Unit V

Chapter - 12 : Transaction Management

Q.1 What is transaction. SPPU- Oct.19, 6 Marks

Ans. : Transaction

1. Introduction

- Single SQL command is sent to database server as a query and server will reply with answer.
- Multiple SQL commands (DML,DRL etc.) are sent to database server which executed one after other (as shown in Fig. 12.1)

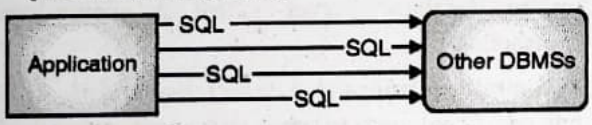

Fig. 12.1 : Executing Single Operation in DBMS

- In place of sending one by one SQL command to server we can combine multiple operations that are logically similar and send to server as a single logical unit called **transaction**.

Example :

Transferring Rs.100 from one account to other

1. Withdraw Rs.100 from account_1
2. Deposit Rs.100 to account_2

- Simple query fired on DBMS is called **SQL operation.**
- Collection of multiple operations that forms a single logical unit is called as **transaction.**
- A transaction is a sequence of one or more SQL statements that combined together to form a single logical unit of work.

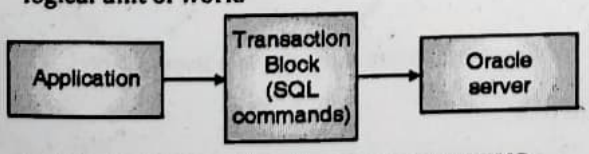

Fig. 12.2 : Executing Transaction in DBMS

- Types of operations that can be done inside transaction

(a) Read operation :

- Read operation transfers data item from (e.g. table) the database memory to a local buffer of the transaction that executed the read operation.

Example :

Data Selection / Retrieval Language

```
SELECT  *
FROM    Students;
```

(b) Write operation :

- Write operation transfer data from local memory to database.
- Write operation transfers one data item from the local buffer of the transaction that executed the write back to the database.

Example :

Data manipulation language (DML)

```
UPDATE Student
SET    Name = 'Bhavna'
Where  Sid  = 1186;
```

- Information processing in DBMS divides operation individual, indivisible operational logical units, called transactions.
- A transaction is a sequence of small database operations.
- Transactions will execute to complete all set of operations successfully.

Example :

During the transfer of money between two bank accounts it is problematic if one of transaction fails.

```
BEGIN TRANSACTION transfer
    UPDATE accounts
    SET balance=balance – 100
    WHERE account=A

    UPDATE accounts
    SET balance=balance + 100
    WHERE account=B
if no errors then
    Commit Transaction
else
    Rollback Transaction
End If
END TRANSACTION transfer
```

Fig. 12.3 : Sample Implementation of Transaction in SQL

2. Transaction structure

- The transaction consists of all SQL operations executed between the begin transaction and end transaction.
- A transaction starts with a BEGIN transaction command.
- BEGIN command instructs transaction monitor to starts monitoring the transaction status.
- All operations done after a BEGIN command is treated as a single large operation.

3. Transaction boundaries

- Transaction must ends either by executing a COMMIT or ROLLBACK command.
- Until a transaction commits or rolls back the data in database remains unchanged.

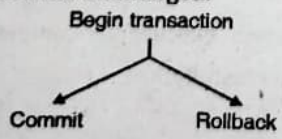

Fig. 12.4

(a) Commit transaction

All successful transactions are required to be committed by issuing commit command. To make all changes permanent and available to other users of the database.

(b) Rollback transaction

- If transaction is unsuccessful due to some error then it must be rolled back.
- Roll back command will remove all changes to the database which are undone and the database remains unchanged by effect of transaction.
- The DBMS should take care of transaction it should be either complete or fail.
- When a transaction fails all its operations and the database is returned to the original state it was in before the transaction started.

Q.2 What are the possible causes of transaction failure ?

