6/3/21.

ASSIGNMENT-2

DBMS

V. Greeshma
CSE-4
19131A05P9

1. Explain primary, secondary, cluster and multilevel indexes?

Ans. **Primary Index :-**

- If the index is created on the basis of the primary key of the table, then it is known as primary indexing.
  - Main file is sorted.
  - Primary key is used as search key.
- No. of entries in the index file = No. of blocks acquired by main file.
- Access time to access a random record if the index file need n blocks = $\text{ceil}(\log_2(n)) + 1$.
- As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.

Secondary Indexing :-

- Main file is unsorted
  - Sparse indexing not possible.
- The search key may or may not be key attribute.
- No. of entries in the index file = no. of blocks acquired by main file.
- No. of block access (or) block access time $\geq \lceil \log_2 n \rceil + 1$

Cluster Indexing :-

- Main file is sorted based on some non-key attribute.
- No. of records in the index table is equal to no. of unique values in the field of main file to which we want to make search key.
- There will be one entry for each unique value of the non-key attribute.
- If the no. of blocks required by Index file is n, then Block access required will be $\geq \lceil \log_2 n \rceil + 1$.

Multilevel Index :-

- Creating index for a index file.
- If index does not fit in memory, access becomes expensive.

- Solution : treat index kept on disk as a sequential file and construct a sparse index on it.
  - outer index – a sparse index of the basic index.
  - inner index – the basic index file.
- If even outer index is too large to fit in main memory, yet another level of index can be created and so on.
- Indices at all levels must be updated on insertion or deletion from the file.

2. Differentiate following :

Ans. 1)

Dense Index	Sparse Index
* In the dense index, there is an index record for every search key value in the database.	* In the sparse index, index records are not created for every search key.
* Index records contain search key value and a pointer to the actual record on the disk.	* An index record here contains a search key and an actual pointer to the data on the disk. We start at that record pointed to by index record and proceed along pointers sequentially until we find the desired record.

2)

Spanned Mapping	Unspanned Mapping.
* Records are stored across the blocks if they do not fit in one block.	* Records are not stored across the blocks if they do not fit in one block.
* We have to access more than one block to access a record.	* No need to access more than one block to access a record.
* We are saving the space.	* Not saving space but saves time.

3)

Ordered File Organization	Unordered File Organisation.
* In sequential (ordered) files, records are ordered by the value of a specified field.	* In heap (unordered) files, records are placed on disk in no particular order.
* Since the records are ordered, binary search can be performed to access a record.	* Since there is no particular ordering, a linear search must be performed to access a record.
* Faster access.	* Relatively slower access.

3. Explain hash function? Explain static and dynamic hashing?

Ans. **Hash function:-**

In Hashing technique, data is stored at the data blocks whose address is generated by using hash function. Most of the time, the hash function uses the primary key to generate address of the data block. A hash function is a simple mathematical function to any complex mathematical function.

Static Hashing:-

- In static hashing, the resultant data bucket address will always be the same.
- If we generate an address for EMP_ID = 103 using the hash function mod(5) then it will always result in same bucket address 3. Here, there will be no change in the bucket address.
- Hence, in this static hashing, the number of data buckets in memory remains constant throughout.

Operations of Static Hashing:

1. Searching — When a record needs to be searched, then the same hash function retrieves the address of the bucket where the data is stored.
2. Insert — When a new record is inserted into the table, then we will generate an address for a new record based on the hash key and record is stored in that location.
3. Delete — To delete a record, we will first fetch the record which is supposed to be deleted. Then we will delete the records for that address in memory.

4. Update — To update a record, we will first search it using a hash function, and then the data record is updated.

If we want to insert some new record into the file but the address of a data bucket generated by hash function is not empty, then it is known as 'bucket overflow'. To overcome this, there are various methods like Open Hashing and Closed Hashing.

Dynamic Hashing :-

- The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
- In this method, data buckets grow or shrink as the records increases or decreases. This method is also know as Extendable hashing method.

Searching a key :

(i) First, calculate the hash address of the key.
(ii) Check how many bits are used in the directory and these bits are called as i.
(iii) Take the least significant i bits of the hash address. This gives an index of the directory.
(iv) Now, using the index, go to the directory and find bucket address where the record might be.

Inserting a new record :

(i) Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
(ii) If there is still space in that bucket, then place the record in it.
(iii) If the bucket is full, then we will split the bucket and redistribute the records.

