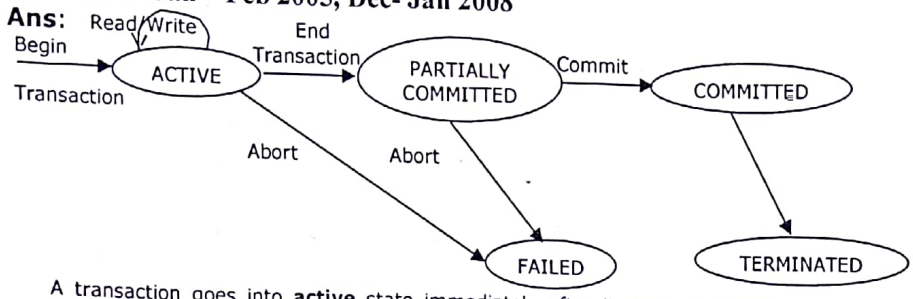
Transaction, Concurrency control & Recovery notes

Q1 : Explain with a state transition diagram, the various states of a transaction execution. Jan – Feb 2005, Dec- Jan 2008

Ans:



A transaction goes into **active** state immediately after it starts execution, where it can issue READ & WRITE operations. When the transaction ends, it moves to the **partially committed** state. At this point certain checks be made to ensure that the transaction did not interfere with execution of other transactions & a system failure will not result in an inability to record the changes of the transaction permanently. Once both checks are successful, the transaction is said to have reached its commit point & enters the **committed** state, which indicates the successful execution of that transaction.

A transaction can go to the **failed** state if one of checks fail or it is aborted during its active sate. The transaction may then have to be rolled back, to undo the effect of its write operations on the database. The **terminated** corresponds to the transaction leaving the system. Failed or aborted transactions may be restarted later as brand new transactions.

Q2 : Write a note on System Log.

Ans: To recover from transaction failures, the system maintains a log (or journal) which keeps track of all operations of the transactions that affect the database.

A unique **transaction_id (T)** is generated automatically by the system & is used to identify each transaction. The following types of entries are made in the log:

1. [start_transaction, T]: Records that transaction T has started execution.
2. [write_item, T, x, old_value, new_value]: Records that transaction T has changed the value of database item x from old_value to new_value.
3. [read_item, T, x]: Records that transaction T has read the value of database item x.
4. [commit, t]: Records that transaction T has successfully completed & affirms that its effect can be committed to database.
5. [abort, t]: Records that transaction T has been aborted.

Q3 : Explain briefly the various transaction states.

Ans: The various transaction states are as follows:

1. BEGIN_TRANSACTION: This marks the beginning of transaction execution.
2. READ OR WRITE: These specify the read or write operations on the database items that are executed as part of a transaction.
3. END_TRANSACTION: This specifies that read & write operations of a transaction have ended. At this poin, it may be necessary to check whether the changes done by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it has violated the concurrency control or for some other reason.
4. COMMIT_TRANSACTION: This signals the successful end of the transaction so that any changes executed by the transaction can be safely committed to the database & will not be undone.
5. ROLLBACK (OR ABORT): This signals that the transaction has ended unsuccessful, so that any changes executed by the transaction may have applied to the database must be undone.

Q4 : What are the reasons for a transaction to fail?

Ans: 1. A computer failure (or system crash): The hardware of a compute system may be crashed because of an error during transaction execution & the contents of the main memory may be lost.

2. A transaction or system error: Some operation in the transaction may cause it to fail, such as division by zero, etc.

3. Local errors or exceptional conditions detected by the transaction: During transaction execution certain condition may occur that necessitate cancellation of the transaction.

4. Concurrency control enforcement: The Concurrency control method may decide to abort the transaction to be restarted later.

5. Disk Failure: Some disk blocks may lose their data because of read or write malfunction or because of a disk read/write head crash.

6. Physical problems or catastrophes: This includes problems like power or air conditioning failure, fire, theft or sabotage.

Q5 : What are the desirable properties (or ACID) a transaction? Jan – Feb 2005, Jul – Aug 2005, Jul 2006

Ans: 1. Atomicity: A transaction is an atomic unit of processing. It's either performed in its entirety or not performed at all.

2. Consistency Preservation: A correct execution of the transaction must take database from one consistent state to another.