SPPU - May 19. 4 Marks

Ans. : Transaction Failure :

- If transaction complete successfully it may be saved on to database server called as **committed** transaction.
- A committed transaction transfers database from one consistent state to other consistent state, which must persist even if there is a system failure.
- Transaction may not always complete its execution successfully. Such a transaction must be **aborted.**
- An aborted transaction must not have any change that the aborted transaction made to the database. Such changes must be undone or **rolled back.**
- Once a transaction is committed, we cannot undo its effects by aborting it.

(A) Transaction States

A transaction must be in one of the following states :

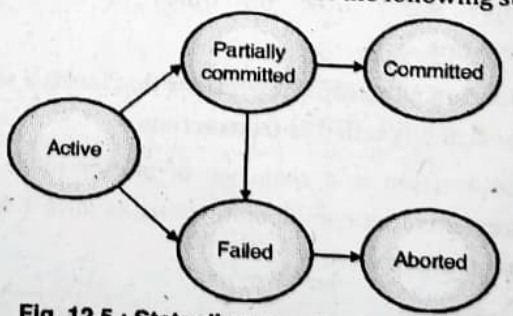

Fig. 12.5 : State diagram of a transaction

(a) Active

- This is initial state of transaction execution.
- As soon as transaction execution starts it is in active state.
- Transaction remains in this state till transaction finishes.

(b) Partially committed

- As soon as last operation in transaction is executed transaction goes to partially committed state.
- At this condition the transaction has completed its execution and ready to commit on database server. But, it is still possible that it may be aborted, as the actual output may be there in main memory, and thus a hardware failure may prohibit its successful completion.

(c) Failed

A transaction enters the failed state after the system determines that the transaction can no longer proceed with its normal execution.

Example : In case of Hardware or logical errors occurs while execution.

(d) Aborted

- Failed transaction must be rolled back. Then it enters the aborted state.
- Transaction has been rolled back restoring into prior state.
- In this stage system have two options :

(1) **Restart the transaction :** A restarted transaction is considered to be a new transaction which may recover from possible failure.

(2) **Kill the transaction :** Because the bad input or because the desired data were not present in the database an error can occurs. In this case we can kill transaction to recover from failure.

(e) Committed

- When last data item is written out, the transaction enters into the committed state.
- This state occurs after successful completion of transaction. A transaction is said to have terminated if has either committed or aborted.

Q.3 Define conflict serializable schedule with example. SPPU - Oct.19

Ans. : Conflict Serializability

1. Introduction

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

2. Conflict

- A pair of consecutive database actions (reads, writes) is in conflict if changing their order would change the result of at least one of the transactions.

		Transaction T_j	
		Read(D)	Write(D)
Transaction T_i	Read(D)	No Conflict	Conflict
	Write(D)	Conflict	Conflict

- Consider schedule S has two consecutive instructions I_i and I_j from transactions T_i and T_j respectively.
- If I_i and I_j access to different data items then they will not conflict and can be swapped, without any problem.
- If I_i and I_j access to same data item D then consider following consequences:
  - I_i = READ (D), I_j = READ (D) then **no conflict** as they only read value.
  - This operation is called as non conflicting swap.
  - I_i = READ (D), I_j = WRITE (D) then they **conflict** and can not be swapped.
  - I_i = WRITE (D), I_j = READ (D) then they **conflict** and can not be swapped.
  - I_i = WRITE (D), I_j = WRITE (D) then they **conflict** and cannot be swapped.
- So we can say that instructions conflict if both consecutive instructions operate on same data item and from different transactions and one of them is WRITE operation.
- If I_i and I_j **access to different data item D** then consider following all consequences **no conflict** as they only read or writing different values.
- I_i = READ (D)/WRITE (D), I_j = READ (P) /WRITE (P) then **no conflict** as they only reading or writing different data.
- The following set of actions is conflicting :
  T_1:R(X), T_2:W(X), T_3:W(X)
- While the following sets of actions are not :
  T_1:R(X), T_2:R(X), T_3:R(X)
  T_1:R(X), T_2:W(Y), T_3:R(X)

3. Conflict equivalence

Two schedules are conflict equivalent if they can be turned into one another by a sequence of non conflicting swaps of adjacent actions.