- A Bucket will have more than one pointers, pointing to if its local depth is less than global depth.
- When overflow condition occurs in a bucket, all the entries in the bucket are rehashed with a new local depth.
- If Local Depth of the overflowing bucket is equal to the global depth, only then the directories are doubled and the global depth is incremented by 1.
- The size of a bucket cannot be changed after the data insertion process begins.

4. Explain log-based recovery and ARIES recovery algorithm?

Ans- **Log-Based Recovery:-**

- The 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 from there.
- If any operation is performed on the database, 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 database.

ex: Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

- When the transaction is initiated, then it writes 'start' log.

  $< T_n, Start >$

- When the transaction modifies the city from 'Noida' to 'Bangalore', then another log is written to the file.

  $< T_n, City, 'Noida', 'Bangalore' >$

- When the transaction is finished, then it writes another log to indicate the end of the transaction.

  $< T_n, Commit >$

There are two approaches to modify the database:

1. **Deferred database modification:**
   - The deferred modification technique occurs if the transaction does not modify the database until it has committed.
   - In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.
2. **Immediate database modification:**
   - The Immediate modification technique occurs if database modification occurs while the transaction is still active.
   - In this technique, the database is modified immediately after every operation. It follows an actual database modification.

Recovery using Log records:-

When the system is crashed, then the system consults the log to find which transactions need to be undone and which need to be redone.

1. If the log contains the record $< T_i, \text{Start} >$ and $< T_i, \text{Commit} >$ or $< T_i, \text{Commit} >$, then the Transaction T_i needs to be redone.
2. If log contains record $< T_n, \text{Start} >$ but does not contain the record either $< T_i, \text{commit} >$ or $< T_i, \text{abort} >$, then the Transaction T_i needs to be undone.

ARIES Recovery Algorithm:-

Algorithm for Recovery and 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:

1. Undo-only log record:
Only the before image is logged. Thus, an undo operation can be done to retrieve the old data.
2. Redo-only log record:
Only the after image is logged. Thus, a redo operation can be attempted.
3. Undo-redo log record:
Both before images and after images are logged.

In it, every log record is assigned a unique and monotonically increasing log sequence number (LSN). Every data page has a page LSN field that is set to the 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 cannot be declared committed until the commit log record makes it to disk.

Once in a while the recovery subsystem writes a checkpoint record to the log. The checkpoint record contains the transaction table and the dirty page table. A master log record is maintained seperately, in stable storage, to store the LSN of the latest checkpoint record that made it to disk. On restart,

the recovery subsystem reads the master log record to find the checkpoint's LSN, reads the checkpoint record, and starts recovery from there on.

The recovery process actually consists of 3 phases:

1. **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.

2. **Redo:**
Starting at the earliest LSN, the log is read forward and each update redone.

3. **Undo:**
The log is scanned backward and updates corresponding to loser transactions are undone.

---

5. Explain different RAID levels?

Ans. RAID refers to redundancy array of the independent disk. It is a technology which is used to connect multiple secondary storage devices for increased performance, data redundancy or both. It gives you the ability to survive one or more drive failure depending upon the RAID level used. It consists of an array of disks in which multiple disks are connected to achieve different goals.

Standard RAID levels :-

→ **RAID 0 :-**

- RAID level 0 provides data stripping, i.e., a data can place across multiple disks. It is based on stripping that means if one disk fails then all data in the array is lost.
- This level doesn't provide fault tolerance but increases the system performance.
- In this level, instead of placing just one block into a disk at a time, we can work with two or more blocks placed it into a disk before moving on to the next one.
- When there is no duplication of data, a block once lost cannot be recovered.

→ **RAID 1 :-**

- This level is called mirroring of data as it copies the data from drive 1 to drive 2. It provides 100% redundancy in case of a failure.
- Only half space of the drive is used to store the data. The other half of drive is just a mirror to the already stored data.

→ **RAID 2 :-**

- RAID 2 consists of bit-level striping using hamming code parity. In this level, each data bit in a word is recorded on a separate disk and ECC code of data words is stored on different set disks.
- Due to its high cost and complex structure, this level is not commercially used. This same performance can be achieved by RAID 3 at a lower cost.

→ **RAID 3 :-**

- RAID 3 consists of byte-level striping with dedicated parity. In this level, the parity information is stored for each disk section and written to a dedicated parity drive.
- In case of drive failure, the parity drive is accessed, and data is reconstructed from the remaining devices. Once the failed drive is replaced, the missing data can be restored on the new drive.
- In this level, data can be transferred in bulk. Thus high-speed data transmission is possible.