3. Isolation: A transaction should not make its updations visible to other transactions until it's committed.

4. Durability (or permanency): Once a transaction changes the database & changes are committed, these changes must never be lost because of subsequent failures.

Q6 : What is a complete schedule?

Ans: A schedule S of n transactions T1, T2, T3.....,Tn is said to be a complete schedule if the following conditions hold:

1. The operations in S are exactly those operations in T1, T2, T3.....,Tn including a commit or abort operation as the last operation for each transaction in the schedule.
2. For any pair of operations for some transaction Ti, their order of appearance in S is the same as their order of appearance in Ti.
3. For any 2 conflicting operations, one of the two must occur before the other in the schedule.

Q6 : How the schedules are characterized based on the recoverability? Jul 2007

Ans: Once a transaction T is committed, it should never be necessary to rollback. The schedules that meet this criteria are called **recoverable schedules.**

A schedule S is said to be recoverable, if no transaction T in S commits until all transactions T^{l} that have written an item that T reads have committed.

Q7 : What do you mean by concurrent execution? Explain why it is required? Jul 2006

Ans: If only a single CPU exists, it can actually process at most one program at a time. However multiprogramming operating systems execute some commands from one program then suspend that program & execute some commands from the next program & so on. A program is resumed at the point where it was suspended when it gets its turn to use the CPU again. This type execution is called concurrent execution in an interleaved way.

If the concurrent execution is not controlled, several problems can occur. These can be illustrated with the help of a Airline reservation database.

Consider the transaction T1 that cancels N reservations from one flight whose number of reserved seats is stored in the database item named x & reserves the same number of seats on another flight whose number of reserved seats is stored in the database item named y. The code for this task can be given as below:

```
T1
Read_item(x);
X:=x-N;
Write_item(x);
Read_item(y);
Y:=y+N;
Write_item(y);
```

Another transaction T2, given below, just reserved M seats on the first flight referenced in transaction T1.

```
T2
Read_item(x);
X:=x+M;
Write_item(x);
```

The following types of problems may be encountered whenever two or more transactions occur concurrently.

1. The Lost Update problem: This occurs when two transactions that access the same database items have their operations interleaved in away that make value of some data item incorrect. Suppose that operations of transactions T1 & T2 are interleaved as in the figure below:

T1 T2

```
Read_item(x);
X:=x-N;
                                    Read_item(x);
                                    X:=x+M;
Write_item(x);
Read_item(y);
                                    Write_item(x);
Y:=y+N;
Write_item(y);
```

Then the final value of x is incorrect, because T2 reads the value of x before T1 changes it in the database & hence the updated value resulting from T1 is lost.

2. The Temporary Update (or dirty Read) problem: This occurs when one transaction updates a database item & then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value.

The following figure shows an example where T1 updates item x & then fails before completion, so the system must change x back to its original value. Before it can do so another transaction T2 reads the temporary value of x, which will not be recorded permanently in the database.

```
   T1                                T2
Read_item(x);
X:=x-N;
Write_item(x);

                                    Read_item(x);
                                    X:=x+M;
                                    Write_item(x);

Read_item(y);
```

→ T1 fails.

3. The Incorrect Summary problem: If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may use some values before they are updated & others after updated.

4. Unrepeatable Read problem: In this case, a transaction T reads an item twice & the item is changed by another transaction T' between the two reads. Hence, T receives different values for its two reads of the same item.

Q8 : Explain the various locking techniques for concurrency control. **Jan – Feb 2005**

Ans: Binary Locking Technique: A binary lock can have two states or values: locked & unlocked (or 0 & 1). A distinct lock is associated with each database item x. If the value of lock on x is 1, then item x cannot be accessed by a database operation that requests it. If the value of lock on x is 0, then item x can be accessed when requested. The value of the lock associated with item x is referred to as LOCK(x).

Two operations lock_item & unlock_item must be included in the transactions when binary locking is used. The operations lock_item & unlock_item are illustrated below:

lock_item (x)

```
B: if lock_item (x) =0 then
        begin
        lock_item (x)← 1
   else begin
        wait (until lock_item (x)=0 & the lock manager wakes up the transaction)
   go to B
   end ;
```

unlock_item (x)

```
lock_item (x)← 0
If any transactions are waiting then wake up one of the waiting transactions
```

When the binary locking scheme is used, every transaction must obey the following rules.