4. Example :

- A schedule is conflict serializable if it is conflict equivalent to a serial schedule.

T_1	T_2
Read(P)	
Write(P)	
	Read(P)
Read(Q)	
	Write(P)
Write(Q)	
	Read(Q)
	Write(Q)

Fig. 12.6 : Schedule S

- Now the schedule S after performing swapping can transformed into schedule R as shown above which also results in same values of P and Q.
- The above schedule is same as serial schedule $<T_1, T_2>$.
- If a schedule S can be transformed into a schedule R by a series of swap operations on non conflicting instructions, then we can say **schedule S and R are conflict equivalent.**
- If a concurrent schedule is conflict equivalent to a serial schedule of same transactions then it is conflict Serializable. So schedule **S is conflict serializable** to serial schedule $<T_1, T_2>$.

Q.4 When are two schedules said to be view equivalent? SPPU - May 18, Dec. 18, 4 Marks

Ans. : View Serializability

1. Introduction

View equivalence is less strict than conflict equivalence, but it is like conflict equivalence based on only the read and write operations of transactions.

2. Conditions for view equivalence

Let, D = Data item

S_1, S_2 = Transaction schedules

T_i, T_j = Database transaction

- Schedules S_1 and S_2 are view equivalent if they satisfy following conditions for each data item D,
  - (a) S_1 and S_2must have same transactions included and also they are performing same operations on same data. If T_i reads initial value of D in S_1, then T_i also reads initial value of D in S_2.
  - (b) If T_i reads value of D written by T_j in S_1, then T_i also reads value of D written by T_j in S_2.
  - (c) If T_i writes final value of D in S_1, then T_i also writes final value of D in S_2.
- First 2 conditions ensure that transaction reads same value in both schedules.
- Condition 3 ensures that final consistent state.
- If a concurrent schedule is view equivalent to a serial schedule of same transactions then it is **view serializable.**
- Consider following schedule S_1 with concurrent transactions $<T_1, T_2, T_3>$.
- In both the schedules S_1 and a serial schedule $S_2<T_1, T_2, T_3>$ T_1 reads initial value of D. Transaction T_3 writes final value of D. So schedule S_1 satisfies all three conditions and is view serializable to $<T_1, T_2, T_3>$.

3. Example :

Below two schedules S and R are view equivalent,

Schedule S		
T_1	T_2	T_3
Read(P)		
	Write (P)	
Write (P)		
		Write (P)
Schedule R		
T_1	T_2	T_3
Read(P)		
Write (P)		
	Write (P)	
		Write (P)

4. Note

- Every conflict serializable schedule is view serializable but not vice versa.
- In above example T_2 and T_3Writes data without reading value of data item by themselves so they are called as "Blind Writes".

Chapter - 13 : Concurrency Control

Q.1 Explain two phase locking protocol & its forms.
SPPU - Dec. 19 · Oct. 19, 4 Marks

Ans. : Two-Phase Locking (2PL) Protocol

(1) Introduction

- Two-Phase Locking (2PL) synchronizes read and write by explicitly detecting and preventing conflicts between concurrent operations.
- Before reading data item X, a transaction must "own" a read lock on X. Before writing into X, transaction must "own" a write lock on X.
- The ownership of locks is governed by two rules :
  (a) Different transactions cannot simultaneously own conflicting locks (i.e. WR).
  (b) Once a transaction surrenders ownership of a lock, it may never obtain additional locks.
- The definition of conflicting lock depends on the type of synchronization being performed:(Refer lock compatibility matrix)
- For 'RW' synchronization two locks conflict if
  Both are locks on the same data item.
  One is a read lock and the other is a write lock.
- For 'WW' synchronization two locks conflict if.
  Both are locks on the same data item.
  Both are write locks.
- The second lock ownership rule causes every transaction to obtain locks in a two phase manner.

