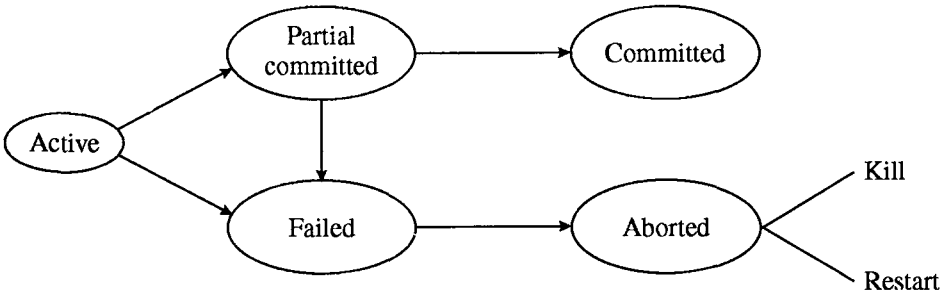
This transaction can be defined as :

```
T1          read (A)
            A : = A−500;
            write(A);
            read (B);
            B : = B+500;
            write (B);
```

4.3 TRANSACTION STATE

A transaction must be in one of the following states :

- **Active :** This state is the initial state, the transaction says in this state while it is executing.
- **Partial Committed :** After the final statement has been executed.
- **Failed :** Failed, after the discovery that normal execution can no longer proceed.



- **Aborted :** Aborted, after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.
- **Committed :** After successful completion.

i.e., A transaction that completes its execution successfully is said to be committed.

- Once a transaction has been committed we cannot undo its effect by aborting it.
- A transaction said to be terminated if it has either committed or aborted.
- A transaction starts in the active state. When it finished its final statements, it enters the partially committed state. At this point, the transaction has completed its execution, but it is still possible that it may have to be aborted.

4.4 CONCURRENT EXECUTIONS

Transaction processing system usually allow multiple transactions to run concurrently allowing multiple transactions to update data concurrently, causes several complications with consistency of the data.

Ensuring consistency in spite of concurrent execution of transactions requires extra work. It is easier to insist that transactions run serially.

i.e., One at a time, each starting only after the previous one has completed.

There are two reasons for allowing concurrency :

(i) Improved throughput & resource utilization.

(ii) Reduced waiting time.

4.4.1 Schedules

- The execution sequences in chronological order are called schedules.
- The schedules are serial; each serial schedule consists of a sequence of instructions from various transactions.

4.5 SERIALIZABILITY

A non serial schedule is said to be serializable, if it is conflict equivalent or view-equivalent to a serial schedule.

e.g.,

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Types of Serializability

- Conflict Serializability
- View Serializability

4.5.1 Conflict Serializability

Let us consider a schedule S in which there are two consecutive instructions Ii & Ij of transaction Ti & Tj respectively ($i \neq j$)

If Ii & Ij refer to different data items, then we can swap Ii & Ij without affecting the results of any instruction in schedule.

If Ii & Ij refer to the same data item Q. Then order may be matter.

There are four cases arise.

Case 1 : If Ii = read (Q), Ij = read (Q)

The order of Ii & Ij does not matter.

Case 2 : If Ii = read (Q), Ij = write (Q)

If Ii comes before Ij then Ti does not read the value of Q that is written by Tj in instruction Ij.

Thus order is matter.

Case 3 : If Ii = write (Q), Ij = read (Q).

The order is matter.

Case 4 : If Ii = write (Q), Ij = write (Q)

The order is not matter.

But for few cases it may be matter.

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

In this schedule; the write (A) of T_1 conflicts with the read (A) of T_2.

While write (A) of T_2 does not conflict with read (B) of T_1.

Because A & B are two different items.

We can swap to non conflicting instructions.

- swap read (B) of T_1 with read (A) of T_2.
- swap write (B) of T_1 with write (A) of T_2.
- swap write (B) of T_1 with read (A) of T_2.

After swapping Above schedule

T_1	T_2
read (A)	
write (A)	
	read (A)
read (B)	
	write (A)
write (B)	
	read (B)
	write (B)

The concept of conflict equivalence lead to the concept of conflict serializability.
We say that a schedule S is conflict serializable if it is conflict equivalent to a serial schedule.
Example : Let a serial schedule S.

T_1	T_2
read (A)	
A : = A – 50	
write (A)	
read (B)	
B : = B + 50	
write (B)	
	read (A)
	temp : = A * 0.1
	A : = A – temp
	write (A)
	read (B)
	B : = B + temp
	write (B)

After the swapping S it becomes schedule S^1 :

T_1	T_2
read (A)	
A : = A – 50	
write (A)	
	read (A)
	temp : = A * 0.1
	A : = A – temp
	write (A)
read (B)	
B : = B + 50	
write (B)	
	read (B)
	B : = B + temp
	write (B)

The schedule S^1 is equivalent to schedule S.
Thus it is the example of conflict serializability.

4.5.2 View Serializability

Consider two schedules S and S^1, where the some set of transactions participates in both schedulers.

The schedule S :

T_1	T_2
read (A)	
A : = A – 50	
write (A)	
read (B)	
B : = B + 50	
write (B)	
	read (A)
	temp : = A * 0.1
	A : = A – temp
	write (A)
	read (B)
	B : = B + temp
	write (B)

The schedules S and S^1 are said to be view equivalent if three conditions are satisfied.

(1) For each data item Q, if transaction Ti reads the initial value of Q in schedule S, then transaction Ti must, in schedule S^1, also read the initial value of Q.

(2) For each data item Q, if transaction Ti executes read (Q) in schedule S, and if that value was produced by a write (Q) operation executed by transaction Tj, then the read (Q) operation of transaction Ti must in schedule S^1, also read the value of Q that was produced by the same 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^1.

• The concept of view equivalence leads to the concept of view serializability.