1. A transaction T must issue the operation lock_item (x) before any read_item (x) or write_item (x) operations are performed in T.
2. A transaction T must issue the operation unlock_item (x) after all read_item (x) or write_item (x) operations are performed in T.
3. A transaction T will not issue the operation lock_item (x) if it already holds a lock on item x.
4. A transaction T will not issue the operation unlock_item (x) unless it already holds lock on item x.

Shared & Exclusive Locks (Multiple Mode Lock): In this scheme there are three locking operations: read_lock(x), write_lock(x) & unlock(x). A lock associated with an item x, LOCK(x) has three possible states: "read_locked","write_locked" & "unlocked". A read_locked item is called **share_locked,** because other transactions are allowed to read the item where as a write_locked item is called **exclusive_locked**, because a single transaction exclusively holds the lock on the item.

The operations read_lock(x), write_lock(x) & unlock(x) are described below:

read_lock(x)

```
B: if LOCK(x) ="unlocked" then
    Begin
     LOCK(x)← "read_locked";
     End;
    Else if LOCK(x)="read_locked" then
     Begin
       no_of_reads(x)← no_of_reads(x) + 1;
     else
      Begin
        Wait ( until LOCK(x) ="unlocked" and the lock manager wakes up the transaction);
        Go to B
     End;
```

write_lock(x)

```
B: if LOCK(x) ="unlocked" then
    Begin
     LOCK(x)← "write_locked";
     End;
     else
      Begin
        Wait ( until LOCK(x) ="unlocked" and the lock manager wakes up the transaction);
        Go to B
     End;
```

unlock_item(x)

```
   if LOCK(x) ="write_locked" then
    Begin
     LOCK(x)← "un_locked";
     End;
    Else if LOCK(x)="read_locked" then
     Begin
       no_of_reads(x)← no_of_reads(x) - 1;
     if no_of_reads(x)=0 then
      Begin
        LOCK(x) ="unlocked";
     Wake up one of the waiting transactions, if any;
     End;
```

When multiple mode locking scheme is used, the system must enforce the following rules:

1. A transaction T must issue the operation read_lock (x) or write_lock(x) before any read_item (x) operations are performed in T.
2. A transaction T must issue the operation write_lock(x) before any write_item (x) operations are performed in T.
3. A transaction T must not issue the operation unlock (x) after all read_item(x) & write_item(x) are performed on item x.
4. A transaction T will not issue the operation read_lock (x) if it already holds a read_lock or write_lock on item x.
5. A transaction T will not issue the operation write_lock (x) if it already holds a read_lock or write_lock on item x.
6. A transaction T will not issue the operation unlock (x) unless it already holds a read_lock or a write_lock on item x.

Two Phase Locking (2PL): A transaction is said to follow the two phase locking protocol if all locking operations (read_lock,write_lock) precede the first unlock operation in the transaction. Such transaction can be divided into two phases: an **Expanding Phase (or Growing Phase)**, during which new locks on items can be acquired but none can be released. And a **Shrinking Phase**, during which existing locks can be released but no new locks can be acquired.

Conservative 2PL: The static 2PL requires a transaction to lock all the item it accesses before the transaction begins execution, by predeclaring its read set & write set. If any of the predeclared items needed cannot be locked, the transaction does not lock any item. Instead, it waits until all the items are available for locking. Conservative 2PL is a dead lock free protocol.
Strict 2PL: With strict 2PL, a transaction T does not release any of its exclusive (write) locks until after it commits or aborts. Hence, no other transaction can read or write an item that is written by T unless T has committed.
Rigorous 2PL: In this variation, a transaction T does not release any of its locks (exclusive or shared) until after it commits or aborts.

Both Strict 2PL & Rigorous 2Pl guarantee strict schedules.

Q9 : What is Dead Lock? Explain the wait_die & wound_wait protocols for dead lock prevention. Jul 2007

Ans: Dead lock occurs , when each of the transaction wait for the other to release the lock on an item.
Ex:

T1	T2
read_lock(y);	
read_item(y);	
	read_lock(x)
	read_item(x)
write_lock(x);	
	write_lock(y);

Here the 2 transactions T1 & T2 are dead locked in a schedule. T1 is waiting for T2 to release item x, while T2 is waiting for T1 to release item y. Mean while neither can proceed to unlock the item that the other is waiting for, & other transactions can access neither x nor y.