→ **RAID 4 :-**

- RAID 4 consists of block-level stripping with a parity disk. Instead of duplicating data, the RAID 4 adopts a parity-based approach.
- This level allows recovery of atmost 1 disk failure due to the way parity works. In this level, if more than one disk fails, then there is no way to recover the data.
- Level 3 and level 4 both are required at least three disks to implement RAID.

→ **RAID 5 :-**

- RAID 5 is a slight modification of the RAID 4 system. The only difference is that in RAID 5, the parity rotates among the drives.

- It consists of block-level striping with DISTRIBUTED parity.
- Same as RAID 4, this level allows recovery of atmost 1 disk failure. If more than one disk fails, then there is no way for data recovery.
- This level was introduced to make the random write performance better.

→ **RAID 6 :-**

- This level is an extension of RAID 5. It contains block-level stripping with 2 parity bits.
- In RAID 6, you can survive 2 concurrent disk failures. Suppose you are using RAID 5 and RAID 1. When your disks fail, you need to replace the failed disk because if simultaneously another disk fails then you won't be able to recover any of the data, so in this case RAID 6 plays its part where you can survive two concurrent disk failures before you run out of options.

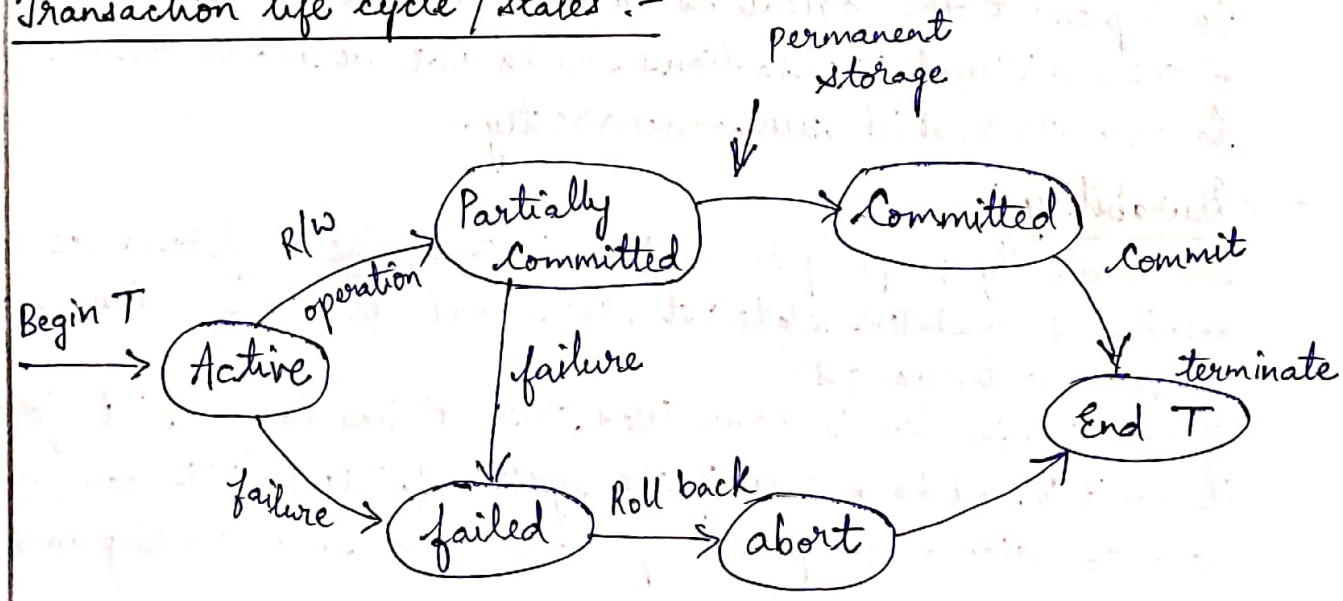
---

6. Explain Transaction and Transaction life cycle and ACID Properties of Transaction?

Ans- **Transaction :-**

Transaction is a sequence of operations / instructions.
"Transaction is a single logical unit of work which access database & possibly modify the database".

Transaction life cycle / states :-



Active state — It is the first state of every transaction. In this state, the transaction is being executed.

Partially committed — Here, a transaction executes its final operation, but the data is still not saved to the database.