The schedule S^1 are :

T_1	T_2
read (A)	
A : = A – 50	
write (A)	
	read (A)
	temp : = A * 0.1
	A : = A – temp
	write (A)
read (B)	
B : = B + 50	
write (B)	
	read (B)
	B : = B + temp
	write (B)

Thus schedules S & S^1 are view serializable, because the schedules S & S^1 are view equivalent.

4.5.3 Testing of Serializability

When designing concurrency control schemes, we must show that schedules generated by the scheme are serializable.

Testing for conflict serializability : Consider a schedule S. We construct a directed graph called a "Precedence Graph" from S.

The set of edges of graph holds one of the three conditions.

$$Ti \longrightarrow Tj \text{ (edge)}$$

(i) Ti executes write (Q) before Tj executed read (Q).

(ii) Ti executes read (Q) before Tj executed write (Q).

(iii) Ti executes write (Q) before Tj executed write (Q).

(A) (B)

If an edge Ti $\longrightarrow$ Tj exists in the precedence graph then, in any serial schedule S^1 equivalent to S. Ti must appear before Tj.

Example : In graph (A) The precedence graph for schedule 1, contains the single edge $T_1 \longrightarrow T_2$, since all instructions of T_1 are executed before the instruction of T_2 executed.

Similarly : The graph (B) shows, the precedence graph for schedule 2 with the single edge $T_2 \longrightarrow T_1$.

Since all the instructions of T_2 are executed before the instruction of T_1 is executed.

"If the precedence graph for S has a cycle, then schedule S is not conflict serializable".

"If the graph contains a no cycle, then the schedule S is conflict serializable".

- To test for conflict serializability, we need to construct the precedence graph and to invoke a cycle-detection algorithm.
- Cycle-detection algorithms based on depth-First search.

e.g.,

The precedence graph contains the edge $T_1 \longrightarrow T_2$, because T_1 executed read (A) before T_2 executed write (A).

It also contain the edge $T_2 \longrightarrow T_1$ because T_2 executed read (B) before T_1 executes write (B).

Since precedence graph contains a cycle hence, it is not conflict serializable.

Testing for view serializability : The testing for view serializability is complicated. In fact, it has been shown that the problem of testing for view serializablitiy is itself NP-Complete.

Thus there is no efficient algorithm to test for view serializability.

- Concurrency-control scheme can use sufficient condition for view serializability, But view serializability schedule may not satisfy the sufficient condition.

Note : We can test a given schedule for conflict serializability by constructing a precedence graph for the schedule and by searching the absence of cycle in the graph.

4.6 RECOVERABILITY

If a transaction Ti fails, for whatever reason. We need to undo (roll back) the effect of this transaction to ensure the atomicity property of the transaction.

In a system that allows concurrents execution it is necessary also to ensure that any transaction

Tj that is dependent on Ti (i.e. Tj has read data written by Ti) is also aborted. To achieve this surety we need to place restrictions on the type of schedules permitted in the system.

4.6.1 Recoverable Schedules

A recoverable schedule is one where, for each pair of transactions Ti & Tj such that Tj reads a data item previously written by Ti.

The commit operation of Ti appears before the commit operation of Tj.

T_1	T_2
read (A)	
write (A)	
	read (A)
read (B)	

In the given transaction, T_2 performs only one instruction read (A).

Suppose that the system allows T_2 to commit immediately after executing the read (A) instruction.

Thus T_2 commit before T_1 does. Now suppose that T_1 fails before it commits, we must abort T_2 to ensure transaction atomocity. However, T_2 has already committed and cannot be aborted.

Thus this situation T_1 is impossible to recover correctly from the failure of T_1.

Note : Most database system require that all schedules be recoverable.

4.6.2 Cascadeless Schedules

Even if a schedule is recoverable, to recover correctly from the failure of a transaction Ti.

We may have to roll-back several transactions. Such situations occur if transactions have read data written by Ti.

- The phenomenon, in which a single transaction failure leads to a series of transaction roll-backs, is called "Cascading Rollback".

e.g.,

T_1	T_2	T_3
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)

In this schedule, Transaction T_1 writes a value of A, that is read by transaction T_2.

Transaction T_2 writes a value of A that is read by transaction T_3.

If T_1 fails, T_1 must roll back since T_2 depends on T_1. So T_2 also rollback and since T_3 depends on T_2 so T_3 also rollback.

Thus after failure of Transaction T_1 all transaction T_2 & T_3 also rollback with T_1.

Hence, the given transaction is cascading rollback.

4.7 TRANSACTION RECOVERY

A transaction begins with successful execution of a BEGIN TRANSACTION statements and it ends with successful execution of either a COMMIT or a ROLLBACK statement.

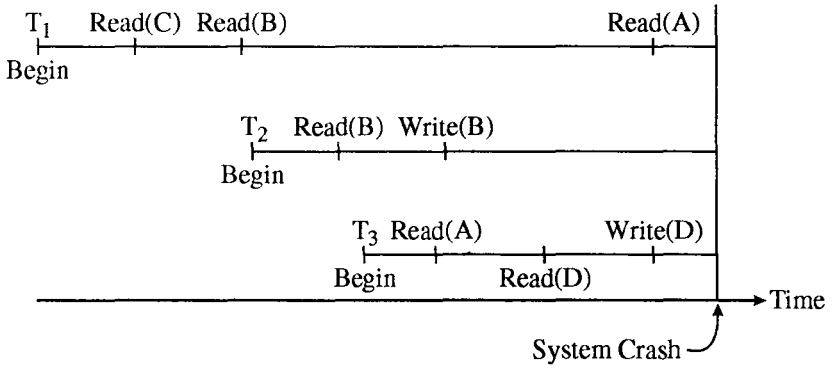
We can now see that transaction are not only the unit of work but also the unit of recovery.

