UNIT-5

* Deadlock:

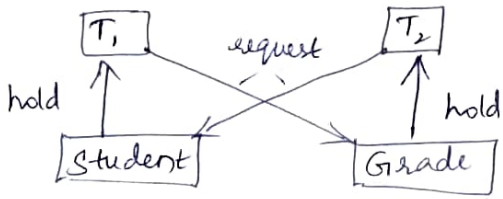
» is a condition where two or more transactions are waiting indefinitely for one another to give up locks.

- For example, In the student table, transaction T_1 holds lock on some rows & in grade table.
  ↓
  needs to update some rows

  Simultaneously T_2 transaction holds lock on some rows in grade table & needs to update the rows in the student table held by T_1

- So, here main problem is, T_1 is waiting for T_2 to release its lock & similarly T_2 is waiting for T_1 to release its lock.

  Here DBMS detects the deadlock & aborts one of the transaction

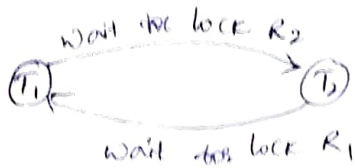


* Deadlock detection:

In a DB, when a transaction waits indefinitely to obtain a lock then the DBMS should detect whether the transaction is involved in deadlock or not. The lock manager maintains a WFG (Wait For Graph) to detect the deadlock cycle in the DB

Wait for Graph:

- A graph is created based on the transaction & their lock
- If the created graph has a cycle or closed loop, then there is a deadlock
- WFG maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph



* Deadlock - Avoidance:

- When a DB is stuck in a deadlock state, then it is better to avoid the DB rather than aborting or restarting the DB. It will be waste of time & resources.
- Deadlock avoidance mechanism is used to detect any deadlock situation in advance.
- wait for graph is used for detecting the deadlock situation but this method is suitable only for smaller DB. For larger DB, deadlock prevention method can be used.

* Deadlock prevention:

- " " is suitable for large DB.
- If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.
- DBMS analyzes the operations of the transaction whether they can create a deadlock situation or not. If it does, then DBMS never allows that transaction to be executed.

Failure classification:

To find that where the problem has occured, we generalize a failure into the following categories:

(i) Transaction failure
(ii) System crash
(iii) Disk failure

(i) Transaction failure:

" " occurs when it fails to execute or when it reaches a point from where it can't go any further. If a new transaction or process is failed then that is called as transaction failure.

Reasons for transaction failure are:

Logical errors: If a transaction can't complete its execution due to code error or internal logic error.

Syntax error: If DBMS itself terminates an active transaction because the DB system is not able to execute it.

Eg: System aborts its active transaction in case of deadlock or resource unavailability

(ii) System crash:

System failure can occur due to power failure or other h/w or s/w failure

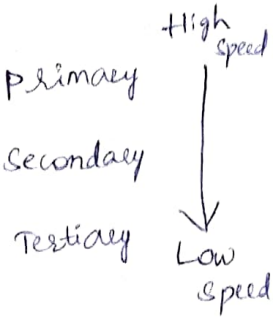
Fail stop assumption: In system crash, non volatile storage is assumed not to be corrupted.

(iii) Disk failure:

- It occurs where hard disk drives or storage devices fail frequently. It was the common problem in early days of technology evolution

- Disk failure occurs due to the formation of bad sectors, disk head crash & unreachability to the disk or any other failure, which destroys the disk storage.

* Storage Structure:



(i). Primary Storage:

- " " is used to access the data quickly.
- Volatile storage (ie., It will not store the data permanently). If any system failure or power cut, the data will be lost.
- Eg: main memory & cache memory

(ii). Secondary Storage:

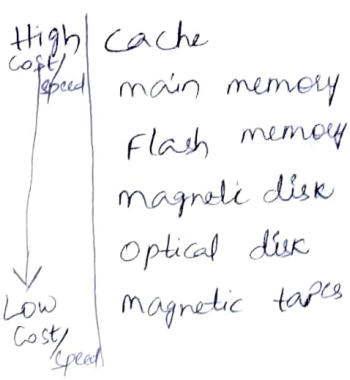
- " " also called as online storage. It allows the user to store the data permanently.
- Non volatile memory (If power failure or system crash) Data will be there. There won't be any loss of data.

Eg: Flash memory, magnetic disk storage (USB)

(iii). Tertiary Storage:

- " " is external from the computer system. It has slowest speed. But it is capable of storing large amount of data. It is known as offline storage.
- It is used for data backup

Eg. Optical storage, Tape storage (CD), (DVD)



* Recovery and Atomicity:

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

- Transactions are made of various operations, which are atomic in nature.
- When DBMS recovers from a crash, it should maintain the following.
  (i). It should check the states of all the transactions, which were being executed.
  (ii). A transaction may be in the middle of some operation, which were being executed. DBMS must ensure the atomicity of transaction in this case.
  (iii) It should check whether the transaction can be completed now or it needs to be rolled back.
  (iv) No transactions would be allowed to leave DBMS in an inconsistent state.