Suppose that transaction T_i tries to lock an item x, but is not able to do so because x is locked by some other transaction T_j. The rules to be followed are given below for different schemes:
Wait_die: if $TS(T_i) < TS(T_j)$ then T_i is allowed to wait
Otherwise abort T_i & restart it with same time stamp
In wait_die, an older transaction is allowed to wait on a younger transaction, where as a younger transaction requesting an item held by older transaction is aborted & restarted.

Wound_wai: if $TS(T_i) < TS(T_j)$ then abort T_j & restart it with same time stamp
Otherwise abort T_i is allowed to wait
In wound_wait, a younger transaction is allowed to wait on an older transaction, where an older transaction requesting an item held by younger preempts the younger transaction by aborting.

Q10: Explain the two main techniques used for recovery from no catastrophic transaction failures. Dec-Jan 2008

Ans: The 2 main techniques used for recovery from no catastrophic transaction failures are:
1. Deferred update technique
2. Immediate update technique

1. **Immediate update technique:** In this technique, the database may be updated by some operations of a transaction before transaction reaches its commit point. However, these operations are recorded on disk by force-writing before they are applied to the database. If a transaction fails after recording some changes but before reaching its commit point, the effect its operations on the database must be undone. That is the transaction must be rolled back. The immediate update, both undo & redo are required during recovery, so it's known as **undo/redo** algorithm.
2. **Deferred update Technique:** A typical deferred update protocol can be stated as follows:
   i. A transaction cannot change the database until it reaches its commit point.
   ii. A transaction does not reach its commit point until all its update operations are recorded in the log & the log is force written to disk.

Because the database is never updated until after the transaction commits, there is never a need to undo any operations. Hence, this technique is known as the **no-undo / redo** algorithm. The redo is required in case the system fails after the transaction commits but before all of its changes are required in the database.

Recovery Using Deferred Update in a Single User environment:

PROCEDURE RDU_S: Use two lists of transactions. The committed transactions since the last check point & the active transactions. Apply the redo operations to all write_item operations of the committed transactions from the log in the order in which they were written to the log. Restart the active transactions.

The REDO procedure is defined as below:

REDO (WRITE_OP): Redoing a write operation WRITE_OP consists of examining its log entry [write_item,t,x,old_value,new_value] & setting the value of item x in the database to new_value which is the after image (AFIM)

Recovery Using Deferred Update in a Multi User environment:

PROCEDURE RDU_M: Use two lists of transactions. The committed transactions since the last check point & the active transactions. Apply the redo operations to all write_item operations of the committed transactions from the log in the order in which they were written to the log. Restart the active transactions.

The REDO procedure is defined as below:

REDO (WRITE_OP): Redoing a write operation WRITE_OP consists of examining its log entry [write_item,t,x,old_value,new_value] & setting the value of item x in the database to new_value which is the after image (AFIM)

Q11: Explain with an example how the use of two-phase locking results in a serializable schedule. Jan – Feb 2006

Ans: Consider the following example

T_1	T_2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
X:=X+Y;	Y:=X+Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Fig A