Growing phase

In this phase, transaction may obtain n number of new locks but may not release any lock.

Shrinking phase

Transaction may release locks but cannot obtain any new Lock.

(2) Working of 2P protocols

- Initially transaction is in 'growing phase', it acquires locks as needed. Once it starts releasing locks it enters into the 'Shrinking Phase' and now it may not acquire any lock after shrinking phase starts.
- The point at which transaction obtains final (last) lock is called as lock point of transaction.

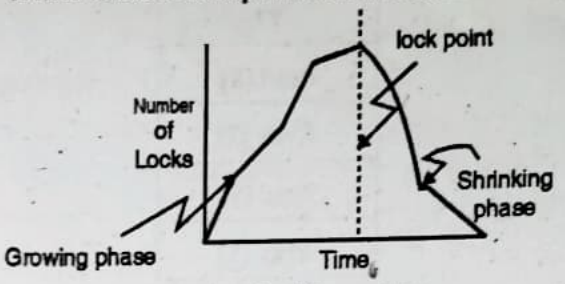

Fig. 13.1 : Locking point

- Lock point is end of growing phase for that transaction and start of shrinking phase.
- The schedule to be serializable, transactions should be arranged according to their lock points.
- When the transaction terminates (or aborts), all remaining locks are automatically released. Some systems also require that transactions hold all locks until termination.

(3) Advantages

- Two - phase locking protocol ensures conflict serializability.
- Two-phase locking protocol is simple to implement and understand.

(4) Drawbacks

- Deadlock may occur in two phase locked schedule.
- Cascaded rollback may occur under two phase locking. Cascaded rollbacks do mean by if one transaction is rolling back all transactions depends on this transaction will be rolled back.

Example :

(A) No 2P (B) with 2P locking.

Q.2 What is deadlock? Explain how deadlock detection and prevention is done.
SPPU - Dec. 17, Dec. 19, 8 Marks

Ans. : Concept of Deadlock

1. A system is said to be in a state of deadlock if there exists a set of transactions such that every transaction in the set is waiting for another transaction to complete its execution.

2. Example

Consider two transactions given below :

T1 : This transaction first read and then write data on data item X, then after that performs read and write on data item Y.

T1
Read (X)
Write (X)
Read (Y)
Write (Y)

T2 : This transaction first read and then write data on data item Y, then after that performs read and write on data item X.

T2
Read (Y)
Write (Y)
Read (X)
Write (X)

Consider above two transactions are executing using locking protocol as below :

T1	T2	
Lock-X(X)		
Read (X)		
Write (X)		
	Lock-X(Y)	
	Read (Y)	
	Write (Y)	
Lock-X(Y)		Wait for transaction T2 to Unlock Y
Read (Y)		
Write (Y)		
	Lock-X(X)	Wait for transaction T1 to Unlock X
	Read (X)	
	Write (X)	

So in above schedule consider two transactions given below transaction T1 is waiting for transaction T2 to Unlock data item Y and transaction T2 is waiting for transaction T1 to Unlock data item X.

So system is in a deadlock state as there are set of transactions T1and T2 such that, both are waiting for each other transaction to complete. This state is called as deadlock state.

3. There are two principle methods to handle deadlock in system,

(a) Deadlock prevention

We can use deadlock prevention techniques to ensure that the system will never enter a deadlock state.

(b) Deadlock detection and recovery

We can allow the system to enter a deadlock state and then we can detect such state by deadlock detection techniques and try to recover from deadlock state by using and deadlock recovery scheme.

4. Both of above methods may result in transaction rollback.
5. Deadlock Prevention is commonly used if the probability that the deadlock occurs in system is high else detection and recovery are more efficient if probability of deadlock occurrence is less.