If a transaction successfully commits then the system will guarantee that its updates will be

permanently installed in the database, even if the system crashes the very next moment. It is quite possible, for data that the system might crash after the commit has been honoured but before the updates has been physically written to the database so be the lost at the time of the crash.

Even if that happens, the system's restart procedure will still install those updates in the database.

Thus the restart procedure will recover any transactions that completed successfully but did not manage to get their updates physically written prior to the crash.



System Recovery : The system must be prepared to recover, not only from purely local failure such as the occurrence of an overflow condition within an individual transaction, but also from, "Global Failure" such as a power outage.

4.7.1 Failure Classification

Transaction recovery is the process of restoring transaction to a correct (consistent) state in the event of a failure. The failure may be the result of a system crash due to hardware or software errors, a media failure such as head crash, or a software error in the application such as a logical error in the program that is accessing the transaction.

The numbers of recovery techniques that are based on the atomicity property of transactions. Transaction recovery reverses all the changes that the transaction has made to the database before it was aborted.

Local Failure : A local failure affects only the transaction in which the failure has actually occured.

Global Failure : A Global Failure affects all of the transaction in progress at the time of the failure.

Global Failure fall into two broad categories :

- System Failure
- Media Failure

System Failure : There are various types of failure that may occur in a system.

(i) Transaction Failure : In the transaction failure, there are two types of errors that may occur.

(a) Logical Error : The transaction can no longer continue with its normal execution because of some internal conditions such as wrong input, data not found, resources limit exceeded.

(b) System Error : The system has entered an undesirable state as a result of which a transaction can not continue with its normal execution eg. The deadlock occured in system, starvation in system.

(ii) System Crash : There is a hardware malfunction, or a bug in the database software or the

operating system, that causes the loss of the context of volatile storage, and brings transaction processing to a halt.

Thus the system failures, which affect all transactions currently in progress but do not physically damage the database.

- A system failure is sometime called a soft crash.

Media Failure : Which do cause damage to the database, or some portion of it and affect at least those transactions currently using that portion. A media failure is sometimes called a hard crash.

Disk Failure : Disk Failure is an example of Media Failure.

- A disk block loses its content as a result of either a head crash or failure during a data transfer operation.

4.7.2 Types of Transaction Recovery

In case of any type of failures, a transaction must either be aborted or committed to maintain data integrity. Transaction log plays an important role for database recovery and brings the database in a consistent state in the event of failure. Transaction represent the basic unit of recovery in a database system. The recovery manager guarantees the atomicity and durability properties of transactions in the event of failure. During recovery from failure, the recovery manager ensures that either all the effects of a given transaction are permanently recorded in the database or none of them are recorded. A transaction begins with successful execution of a BEGIN TRANSACTION statement. It ends with successful execution of either a COMMIT or ROLLBACK statement. The following two types of transaction recovery are used :

- Forward recovery
- Backward recovery.

Forward Recovery (REDO)

Forward recovery (also called roll-forward) is the recovery procedure, which is used in case of a physical damage, for example crash of disk pack (secondary storage), failures during writing of data to database buffers, or failure during transferring buffers to secondary storage. The intermediate results of the transactions are written in the database buffers. The database buffers occupy an area in the main memory. From this buffer, the data is transferred to and from secondary storage of the database. The update operation is regarded as permanent only when the buffers are transferred to the secondary storage. The flushing (transferring) operation can be triggered by the COMMIT operation of the transaction or automatically in the event of buffers becoming full. If the failure occurs between writing to the buffers and flushing of buffers to the secondary storage, the recovery manager must determine the status of the transaction that performed the WRITE at the time of failure. If the transaction had already issued its COMMIT, the recovery manager redo (roll forward) so that transaction's updates to the database. This redoing of transaction updates is also known as roll-forward. The forward recovery guarantees the durability property of transaction.

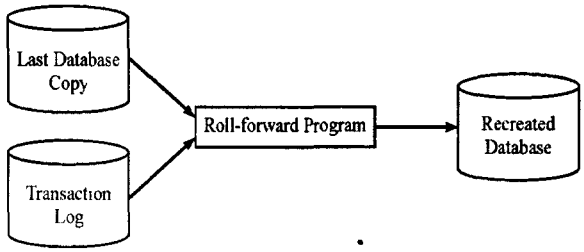


Fig. Forward (roll-forward) recovery or redo

To recreate the lost disk due to the above reasons explained, the systems begin reading the most recent copy of the lost data and the transaction log of the changes to it. A program then starts reading log entries, starting from the first one that was recorded after the copy of the database was made and continuing through to the last one that was recorded just before the disk was destroyed. For each of these log entries, the program changes the data value concerned in the copy of the database to the after value shown in the log entry. This means that whatever processing took place in the transaction that caused the log entry to be made, the net result of the database after that transaction will be stored. Operation for every transaction is performed that caused a change in the database since the copy was taken, in the same order that these transactions were originally executed. This brings the database copy to the up-to-date level of the database that was destroyed.

The figure illustrates an example of forward recovery system. There are a number of variations on the forward recovery method that are used. In one variation, the changes may have been made to the same piece of data since the last database copy was made. In the case, only the last one of those changes at the point that the disk was destroyed need to be used in updating the database copy in the rolled forward operation.

Backward Recovery or UNDO

Backward recovery (also called roll-backward) is the recovery procedure, which is used in case an error occurs in the midst of normal operation on the database. The error could be a human keying in a value, or a program ending abnormally and leaving some of the changes to the database that it was suppose to make. If the transaction had not committed at the time of failure, it will cause inconsistency in the database as because in the interim, other programs may have read the incorrect data and made use of it. Then recovery manager must undo (roll back) any effects of the transaction database. The backward recovery guarantees the atomicity property of transactions.