Committed — A transaction is said to be in a committed state, if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state — If any of the checks made by the database recovery system fails, then the transaction is said to be in failed state.

Aborted — If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.

ACID Properties :-

→ Atomicity :-

It states transaction has to be executed completely or not at all. No partial execution. If there is partial execution then roll back.

→ Consistency :-

This property states that every transaction sees a consistent database instance. The transaction is used to transform the database from one consistent state to another consistent state.

→ Isolation :-

To implement the logical isolation, the intermediate results of the individual transactions should not be made available to each other which run concurrently.

→ Durability :-

The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.

If the transaction is committed, then it has to persist forever, it can't roll back even if the system fails. In this case, we need to write a compensating transaction, since rolling back is not possible. This is durability.

7. Explain advantage of concurrent execution of transaction and dirty read, lost update, phantom read, unrepeatable read problems due to concurrent execution?

Ans- Non-Serial / Concurrent execution always has overlapped execution of transaction.

Advantages :-

(1) Response time reduces. (Response time is the first time when we get resources/CPU).

(2) Average waiting time reduces. (Waiting time while waiting for resources/CPU).

(3) Resource utilisation increases.

(4) Efficiency (η) increases.

Problems :-

(1) Dirty read problem.
   ↳ Reading uncommitted data.
   solution → do not read uncommitted data.
   → if you read then do not commit.

(2) Unrepeatable Read problem.
   ↳ Occurs when two or more read operations of the same transaction read different values of the same variable.
   i.e it occurs because of isolation break.
   solution → do not read repeatedly (or roll back).

(3) Phantom read problem.
   ↳ trying to read something which does not exist.

(4) Lost update problem.
   ↳ Update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction. (trying to access lost updates).

8. What is Serializability ? Explain conflict and View serializability ?

Ans- **Serializability :-**

Serializability is a concept that helps us to check which schedules are serializable. A serializable schedule is the one that always leaves the database in consistent state.

Conflict Serializability :-

- Conflict Serializability is one of the type of Serializability, which can be used to check whether a non-serial schedule is conflict serializable or not.
- A schedule is called conflict serializable if we can convert it into a serial schedule after swapping its non-conflicting operations.
- Two operations are said to be in conflict, if they satisfy all the following three conditions :

1. Both the operations should belong to different transactions.
2. Both the operations are working on same data item.
3. At least one of the operation is a write operation.

ex: W(X) of T1 and R(X) of T2.
W(X) of T1 and W(X) of T2.

Example :

T1	T2
R(A)	
	R(A)
	R(B)
	W(B)
R(B)	
W(A)	

Lets swap non conflicting operations:
After swapping R(A) of T1 and R(A) of T2 we get :

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

After swapping R(A) of T1 and R(B) of T2 we get :

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

After swapping R(A) of T1 and W(B) of T2 we get :

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

We finally got a serial schedule after swapping all the non-conflicting operations so we can say that the given schedule is Conflict Serializable.

View Serializability :-

- View Serializability is a process to find out that a given schedule is view serializable or not.
- If we can prove that the given schedule is View Equivalent to its serial schedule then the given schedule is called View Serializable.
- Two schedules T1 and T2 are said to be view equivalent, if they satisfy all the following conditions:

1. Initial Read : Initial read of each data item in transactions must match in both schedules. For example, if transaction T1 reads a data item X before transaction T2 in schedule S1 then in schedule S2, T1 should read X before T2.
2. Final Write : Final write operations on each data item must match in both the schedules. For example, a data item X is last written by Transaction T1 in schedule S1 then in S2, the last write operation on X should be performed by the transaction T1.
3. Update Read : If in schedule S1, the transaction T1 is reading a data item updated by T2 then in schedule S2, T1 should read the value after the write operation of T2 on same data item. For example, in schedule S1, T1 performs a read operation on X after the write operation on X by T2 then in S2, T1 should read the X after T2 performs write on X.

Example:

T_1	T_2	T_3	T_4
	R(A)		
R(A)			
W(C)			
		R(C)	
W(B)			
			R(B)
		W(A)	
			R(C)
	W(D)		
	R(B)		
			W(A)
			W(B)

	A	B	C	D
IR	$T_1 T_2$	X	X	X
FW	T_4	T_4	T_1	T_2
U.R	X $T_1 \rightarrow T_4$ $T_2 \rightarrow T_4$	$T_1 \rightarrow T_4$ $T_1 \rightarrow T_2$ $T_2 \rightarrow T_4$	$T_1 \rightarrow T_4$ $T_1 \rightarrow T_3$	X

$\therefore T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$

Hence it is View Serializable.