Approaches for Deadlock Prevention

1. Approach 1

A simplest form, in which a transaction acquires lock on all data items which will be required by transaction at the start of execution. It is effective as other transactions can't hold lock on those data items till first unlocks data item.

Disadvantages

(i) It is difficult to know in advance which data items need to be locked.

(ii) Data utilization is very low as may be unused data items locked by transaction.

2. Approach 2

This approach uses timestamp ordering of data items. It is something like tree protocol and every transaction have to access data item in given sequence only. The variation of this approach with

two phase protocol assures deadlock prevention. Order of data items must be known to every transaction.

(i) To have concurrency control, two phase locking protocol is used which will ensure locks are requested in right order.

(ii) Timestamp ordering is determined by validation (Ti) i.e. TS (Ti) = validation (Ti) to achieve serializability.

(iii) The validation test for transaction Tj requires that for all transaction Ti with TS (Ti) < TS (Tj) then one of the following conditions must be hold.

(a) Finish (Ti) <start (Tj) as Ti finishes before Tj starts the serializability should be maintained.

(b) Ti completes its write phase before Tj starts its validation phase (Start (Tj) < finish (Ti) <Validation (Tj)).

(iv) This ensures writes of both transactions do not take place at same time.

(v) Read of Tj not affected by writes of Tj and Tj can't affect read of Ti so serializability is maintained.

3. Approach 3 : Prevention and transaction rollbacks

(i) Pre-emptive Technique

(ii) Pre-emption means, if transaction Ti wants to hold lock on data item hold by Tj. then system may preempt (UNLOCK all previous locks) Ti by rolling it back and granting lock to Tj on that data item.

(iii) To control Pre-Emption, Transaction can be assigned a unique timestamp.

(iv) System will use this timestamp to decide whether to wait or rollback transaction.

(v) The transaction keeps its old time stamp if it is rollback and restarted.

(vi) Various deadlock prevention techniques using timestamps,

1. Wait-Die

- This is Non pre-emptive technique of deadlock prevention.
- When transaction Ti wants to hold data item, currently hold by Tj, then Ti is allowed to wait if and only if it is older than Tj otherwise Ti is rolled back (die).
- If T1 requests for data item hold by T2 then as TS (T1) < TS (T2) so T1 should wait.

2. Wound-Wait

- This is Pre-emptive Technique of deadlock prevention.
- When Ti wants to hold data item, currently hold by Tj, then Ti is allowed to wait if and only if Ti is younger to Tj otherwise Tj is rollback (wounded).
- Consider T1, T2, T3 transactions.
- If T1 requests for data item hold by T2 then data item is pre-empted from T2 and given to T1 and T2 is rollback.
- If T3 requests for data item hold by T2 then T3 need to roll back.

(vii) Prevention may lead to starvation of transaction, if they are rollback again and again. But both schemes wait-die and wound-wait avoid starvation by making use of timestamps.

Deadlock Detection and Recovery

1. Introduction

(i) When system is having deadlock detection and recovery techniques, the detection is done periodically to check whether the system is in deadlock, if yes then the recovery techniques are used to resolve this deadlock.

(ii) This can be done with help of some deadlock detection algorithms.

(iii) To achieve this, system must do the following :

(a) System should maintain information about current data items allocation to transactions and requests to be satisfied.

(b) Algorithm that uses this information to detect deadlock.

(c) Recovery techniques to be applied after detection of deadlock.

2. Deadlock detection

(i) Deadlock can be detected using directed graph called as wait-for-graph.

(ii) The graph G = (V,E) can be seen as V is of vertices i.e. set of transaction in execution concurrently and E is set of edges.

(iii) Such that edge Ti→ Tj if Ti→ Tj is present in graph it shows that, transaction Tj is waiting for transaction Ti to release a data item it needs.

(iv) If cycle is present in wait-for-graph then deadlock is present and transactions in cycle are deadlock.