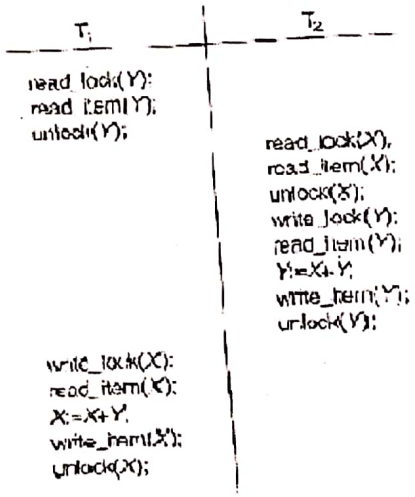


Fig B

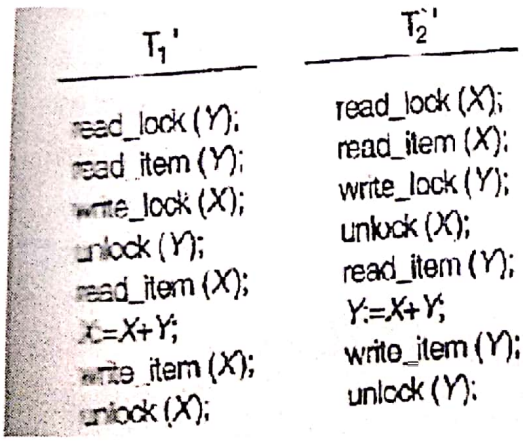


Fig C

Here the transactions T1 & T2 of Fig- A don't follow the two-phase locking. This is because the write_lock(X) operation follows the unlock(Y) operation in T1 and similarly the write_lock(Y) operation follows the unlock(X) operation in T2.
Figure-C shows transactions T1 ' & T2 ' after enforcing 2PL. Now the schedule in shown in fig-B is not permitted for T1 ' & T2 ' under the rules of locking. This is because T1 ' will issue its write_lock(X) before it unlocks item Y; When T2 ' issues its read_lock(X) it is forced to wait until T1 ' releases the lock by issuing an unlock(X) in the schedule. Thus it can be proved that every transaction in a schedule that follows 2PL protocol, the schedule is guaranteed to be serializable.

Q12: what is serializability ? Jan – Feb 2006

Ans: A schedule S of n transactions is said to be serializable if it is equivalent to some serial schedule of the same n transactions.

Q13: Explain multiversion concurrency control techniques based on time stamp ordering. Jan – Feb 2006

Ans: This technique keeps the old value of a data item when the item is updated. As a result multiple versions of the same data item are maintained. Hence the name multiversion protocol. The major draw back of this protocol is that it requires more storage space. This technique maintains two timestamps:

1.read_TS(X_i) : The read timestamp of X_i is the largest of all timestamps of transactions that have successfully read version X_i

2.write_TS(X_i) : The write timestamp of X_i is the timestamp of the transaction that wrote the value of X_i

Following rules are used:

1.If transaction T issues a write_item(X) operation & version I of X has the highest write_TS(X_i) of all versions of X that is also less than or equal to TS(T), & read_TS(X_i) > TS(T) , then abort & rollback transaction T; otherwise create a new version X_j of X with read_TS(X_j) = write_TS(X_j) = TS(T).
2. If transaction T issues a read _item(X) operation, find the version I of X that has the highest write_TS(X_i) of all versions of X that is also less than or equal to TS(T); then return the value of X_i to transaction T & set the value of read_TS(X_i) to the larger of TS(T) & the current read_TS(X_i).

Q14: Explain intension mode locking. Jan – Feb 2006

Ans: The idea behind intension locking is for a transaction to indicate , along the path from the root to the desired node, what type of lock it will require from one of the node's descendents. There are 3 types of intension locks:

1. Intension-shared (IS) indicates that a shared lock will be requested on some descendent nodes.
2. Intension-Exclusive (IX) indicates that a exclusive lock will be requested on some descendent nodes.
3. Shared-Intension-Exclusive (SIX) indicates that the current node is locked in shared mode but an exclusive lock will be requested on some descendent nodes.

Following table shows the lock compatibity for the three intension locks

	IS	IX	S	SIX	X
IS	YES	YES	YES	YES	NO
IX	YES	YES	NO	NO	NO
S	YES	NO	YES	NO	NO
SIX	YES	NO	NO	NO	NO
X	NO	NO	NO	NO	NO

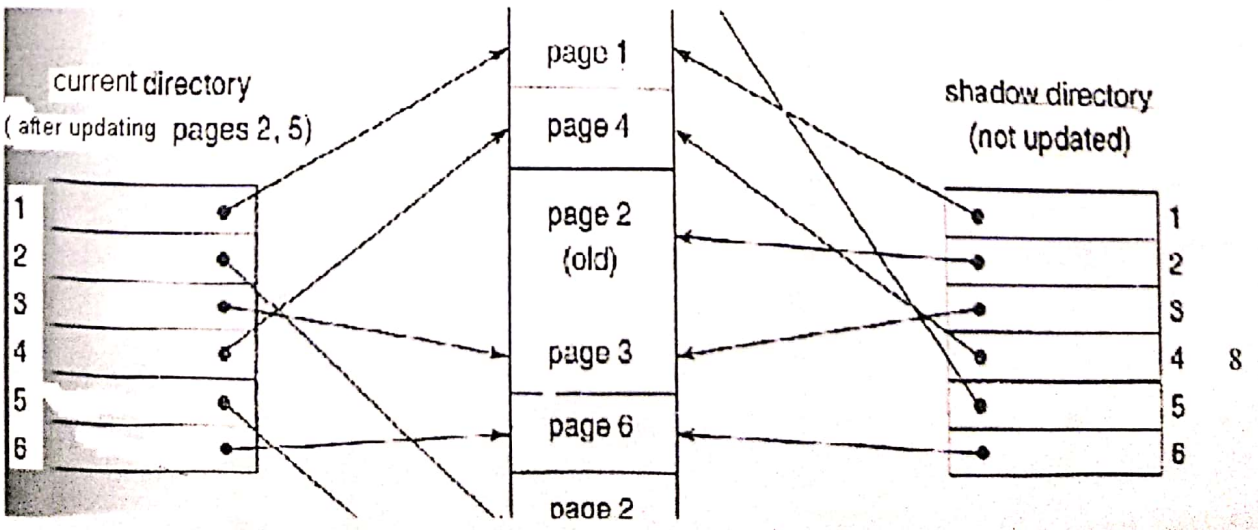
Q15: What is Write-Ahead logging Jul – Aug 2005