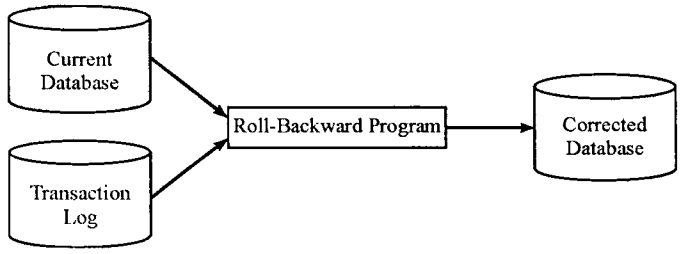


Fig. Backward recovery or UNDO.

Figure illustrates an example of backward recovery method. In case of a backward recovery, the recovery is started with the database in its current state and the transaction log is positioned at the last entry that was made in it. Then a program reads 'backward' through log, resetting each updated data value in the database to it "before image" as recorded in the log until it reaches the point where the error was made. Thus, the program 'UNDOES' each transaction in the reverse order from that in which it was made.

Example, At restart time, the system goes through the following procedure in order to identify all transaction of type $T_2 - T_5$.

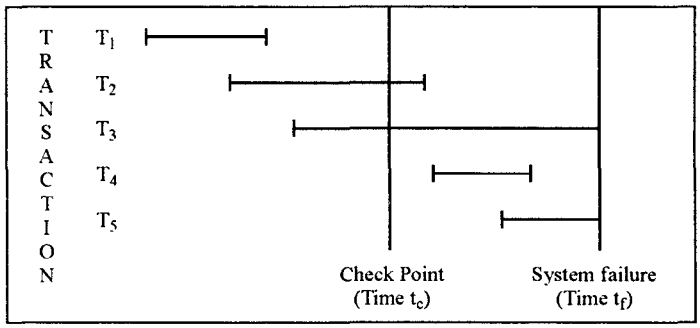


Fig. Five transaction categories

(1) Start with two lists of transactions, the UNDO list and the REDO list. Set the UNDO list equal to the list of all transactions given in the most recent checkpoint given in the most recent checkpoint record; set the REDO list to empty.
(2) Search forward through the log, starting from the check point record.
(3) If a BEGIN TRANSACTION log entry is found for transaction T add T to the UNDO list.
(4) If a COMMIT log entry is found for transaction T, move T from UNDO list to the REDO list.
(5) When the end of the log is reached the UNDO and REDO list identify respectively, transaction of types T_3 and T_5 and transaction of types T_2 and T_4.

The system now works backward through the log, undoing the transactions in the UNDO list. Restoring the database to a consistent state by undoing work is called backward recovery.

4.8 LOG BASED RECOVERY

The very important structure is used for recording database modifications is the log.

The log is a sequence of log records, recording all the update activities in the database.

There are many types of log records. An update log record has these fields.

- **Transaction identifier :** It is the unique identifier of the transaction that performecd the write operation.
- **Data-Item Identifier :** It is the unique identifier of the data written. It is the location on disk of the data item.
- **Old Value :** Old value is the value of the data item prior to the write.
- **New Value :** It is the value that the data item will have after the write.

There are various log records which exists during the transaction processing such as start of the transaction, commit or abort of a transaction.

These log records are :

- <Ti Start> Transaction Ti started.
- <Ti, Xj, V_1, V_2> Transaction Ti has performed a write on data item Xj. Xj has value V_1 before the write and will have value V_2 after the write
- <Ti commit> Transaction Ti has committed.
- <Ti abort> Transaction Ti has aborted.
- log records to be useful for recovery from system and disk failures, the log must reside in stable storage for now, we assume that every log record is written to the end of the log on stable storage as soon as it is created.

The execution of transaction Ti proceeds as follows. Before Ti starts its execution, a record <Ti

start> is written to the log. A write (x) operation by Ti results in the writing of a new record to the log. Finally, when Pi partially commits, a record <Ti commit> is written to the log.

For Example : Consider T_1 is a transaction that transfer Rs. 50 from account A to account B.

```
T1 :    read (A);
        A = A − 50;
        write (A);
        read (B);
        B = B + 50;
        write (B)
```

Let Transaction T_2 withdraws Rs 100 from account C.

```
T2 :    read (C);
        C : = C − 100;
        write (C).
```

Suppose the values of accounts A, B & C before execution took place Rs. 2000, Rs 500 and Rs 1000 respectively.

The log as a result of execution of T_1 & T_2.

<T_1 start>
<T_1, A, 1950>
<T_1, B 550>
<T_1 commit>
<T_2 start>
<T_2, C, 900>
<T_2 commit>

The portion of the database log.

Log	**Database**
<T_1 start>	
<T_1, A, 1950>	
<T_1, B, 550>	
<T_1 commit>	
	A = 1950
	B = 550
<T_2 start>	
<T_2, C, 900>	
<T_2 commit>	
	C = 900

State of the log & database.

4.9 CHECK POINTS

A check point is a point of synchronization between the database and the transaction log file. All buttons are forced written to secondary storage at the check point.

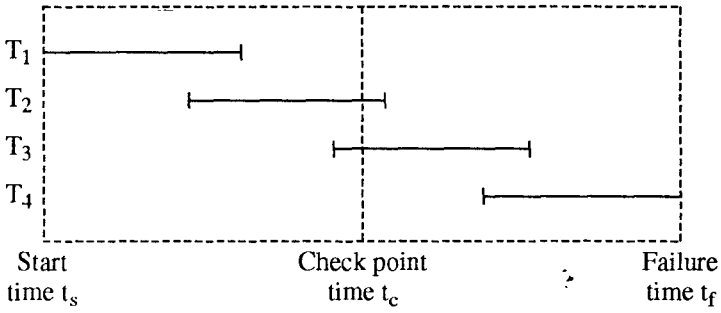
Check points also called syncpoints or save point.