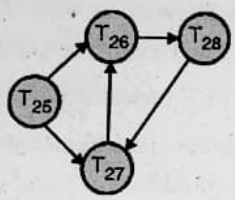

Fig. 13.1 : Wait-for graph with a cycle

(v) To detect deadlock, system must maintain wait-for-graph and periodically invoke an algorithm that searches cycle in the graph.

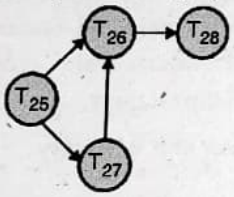

Fig. 13.2 : Wait-for graph with no cycle

(vi) If no cycle is present it mean no deadlock in system.

(vii) If deadlock occurs frequently, then detection algorithm should be invoked more frequently problem in this scenario is the data items locked by deadlock transaction will be unavailable until the deadlock is problem.

(viii) This may tend to more cycles in graph degrading capacity of granting lock requests.

3. Recovery from deadlock

- When deadlock is detected in the system, then system should be recovered from deadlock using recovery schemes.
- A most common solution is rollback one or more transaction to break the deadlock.
- Methods for recover from deadlock :

(a) Selection of victim

- If deadlock is detected then a transaction one or more transactions (victims) to be selected to break the deadlock.
- Transactions with minimum cost should be selected for rollbacks.
- Cost can be detected by following factors.
  - For how much time transaction has computed and how much time it needs to do.
  - How many data items it has locked ?
  - How many data items it may need ahead ?
  - Number of transaction to be rollback.

(b) Rollback

- Once victims are decided then there are two ways to do so.
- Total rollback : The transaction is aborted and then restart it.
- Partial rollback : Rollback the only transaction which is needed to break the deadlock.
- But this approach needs to record some more information like state of running transactions, locks on data item held by them, and deadlock detection algorithm specifies points up to which transaction to be rollback and recovery method has to rollback.
- After sometime transaction resume; partial transaction.

(c) Starvation

- It may happen every time same transaction is selected as victim and this may lead to starvation of that transaction (minimum cost transaction it selected every time).
- System should take care that every time same transaction should not be selected as victim. So it will not be starved.

Q.3 Explain time stamp based concurrency control.

SPPU - Dec. 17, Oct. 19, 4 Marks

Ans. : Timestamp Based Protocols

(1) Introduction

- To achieve serializability order of transactions for execution can be decided in advance using its time at which transaction entered in system.

- A general method to achieve this is using time stamp ordering protocol.

(2) Concept of Timestamp

- A fixed timestamp is assigned at start of execution of the transaction. Every transaction Tj has been assigned a timestamp by database system denoted as TS (Tj).
- A transaction which has entered in system recently will have greater timestamp. If transaction Tj starts after Ti then,

  TS (Ti) < TS (Tj)
- System clock is used as timestamp i.e. system time when transaction Tienters system or a logical counter can be used as timestamp and incremented after every assignment.
- If TS(Ti) < TS(Tj), then system must ensure that serializable schedule is equivalent to a serial schedule as < Ti, Tj>
- Every data item X is with two timestamp values :

(a) W-timestamp (X)

- It denotes the largest timestamp of any transaction that executed WRITE (X) successfully on given data item X.
- That mean it is timestamp of recent WRITE (X) operation.
- On execution of next WRITE (X) operation (O1), W-timestamps will be updated to new timestamp of O1 i.e. TS (O1).

(b) R-timestamp (X)

- Denotes the largest timestamp of any transaction that executed READ (X) successfully on data item (X).
- That mean it is timestamp of recent READ (X) operation.
- On execution of next READ (X) operation (O1), R-timestamps will be updated to new timestamp of O1 i.e. TS (O1).

(3) Timestamp - ordering Protocol

- This protocol ensures any conflicting READ or WRITE is executed in order of timestamp.
- If by any reasons, if transaction is aborted then on restarting, new timestamp is assigned.

Working

(a) For transaction Ti to execute READ (X) Operation,

If TS (Ti) < W - timestamp (X)

Then Ti is trying to read value of X that is overwritten by other transaction.