- There are two types of techniques, which can help DBMS in recovering as well as maintaining atomicity of a transaction.
  (i). Maintaining the logs of each transaction & writing them onto some stable storage before actually modifying the DB.
  (ii). Maintaining shadow paging, where changes are done on a volatile memory & later, actual DB is updated.

- The process of restoring the DB to a correct state in the event of a failure is known as DB recovery.
- The DB has to be restored back to consistent state from its inconsistent state that has been caused due to failure.
- Two techniques are used for recovery from transaction failure
  (i) Deferred update (No UNDO/REDO) (ii) Immediate update (UNDO/REDO)

*Log based Recovery:

- Log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered.
- If any operation performed on the DB, then it will be recorded in the log. But the process of storing the logs should be done before the actual transaction is applied in the DB.
- Let's assume there is a transaction to modify the city of a student
  (i) When the transaction is initiated, then it writes start log
  $\langle T_n$, Start$\rangle$
  (ii) When transaction modifies the city from 'Hyd' to 'Delhi'
  $\langle T_n$, City, 'Hyd', 'Delhi'$\rangle$
  (iii) When transaction is finished, then it writes another log to indicate the end of the transaction
  $\langle T_n$, Commit$\rangle$

Deferred DB Modification:

- " " technique occurs if transaction does not modify the DB until it has committed.
- All logs are created & stored in stable storage & DB is updated when a transaction commits.
- It modifies the DB after completion of transaction.

Recovery System uses following operation:

Redo(T_i) : All data items updated by T_i are set to new value

Immediate DB modification:

" " " occurs if DB modification occurs while the transaction is still active

- DB is modified immediately after every operation. It follows an actual DB modification

  - Undo (T_i): All data items updated by T_i are set to old value.
  - Redo (T_i): " " " " " new value.

* When the system is crashed, then the system consults the log to find which transaction needs to be undone & which needs to be redone.

(i). If the log contains the record $<T_i, start>$ & $<T_i, commit>$ (or) $<T_i, commit>$ then T_i needs to be redone.

(ii). If the log contains the record $<T_n, start>$ but doesn't contain the record either $<T_i, commit>$ or $<T_i, abort>$ then T_i needs to be undone.

* Recovery with Concurrent transactions:

- Concurrency control means multiple transactions can be executed at the same time & then interleaved logs occur. But, there may be changes in transaction results so maintain the order of execution of those transactions.
- During recovery, it would be very difficult for the recovery system to rollback all the logs & then start recovering
- Recovery with concurrent transactions can be done in following four ways.

(i) Interaction with concurrency control
(ii) Transaction rollback
(iii) Checkpoints
(iv) Restart recovery

(i). Interaction with Concurrency control:

- the recovery depends on concurrency control scheme that is used So, to rollback a failed transaction, we must undo the updates performed by the transaction.

(ii). Transaction rollback:

- Here, we will rollback a failed transaction by using the log.
- System scans the log backward a failed transaction, for every log record found in the log the system restores the data item.

(iii). Checkpoints:

- Checkpoints is a process of saving a snapshot of the applications state so that it can restart from that point in case of failure.
- checkpoint is a point of time at which a record is written onto the DB from the buffers
- Checkpoint shortens the recovery process.
- when it reaches the checkpoint, then the transaction will be updated into the DB. & till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till the next checkpoint & so on.
- Checkpoint is used to declare the point before which DBMS was in the consistent state & all the transactions were committed.
- we can use checkpoints to reduce the no. of log records that the system must scan when it recovers from a crash

(iv) Restart recovery: When system recovers from crash, it consists of 2 lists

- Undo list contains transactions to be undone.
- Redo " " " redone.
- Initially, both are empty. The system scans the log backward & examines each record

until it finds the first checkpoint record.

- Keeping & maintaining logs in real-time is difficult. As time passes, the log file will grow too big. Checkpoint is a mechanism where all previous logs are removed from system & stored permanently in a storage disk.
- checkpoint declares a point before which DBMS was in consistent state & all transactions were committed.
- If the recovery system sees a log with $<T_n$, Start> & $<T_n$, Commit> or just $<T_n$, Commit>, it puts the transaction in the redo-list.
- If log with $<T_n$, Start> but no commit, it will be going into undo-list.
- All transactions in undo-list are then undone & their logs are removed.
- " " redo-list & their previous logs are removed & then redone before saving their logs.

* Buffer management:

- DB buffer is a temporary storage area in the main memory. It allows storing the data temporarily when moving from one place to another.
- DB buffer stores a copy of disk blocks
- Buffer manager is responsible for allocating space to the buffer in order to store data into the buffers.
- If a user request a particular block & the block is available in the buffer, the buffer mgr provides block address in the main memory.
- If the block is not available in the buffer, buffer mgr allocates the block in the buffer.
- If free space is not available, it throws out some existing blocks from

the buffer to allocate the required space for the new block.

- the blocks which are thrown are written back to the disk only if they are recently modified when writing on the disk.
- If the user requests such thrown out blocks, buffer mgr reads the requested block from disk to the buffer & then passes the address of the " " to the user in the main memory
- The internal actions of the buffer mgr are not visible to the program that may create any problem in disk block requests. The buffer mgr is just like a virtual machine.