9. Explain the following:

Ans- (A) **Recoverability :-**

Schedules in which transactions commit only after all transactions whose changes they read commit are called recoverable schedules. In other words, if some transaction T_j is reading value updated or written by some other transaction T_i, then the commit of T_j must occur after the commit of T_i.
Transactions must be committed in order.
Dirty Read problem and Lost Update problem may occur.

(B) **Cascadlessness :-**

Dirty Read not allowed, means reading the data written by an uncommitted transaction is not allowed. Lost update problem may occur. Schedules in which transactions read values only after all transactions whose changes they are going to read commit are called cascadeless schedules.

(C) **Strict schedule :-**

Strict schedule doesn't allow dirty read and dirty write operations.
Solution: Read and write only committed data.
Otherwise lost update problem occurs.

(D) **Thomas Write Rule :-**

Thomas Write Rule states, "if there is any obsolete write before/after the blind write then we ignore the obsolete write and consider only the blind write".
The basic Thomas write rules are as follows:

- If $TS(T) < R\_TS(X)$ then transaction T is aborted and rolled back, and operation is rejected.
- If $TS(T) < W\_TS(X)$ then don't execute the W_item(X) operation of the transaction and continue processing.
- If neither condition 1 nor condition 2 occurs, then allowed to execute the WRITE operation by transaction T_i and set $W\_TS(X)$ to $TS(T)$.

10. Explain all deadlock handling techniques in DBMS ?

Ans- In a database, a deadlock is an unwanted situation in which two or more transactions are waiting indefinitely for one another to give up locks.

→ **Deadlock Avoidance :-**

- When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restarting the database. This is a waste of time and resource.
- Deadlock avoidance mechanism is used to detect any deadlock situation in advance.

→ **Deadlock Detection :-**

In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

Wait for Graph

- In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.
- The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps changing the graph if there is any cycle in the graph.

→ **Deadlock Prevention :-**

- Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.
- The DBMS analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed that transaction to be executed.

Wait - Die Scheme

In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.

Let's assume there are two transactions T_i and T_j and let $TS(T)$ is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS:

1. Check if $TS(T_i) < TS(T_j)$ — If T_i is the older transaction and T_j has held some resource, then T_i is allowed to wait until the data item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.
2. Check if $TS(T_i) < TS(T_j)$ — If T_i is older transaction and has held some resource and if T_j is waiting for it, then T_j is killed and restarted later with the random delay but with the same timestamp.

Wound Wait Scheme

- In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After the minute delay, the younger transaction is restarted but with the same timestamp.
- If the older transaction has held a resource which is requested by the younger transaction, then the younger transaction is asked to wait until older releases it.

---

11. Explain all versions of 2PL ?

Ans- 2PL (Two Phase Locking):-

a) Basic 2PL :

- growing phase (lock can be granted but not released)
- shrinking phase (lock cannot be granted but can be released).

- The moment a transaction takes the last lock (or) the moment it switches from growing phase to shrinking phase, there will be lock point.
- Order of lock point is equal to the order of serializability.

- It will be conflict serializable, view serializable, may or may not be recoverable, deadlock and starvation will be there and no cascadelessness.

b) **Conservative 2PL (or) Static 2PL :-**

It says "take all the locks before performing any operation".

- Advantage is that there will be no deadlock when we are not able to start. There can be starvation.
- Hence Conservative 2PL solves the problem of deadlock i.e. it is free from deadlock, because deadlock always occurs in execution and here execution doesn't start if the locks are not provided first & hence if there's no start — no deadlock.
- There is problem of Recoverability and cascadelessness.

c) **Rigorous 2PL :-**

It says "cannot release lock until we decide to commit".

- Hence this solves the problem of recoverability and cascadelessness. It is also conflict serializable and view serializable.
- But it is not deadlock free.
- It reduces the order of concurrency. This is the major problem.

d) **Strict 2PL :-**

It says "if you have a exclusive lock, you can release it only after committing but if there is a shared lock, then you can release it anytime".

- It is conflict serializable, view serializable, recoverability and cascadelessness is there, deadlock is there. It reduces the order of concurrency but it is better than rigorous 2PL.