- We used checkpoint to the number of log records that the system must scan when it recovers from a crash.

When a system failure occurs, we must consult the log to determine those transactions that need to be redone & those that need to be undone.

It was necessary to consider only the following transcations during recovery.

(1) Those transaction that started after the most recent checkpoint.
(2) The one transaction, if any, that was active at the time of the most recent check point.
Example :



Let us assume that a transaction log is used with immediate updates. Also, consider that the timeline for transaction T_1, T_2, T_3 & T_4 are shown in Fig. When the system fails at time t_f, the transaction log need only be scanned as far back as the most recent checkpoint etc. Transaction T_1 is okay, unless there has been disk failure that destroyed it and probably other records prior to the last check point. In that case, the database is reloaded from the backup copy that was made at the last check point. In either case, transactions T_2 & T_3 are redone from the transaction log, and transaction T_4 is undone from the transaction log.

4.10 DEADLOCKS

Deadlock : Deadlock is a situation where a process or set of processes are blocked, waiting on an even which will never occur.

We can say, "A state where neither of transactions can every proceed with its normal execution. This situation is called deadlock".

Example :

T_1	T_2
Lock – X (B)	
read (B)	
B : = B – 50	
write (B)	
	Lock – S (A)
	read (A)
	lock – S (B)
Lock – X (A)	

Since T_1 holding an exclusive mode lock on B & T_2 is requesting a shared mode lock on B, T_2 is waiting for T_1 to unlock B.

Similarly, since T_2 is holding a shared-mode lock on A & T_1 is requesting on exclusive mode lock on A, T_1 is waiting for T_2 to unlock. At this transaction is deadlock.

4.10.1 Deadlock Handling

When deadlock occurs, the system must roll back on of the transaction. Once a transaction has been roll back, the data items that were locked by that transaction are unlock.

Ther are two important methods for handling the deadlock.

• Deadlock Prevention. • Deadlock Detection & Recovery.

4.10.1.1 Deadlock Prevention : We can use the deadlock prevention method to ensure that the system will never enter in deadlock state.

There are two basic approaches to deadlock prevention.

(i) One approach ensures that no cyclic waits can occur by ordering the requests for locks, or requesting all locks to be acquired together.

(ii) The other approach is closer to deadlock recovery & performs transaction rollback instead of waiting for a lock.

There are two different deadlock prevention schemes using time stamps.

(1) The Wait-die scheme is a non preemptive technique. When transaction Ti requests a data item currently held by Tj, Ti is allowed to wait only if it has a timestamp smaller than that of Tj.

That is Ti is older than Ti, otherwise Tj is rolled back (dies).

Example : Suppose that transaction T_1, T_2 & T_3 have timestamps 5, 10 & 15 respectively. If T_1 requests a data item held by T_2 then T_1 will wait. If T_3 requests a data item held by T_2, then T_3 will be rolled back (Dies).

(2) The Wound-wait scheme is a preemptive technique. It is a counterpart to the wait-die scheme.

When transaction Ti requests a data item currently held by Tj, Ti is allowed to wait only if it has timestamp larger than that of Tj.

That is Ti is younger than Tj, otherwise Tj is rolled back (Tj is wounded by Ti).

Example : Transactions T_1, T_2 & T_3 have time stamps 5, 10 & 15 respectively. If T_1 requests a data item held by T_2, then the data item will be preempted from T_2 & T_2 will be rolled back.

If T_3 requests a data item held by T_2, then T_3 will wait.

- Whenever the system rolls back transactions, it is important to ensure that there is no starvation.

Note : Both the Wait-die & Wound-wait schemes avoid starvation, at any time, there is a transaction with the smallest timestamps. This transaction cannot be required to roll back in either scheme.

Diff. between Wait-die & Wound-wait schemes :

(i) The Wait-die scheme is a non preemptive technique, while the wound-wait scheme is a preemptive technique.

(ii) In the Wait-die scheme, an older transaction must wait for a younger one to wait, while in the wound-wait scheme, an older transaction never waits for a younger transaction.

(iii) In the wait-die scheme, if a transaction Ti dies and is rolled back, then Ti may reissue the same sequence of requests when it is restarted. If the data item is still held by Tj, then Ti will die again.

Thus Ti may die several times before acquiring the needed data item.

While in Wound-wait scheme transaction Ti is wounded and rolled back because Tj requested a data item that it holds. When Ti is restarted & requests the data item now being held by Tj, Ti waits. Thus, there may be fever roll backs in the wound-wait scheme.

4.10.1.2 Deadlock Detection and Recovery : An other important method for deadlock handling is deadlock detection and Recovery method. Where the system checks if a state of deadlock actually exists.

There are two situations arise in this method :

(i) How we detect the deadlock?

(ii) Then how Recovery from deadlock?

The Deadlock Detection

Deadlock can be described in terms of a directed graph called a "Wait-for Graph". This graph consists of a set of vertices & edges is G = (V, E)

Where V is a set of vertices & E is a set of edges.

"A deadlock exists in a system if and only if the wait for graph contain a cycle".

- Each transaction involved in the cycle is said to be deadlock.
- For detection of deadlocks, the system needs to maintain the wait for graph and periodically to invoke an algorithm that searches for a cycle in the graph.

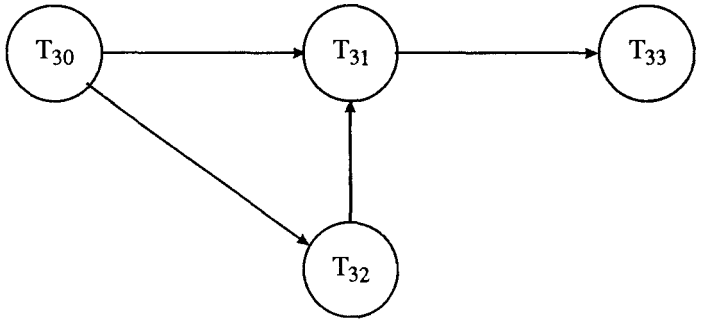


Fig.