Ans: When in-place updating is used, it s necessary to use a log for recovery. In this case, the recovery mechanism must ensure that the Before Image (BFIM) of the data item is recorded in the appropriate log entry & that the log entry is flushed to disk before the BFIM is overwritten with After Image (AFIM) in the database on disk. This process is known as write-ahead logging.

Q16:Explain shadow paging Jan – Feb 2006, Jul 2006

Ans: With shadow paging a directory with n entries is maintained where the i^{th} entry points to the i^{th} database page on the disk. The directory is kept in the main memory if the size is not too large. When a transaction begins executing, the current directory whose entries point to the most recent or current database pages on disk is copied into a shadow directory. The shadow directoty is then saved on disk while the current directory is used by the transaction.

During transaction execution the shadow directory is never modified. When a write item operation is performed, a new copy of the modified database page is created, but the old copy of that page is not overwritten. Instead, the new page is written elsewhere. The current directory entry is modified to point the new disk block where as the shadow directory is not modified & continues to point to the old unmodified block. Following figure illustrates the concept of shadow paging.



For pages updated by the transaction, two versions are kept. The old version is referenced by the shadow directory, & new version by current directory.

To recover from a failure during transaction execution, it is sufficient to free the modified database pages & to discard the current directory. The state of the database before transaction execution is available through the shadow directory, & that state is recovered by reinstating the shadow directory. The database is thus returned to its state prior to the transaction that was executing when the crash occurred, & any modified pages are discarded. Committing a transaction corresponds to discarding the previous shadow directory. Since recovery involves neither undoing nor redoing data items, this technique can also be categorized as a NO-UNDO/NO-REDO technique for recovery.

Q17: Write a note on buffering of blocks

Ans: Each buffer is associated with a **dirty bit** in the cache, which can be included in the directory entry, to indicate whether or not the buffer has been modified. When a page is first read from the database disk into a cache buffer, the cache directory is updated with the new disk page address, and the dirty bit is set to 0(zero). As soon as buffer is modified, the dirty for the corresponding directory entry is set to 1(one).

When the buffer contents are replaced (flushed) from the cache, the contents must first be written back to the corresponding disk page *if it's dirty bit is* 1. Another bit, called the pin-unpin bit, is also needed – a page in the cache is **pinned** (bit value 1(one)) if it cannot be written back to disk as yet.

Two main strategies can be employed when flushing a modified buffer to disk.

The first strategy, known as **in-place updating**, writes the buffer back to the *same original disk location*, thus overwriting the old value of any changed data items on disk[2]. Hence, a single copy of each database disk block is maintained. The s4econd strategy, known as **shadowing,** writes an updated buffer at a different disk location multiple versions of data items can be maintained.

Q18: Explain Steal / no-steal approaches

Ans: If a cache page updated by a transaction cannot be written to disk before the transaction commits, this is called a **no-steal approach.** The pin-unpin bit indicates if a page cannot be written back to disk. Otherwise, if the protocol allows writing an updated buffer before the transaction commits, it is called **steal.** Steal is used when the dbms cache (buffer) manger needs a buffer frame for another transaction and the buffer manager replaces an existing page that had been updated but whose transaction has not committed.