(i). Buffer replacement strategy:

If no space is left in the buffer, it is required to remove an existing block from the buffer before allocating the new one.

The various OS uses the LRU (Least Recently Used) scheme.

- In LRU, the block that was least recently used is removed from the buffer & written back to the disk. this type of replacement strategy is known as buffer replacement strategy.

(ii). Pinned blocks:

- If user wants to recover any DB system from the crashes, it is essential to restrict the time when a block is written back to the disk.
- most recovery systems do not allow the blocks to be written on the disk if the block updation being in progress. such type of blocks that are not allowed to be written on the disk are known as pinned blocks. many OS don't support pinned blocks

(iii) Forced o/p of blocks:

- In some cases, it becomes necessary to write the block back to disk even though the space occupied by block in the buffer is not required. When such type of write is required. It is known as forced o/p of a block.
- It is because sometimes the data stored on the buffer may get lost in some system crashes, but the data stored on disk usually doesn't get affected due to any disk crash.

* Failure with loss of non volatile storage:

- the basic measure is to dump the entire contents of the DB to stable storage periodically.
- one approach to dump the DB requires that no transaction is active during the dumping procedure & uses a procedure similar to checkpointing
  (i) o/p of all the log records currently present in the main memory into the stable storage.
  (ii) o/p all the buffer blocks into the disk.
  (iii) copy all the data present in the DB to the stable storage.
  (iv) o/p a log record <dump> into the stable storage.
- To recover from the loss of non volatile memory, we restore the DB from the archive & all the transactions that have been committed since the most recent dump are redone. this is known as archival dump. (Dump of DB contents)
- Dumps of DB & checkpointing are very similar

* ARIES Recovery method: (Algorithm for Recovery & Isolation Exploiting Semantics)

- ARIES is based on the Write Ahead log (WAL) protocol.
- Every update operation writes a log record which is one of the following:

(i) Undo-only log record:
only the before image is logged. So, undo operation can be done to retrieve the old data.

(ii). Redo-only log record:
only after image is logged. So, redo can be attempted.

(iii). Undo-redo log record:
Both before & after images are logged.

- Every log record is assigned a unique log sequence number (LSN).
- Every data page has a page LSN field that is set to LSN of the log record corresponding to the last update on the page.
- WAL requires that the log record corresponding to an update make it to stable storage before the data page corresponding to that update is written to disk.
- For performance reasons, each log write is not immediately forced to disk.
- A log tail is maintained in main memory to buffer log writes. The log tail is flushed to disk when it gets full.
- A transaction can't be declared committed until the commit log record makes it to disk.
- Once in a while the recovery sub system writes a checkpoint record to log.
- The checkpoint record contains the transaction table & dirty page table.
- A master log record is maintained seperately. In stable storage, to store the LSN of latest checkpoint record that made it to disk.
- On restart, the recovery subsystem reads the master log record to find checkpoints LSN, starts recov

The recovery process consists of 3 phases:

(i) Analysis:

The recovery subsystem determines the earliest log record from which the next pass must start. It also scans the log forward from the checkpoint record to construct a snapshot of what the system looked like at the instant of the crash.

(ii) Redo:

Starting at the earliest LSN, the log is read forward & each update redone.

(iii) Undo:

The log is scanned backward & updates corresponding to loser transactions are undone.

- ARIES is a no force type of backup approach ie, if transaction completes updated values are noted update in the disk at that time, it takes some time.
- Recovery mgr called as time of crash.
- In analysis phase identify the dirty pages (not updated or committed) in the buffer & set active transaction at the time of failure

(P_1, P_3, P_5) are active so needs to restart
T_1, T_4 & T_3

T_1 — write P_5
T_2 — write P_3
T_2 - commit
T_3 - write P_1
T_4 - write P_3
crash

- In Redo, find all operations done by DBMS before crash & restore the system back to the same state before crash

It aborts all active transactions that are still running. T_1 & T_3 write operations written again in redo.

- In undo, scans the log backward of active transaction in reverse

T_4 with write P_3 undo

T_3 " " P_1 "

T_1 " " P_5 "

Advantages:

- Simple & flexible
- Supports concurrency control protocols
- We can recover the pages independently.

* Remote backup systems:

" " " provide a wide range of availability, allowing the transaction processing to continue even if the primary site is destroyed by fire, flood or earthquake

- Data & log records from primary site are continuously backed up into a remote backup site.
- & the remote site is also called as "secondary site"
- Remote backup provides security in case primary gets destroyed
- " " can be offline or online or real-time. If it is offline, It is maintained manually.
- online backup systems are more real-time & lifesavers for DB admins Here, every bit of real-time data is backed up simultaneously at two distant places. one is directly connected to system. other one kept as backup at remote place.
- while designing a remote backup system, following are important

i) Detection of failure: It is important to detect when primary failed.

(ii). Transfer of control: When primary site fails, backup site takes over the processing & becomes the new primary system.

(iii). Time to recover:

If the log at remote backup becomes large, recovery will take long time.

(iv). Time to commit:

To ensure that the updates of committed transaction are durable, a transaction should not be announced committed until its log records have reached the backup site.