There are the following situation in Fig.

- Transaction T_{30} is waiting for transaction T_{31} & T_{32}.
- Transaction T_{32} is waiting for transaction T_{31}.
- Transaction T_{31} is waiting for transaction T_{33}

Since the graph has no cycle, the system is not in a deadlock state.

Suppose now that transaction T_{33} is requesting an item held by T_{32}.

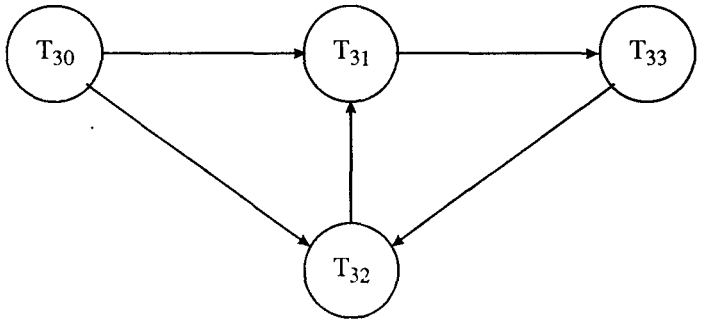


Fig.

The edge $T_{33} \longrightarrow T_{32}$ is added to the wait for graph, resulting New graph (2) contain the cycle.

Since the Fig (2) contain the cycle.

$$T_{31} \longrightarrow T_{33} \longrightarrow T_{32} \longrightarrow T_{31}$$

Thus it implies that transaction T_{31}, T_{32} & T_{33} are all deadlock.

Recovery from Deadlock

When a detection algorithm determines that a deadlock exists, the system must recover from the deadlock.

There are two most common solution to recover the deadlock.

(1) Abort the transaction one by one to break the deadlock.

(2) Abort the all transaction which participate in deadlock to break the deadlock.

There are three actions need to be taken.

(1) Selection of a Victim : In a set of deadlock transaction, we must determine which transaction to roll back to break the deadlock. We should rollback those transaction that will loss the minimum cast.

(2) Roll back : Once we have decided that a particular transaction must be rolled back. We must determine how far this transaction should be rolled back? The simple solution is a total rollback, Abort the transaction and then restart it.

(3) Starvation : Starvation occurs when a transaction cannot proceed for an indefinite period of time while other transactions in the system continue normally. This may occur if the waiting scheme for locked items is unfair, giving priority to some transations over others. The rest solution of starvation to allow some transactions to have priority over others but increases the priority of a transaction the longer it waits, until it eventually gets the highest priority & proceeds.

4.11 CONCEPT OF PHANTOM DEADLOCK

A deadlock that is detected but is not really a deadlock is called Phantom deadlock.

- Autonomous aborts may cause these deadlocks.
- Phantom deadlock occurs when an external observer can see deadlock where there is none.

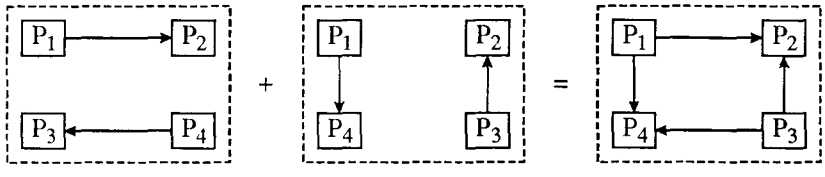
Following are four conditions.

- R_1 is stored at S_1
- R_2 is stored at S_2
- T_1 runs at S_3
- T_3 runs at S_4

Two transactions run concurrently

T_2 : lock R_1	T_2 : unlock	T_2 : lock R_2
T_1 : lock R_1	T_1 : unlock R_1	T_1 : lock R_2
T_1 : unlock R_2	T_2 : lock R_2.	

False Deadlock : What if each site maintain its local view of the system state, and periodically sends this to the control site ?



Solved Problems

Q. 1. *Explain why a transaction execution should be atomic. Explain ACID properties, considering the following transaction.*

```
Ti :   read (A);
       A : = A − 50;
       write (A);
       read (B);
       B : = B + 50;
       write (B)
```

(UPTU 2003, 04)

Ans. Considering that a transaction Ti starts execution when database is in a consistent state. If atomicity of a transaction Ti is not ensured, then its failure in the mid of its execution may leave the database in an inconsistent state. Thus the transaction should be atomic.