Q19: Explain force / no- force approaches

Ans: If all pages updated by a transaction are immediately written to disk the transaction commits, this is called a **fore approach.** Otherwise, it is called **no-force.**

Q20: What is a check point? Explain in brief.

Ans: A (check point) record is written into the log periodically at that point when the system writes out to the database on disc all **DBMS** buffers that have been modified. Therefore all transactions that have their *(commit, T)* entries in the log before a [check point] entry do not need to have their WRITE operations redone in case of a system crash, since all their updates will be recorded in the database on disk during check pointing. The recovery manager of a DBMS must decide at what intervals to take a check point. Taking a check point consists of the following actions:

1. Suspended execution of transactions temporarily.
2. force write all main memory buffers that have been modified to disk
3. writ a [check point] record to the log, and force write the log to disk
4. resume executing transactions

Q21: Explain the concept of fuzzy check pointing.

Ans: .In this technique, the system can resume transaction processing after the [check point] record is written to the log without having to wait for the system to force write all main memory buffers that have been modified to disk. However, the previous [check point] record should remain valid till the force writing is over. To accomplish this, the system maintains a pointer to the valid check point, which continue to point to the previous [check point] record in log. Once force writing is concluded, that pointer is changed to point to the new check point in the log

Q22: Write a note on transaction rollback.

Ans: If transactions fail for whatever reason after updating the data base, it may be necessary to roll back the transaction. If any data item values have been changed by the transaction and written to the database, they must restored to their previous values [BFIMs].The undo type log entries are used to restore the old vales of the data items that must be rolled back

Q23: Write a note on cascading rollback.

Ans: If transaction T is rolled back, any transaction S that has, in the interim, read the values of data item X written by T must also be rolled back. Similarly, once S is rolled back, any transaction R that has read the vale of some data item Y written by S must also be rolled back; and so on. This phenomenon is called **cascading rollback.**

Q24: How do you achieve recovery in multidatabase system.

Or

Write a note on two-phase commit protocol. **January – February 2005**

Ans:A multidatabase system is a special distributed database system where one node may be running relational database system under Unix, another may be running object-oriented system under window and so on. A transaction may run in a distributed fashion at multiple nodes. In this execution scenario the transaction commits only when all these multiple nodes agree to commit individually the part of the transaction they were executing. This commit scheme is referred to as "*two-phase commit*" (2PC). If any one of these nodes fails or cannot commit the part of the transaction, then the transaction is aborted. Each node recovers the transaction under its own recovery protocol.

Q25: Explain ARIES recovery algorithm. June – July 2008

Ans: The ARIES Recovery Algorithm is based on:

1. WAL (Write Ahead Logging)
2. Repeating history during redo: ARIES will retrace all actions of the database system prior to the crash to reconstruct the database state when the crash occurred.
3. Logging changes during undo: It will prevent ARIES from repeating the completed undo operations if a failure occurs during recovery, which causes a restart of the recovery process.

The Log and Log Sequence Number (LSN)

A log record is written for (a) data update, (b) transaction commit,(c) transaction abort, (d) undo, and (e) transaction end. In the case of undo a compensating log record is written.

A unique LSN is associated with every log record. LSN increases monotonically and indicates the disk address of the log record it is associated with. In addition, each data page stores the

LSN of the latest log record corresponding to a change for that page. A log record stores (a) the previous LSN of that transaction, (b) the transaction ID, and (c) the type of log record.

The Transaction table and the Dirty Page table

For efficient recovery following tables are also stored in the log during check pointing:

Transaction table: Contains an entry for each active transaction, with information such as transaction ID, transaction status and the LSN of the most recent log record for the transaction.

Dirty Page table: Contains an entry for each dirty page in the buffer, which includes the page ID and the LSN corresponding to the earliest update to that page.

The following steps are performed for recovery

1. Analysis phase: Start at the begin_checkpoint record and proceed to the end_checkpoint record. Access transaction table and dirty page table are appended to the end of the log. Note that during this phase some other log records may be written to the log and transaction table may be modified. The analysis phase compiles the set of redo and undo to be performed and ends.

2. Redo phase: Starts from the point in the log up to where all dirty pages have been flushed, and move forward to the end of the log. Any change that appears in the dirty page table is redone.

3. Undo phase: Starts from the end of the log and proceeds backward while performing appropriate undo. For each undo it writes a compensating record in the log.

The recovery completes at the end of undo phase.