Explanation of ACID properties with respect to following transaction :

```
Ti :   read (A);
       A : = A − 50;
       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 that is if the database is consistent before an execution of the transaction, then the database remains consistent after the execution of the transaction.

(ii) Atomicity : Suppose that, just before the execution of transaction Ti the values of Accounts A & B are Rs 2000 & Rs 3000 respectively.

Suppose that the failure happened after the write (A) operation but before the write values (B) operation. In this case, the values of account A & B are Rs. 1,950 & Rs. 3,000 respectively. The system destroyed Rs. 50 as a result of this failure. Thus such a state is an inconsistent state. We must ensure that such inconsistencies are not visible in database system. That is the reason for atomicity requirement.

If atomicity is present, all actions of the transaction are reflected in the database, or none are.

(iii) Durability : The durability property guarantees that, once a transaction completes successfully, all the updates do not get losts, even if there is a system failure.

If transaction Ti committed successfully, then the sum of A & B before execution and after execution always equal (same).

(iv) Isolation : The isolation property of a transaction ensures that the concurrent execution of transactions results in a system state that is equivalent to a state that could have been obtained had these transactions executed one at a time in some order.

Q. 2. *A transaction is failed & enters into aborted State. Mention the conditions under which we can restart the transaction & kill the transaction.* (UPTU 2003, 04)

Ans. A transaction enters the failed state after the system determined that the transaction can no longer proceed with its normal execution. Such a transaction must be rolled back. Then, it enters the aborted state.

At this point, the system has two options :

(i) Restart : It can restart the transaction, but only if the transaction was aborted as a result of

some hardware or software error that was not created through the internal logic of the transaction. A restarted transaction is considered to be a new transaction.

(ii) Kill : It can kill the transaction. It usually does so because of some internal logical error that can be corrected only by rewriting the application program, or because the input was bad, or because the desired data were not found in the database.

e.g., Suppose a program developer has written a program to find the sum of two accounts but after the execution of the transaction the result is multiplication of two accounts, then in such situation the transaction will be killed.

Q. 3. *Consider the following transaction :*

T_1 : *read (A);*
read (B);
if A = 0 then *B : = B + 1*
write (B);
T_2 : *read (B);*
read (A);
if B = 0 then *A : A = 1;*
write (A);

Add lock and unlock instruction to transactions T_1 *and* T_2, *so that they observe the two-phase locking protocol.* **(UPTU 2003, 04)**

Ans. Lock – X = Exclusive lock
Lock – s = Shared lock

T_1	T_2
Lock – S (A)	
Lock – X (B)	
read (A)	
read (B)	
if A = 0 then B : = B + 1	
write (B)	
Unlock – S (A)	
Unlock – X (B)	
	Lock – S (B)
	Lock – X (A)
	read (A)
	read (B)
	if B = 0 then A : = A + 1
	write (A)
	Unlock – X (A)
	Unlock – S (B)

Q.4. *Consider the following two transactions*

T_1 : *read (A)*
read (B)
if A = 0 then *B : = B + 1*
write (B)

T_2 : *read (B)*
read (A)
if B : = 0 then $A : = A + 1$
write (A)

Let the consistency requirement be
$A = 0 \; V \; B = 0$ *with* $A = B = 0$ *the initial values.*

(i) Show that every serial execution involving these two transactions preserves the consistency.

(ii) Show a concurrent execution T_1 & T_2 *that produces a non-serializable schedule.*

(iii) Is there a concurrent execution of T_1 & T_2 *that produces a serializable schedule?*

(UPTU 2002, 03)

Ans.

(i) Serial execution of transaction T_1 followed by T_2 = 1 and A = 0, which preserves consistent of the database and another serial execution of transaction T_2 followed by T_1 produces B = 0 & A = 1, which preserves consistency of the database.

(ii) Consider the following schedule which have concurrent exeuction of T_1 & T_2

T_1	T_2
read (A)	
	read (B)
read (B)	
if A = 0 then	
B : = B + 1	
write (B)	
	read (A)
	if B = 0 then
	A : = A + 1
	write (A)

If all instructions of T_2 swapping before read (A) instruction of T_1 then write (A) of T_2 is conflicting read (A) of T_1. So that we are not able to produce serial schedule T_2 followed by T_1 as well as if all instruction of T_1 swapping before read (B) instruction of T_2 then write (B) of T_1 is conflicting read (B) of T_2. So that we are not able to produce serial schedule T_1 followed by T_2. Hence the above concurrent execution of T_1 & T_2 produce a non-serializable schedule.

(iii) No, there are no concurrent execution of T_1 & T_2 that produces a serializable schedule, because it does not produce serial schedule.

Q. 5. *State whether the following schedule is conflict serializable or not. Justify your answer.*

(UPTU 2003, 04)

T_1	T_2
read (A)	
write (A)	
	read (B)
	write (B)
read (B)	
write (B)	
	read (A)
	write (A)

Ans. In this schedule, the write (B) of T_2 conflicts with the Read (A) of T_1, while write (A) of T_1 does not conflict with Read (B) of T_2, because the two instructions access different data items.

So we can swap non conflicting instructions.

- Swap the write (A) of T_1 with Read (B) of T_2.
- Swap the write (B) of T_1 with Read (A) of T_2.

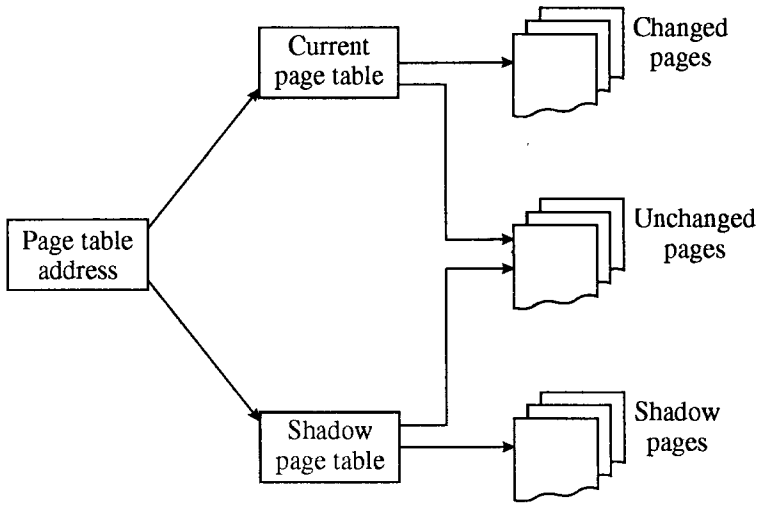
After Swapping :

T_1	T_2
read (A)	
	read (B)
wtite (A)	
	write (B)
read (B)	
	read (A)
write (B)	
	write (A)

Thus we can say the above schedule is conflict serializable.

Q. 6. *Write a short notes on shadow paging.*

Ans. Shadow paging : Shadow paging was introduced by Lorie in 1977 as an alternative to the log-based recovery schemes. The shadow paging technique does not require the use of a transaction log in a single user environment.



In shadow page scheme, the database is considered to be made up of logical units of storage of fixed-size disk pages (or disk blocks). The pages are mapped into physical blocks of storage by means of a page table, with one entry for each logical page of the database. This entry contains the block number of the physical storage where this page is stored. Thus, the shadow paging scheme is one possible form of the indirect page allocation.

The shadow paging technique maintains two page tables during the life of a transaction namely:

(i) A current page table

(ii) A shadow page table.

The shadow page is the original page table and the transaction addresses the database using

current page table. At the sort of the transaction the two tables are same and both point to the same blocks of physical storage. The shadow page table is never changed thereafter and is used to restore the database in the event of a system failure.

However, current page table entries may change during execution of a transaction. The current page table is used to record all updates to the database. When the transaction completes the current page table becomes the shadow page table.

Advantages of shadow paging :

(i) The overhead of maintaining the transaction log file is eliminated.

(ii) Since there is no need for undo or redo operations, recovery is significantly faster.

Disadvantages of shadow paging :

(i) Data fragmentation or scattering.

(ii) Need for periodic garbage collection to reclaim inaccessible blocks.

Review Questions

1. Explain the concept of transaction. Draw and explain stage diagram of a transaction.
2. Describe ACID properties of the transaction? Explain serializability with suitable example.
3. Explain why a transaction execution should be atomic. Explain ACID properties considering the following transaction.

T_1 :
read (A);
$A := A - 50$;
write (A);
read (B);
$B := B + 50$;
write (B);

4. A transaction is failed and enters into aborted state. Mention the conditions under which we can restart the transaction and kill the transaction.
5. Why have database system implementers paid much attention to ACID properties
6. What do you mean by serializability? Differentiate between conflict serializability and view serializability schedules.
7. What do you mean by transaction? What do you mean by atomicity of transaction? Explain the types of failure which may cause abortion of transaction. Explain the salient features of deferred database modification and immediate database modification scheme for recovery.
8. What do you mean by schedule? When is a schedule called serializable? What are conflict serialization schedules? Show, whether the following schedules are conflict equivalent or not.

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

Hint : A schedule is said to be serializable, over a set S of committed transactions whose effect

on any consistence database instance is guranteed to be identical to that of some complete serial schedule over S.

That is the database instance that results from executing the given schedule is identical to the database instance that results from executing the transactions in some serial order.

(UPTU 2003, 04)

9. State the condition when two schedules are considered as view equivalent. State whether the following schedule is view serializable. Justify your answer.

T	T_2	T_3
Read(A)		
	write(A)	
write(A)		
		write(A)

10. How is the checkpoint information used in the recovery operation following a system crash?
11. Show how the backward error recovery technique is applied to a DBMS?
12. Explain the working of lock manager.
13. What is deadlock? How is a deadlock detected? Enumerate the method for recovery from the deadlock.
14. Explain serializability? When is a schedule conflict serializable? What are recoverable? What are cascadeles schedules?
15. What is deadlock? How can a deadlock be avoided?
16. Describe the wait_die and wound_wait techniques for deadlock prevention.
17. What is difference between wait_die and wound_wait techniques for deadlock prevention?
18. What is a wait for graph? Where is it used? Explain with an example.
19. Discuss the different types of transaction failures that may occur in a database environment.
20. What is database recovery? What is meant by forward and backward recovery? Explain with an example.
21. What is a checkpoint? How is the checkpoint information used in the recovery operation following a system crash?
22. What are the types of damages that can take place to the database? Explain.
23. How many techniques are used for recovery from non-physical or transaction failure?

    Ans. The following two techniques are used for recovery from non-physical or transaction failure:

    (i) Deferred update

    (ii) Immediate update
24. Explain the salient features of deferred database modification and immediate database modification schemes of recovery. (UPTU 2003, 04)

    Ans. Deferred Database Modification (Update) : In case of the deferred update technique, updates are not written to the database until after a transaction has reached its COMMIT point. In other words, the updates database are deferred (or postponed) until the transaction completes its execution successfully and reaches its commit points. During transaction execution, the updates are recorded only in the transaction log and in the cache buffers. After the transaction reaches its commit point and the transaction log is forced-written to disk, the updates are recorded in the database. If a transaction fails before it reaches this point, it will not have modified the database and so no undoing of changes will be necessary.

    In case of deferred update, the transaction log file is used in the following ways :

(i) When a transaction T begins, transaction begin (or $<T$, BEGIN$>$) is written to the transaction log.
(ii) During the execution of transaction T, a new log record containing all log data specified previously.
(iii) When all actions comprising transaction T are successfully committed, we say that the transaction T partially commits and the record "$<T$, COMMIT$>$" are written to the transaction log. After transaction T partially commits, the records associated with transaction T in the transaction log are used in executing the actual updates by writting to the appropriate records in the database.
(iv) If a transaction T aborts, the transaction log record is ignored for the transaction T and write is not performed.

Immediate Update : In the case of immediate update technique, all updates to the database are applied immediately as they occur without waiting to reach the COMMIT point and a record of all changes is kept in the transaction log.

In the case of immediate update, the transaction log file is used in the following ways :

(i) When a transaction T begins, transaction begin (or "$<T$, BEGIN$>$") is written to the transaction log.
(ii) When a write operation is performed, a record containing the necessary data is written to the transaction log file.
(iii) Once the transaction log is written, the update is written to the database buffers.
(iv) The updates to the database itself are written when the buffers are next flushed (transferred) to secondary storage.
(v) When the transaction T commits, a transaction commit ("$<T$, COMMIT$>$") record is written to the transaction log.
(vi) If the transaction log reveals the record "$<T$, BEGIN$>$" but does not reveal "$<T$, COMMIT$>$", transaction T is undone. The old values of affected data items are restored and transaction T is restarted.
(vii) If the transaction log contains both of the preceeding records, transaction T is redone. The transaction is not restarted.

□□□