W - timestamp (X) = 149 (Recent Write)			
TS	**Ti**	**Tx**	**Operations**
148	READ (X)	...	TS (Ti) < W - timestamp (X) i.e. 148 < 149 (W-Timestamp)
149	Unlock (X)	WRITE (X)	Record W - timestamp (X) = 149

So this READ is rejected (as Tx is already performing write operation on data so no other operation can be performed by any other transaction) and Ti is rolled back.

If TS (Ti) ≥ W - timestamp (X)

Then READ executed and set

R-timestamp(X) = max {TS (Ti), R-timestamp}			
TS	Ti	Tx	Operations
148	....	WRITE (X)	Record W - timestamp (X) = 148 (recent write)
149	READ (X)	....	TS (Ti) ≥ W - timestamp (X) i.e. 149 ≥ 148 (W-Timestamp) Record R-timestamp(X) = 149
150	....	....	....
151	READ (X)	....	TS (T1) ≥ W - timestamp (X) i.e. 151 ≥ 148 (W-Timestamp) Record R-timestamp(X) = max{149,151} = 151

(b) If transaction Tiexecute WRITE (X) Operation,

If TS (Ti) < R - timestamp (X)

Then Ti has produced value of X which is not needed now so rollback Ti.

TS	Ti	Tx	Operations
148	....	...	...
149	WRITE(X)	....	TS (T1) < R - timestamp (X) i.e. 149 < 151 (R-Timestamp) Reject WRITE roll back Ti
150	....	....	....
151		READ (X)	Record R-timestamp(X) = 151 (Recent READ Operation)

If TS (Ti) < W - timestamp (X)

- Then Ti is trying to write obsolete value of X, So rollback Ti

TS	Ti	Tx	Operations
148	....	...	...
149	WRITE (X)	....	TS (T1) < W - timestamp (X) i.e. 149 < 151 (W-Timestamp)
			Cannot write as other transaction using data X for writing.
150	....	....	....
151		WRITE (X)	Record W-timestamp(X) = 151

(c) Otherwise, system executes WRITE (X) and sets W-timestamp (X) = TS (Ti)

TS	Ti	Tx	Operations
150	....	....	....
151	WRITE (X)		Record W-timestamp(X) = 151

(4) Advantages

- This protocol ensures conflict serializability, as conflicting operations are processed in order of timestamp of operation.
- Ensures that it is free from deadlock.

(5) Disadvantages

- Starvation is possible for long transaction if short transaction conflicts with it causes restorative of long transaction again and again.
- It may not give recoverable schedules.

□□□

Chapter - 14 : Recovery Methods

Q.1 Explain the concept of log and how it helps with database recovery with suitable diagram.

SPPU - May 19, Dec. 19, 4 Marks

Ans. : Log-Based Recovery

- There can be problem in accessing database due to any reason can causes a database system failure.
- The most widely used structure for recording database modifications is **transaction log (Log).**
- The log is a sequence of log records, recording all the update activities done on the database by all database users.

Types of Transaction Log

There are several types of log records.

(a) Update log record

- An update log record describes a single database write.
- It also includes the value of the bytes of page before and after the page change.

(b) Compensation log record

- Compensation log record the rollback of a particular change to the database.
- Each corresponds with exactly one other Update Log Record.

(c) Commit record

Records a decision to commit a transaction.

(d) Abort record

Records a decision to abort and hence rollback a transaction.

(e) Checkpoint record

- Records a point when checkpoint has been made.
- These are used to speed up recovery.
- It also record information that eliminates the need to read a log's past.
- The time of recording varies according to checkpoint algorithm.

(f) Completion record

Records all work has been done for this particular transaction.

Fields in transaction log

It has these fields :

(a) Transaction identifier : It is the unique identifier of the transaction that performed the write operation.

(b) Data-item identifier : It is the unique identifier of the data item written. Typically, it is the location on disk of the data item.

(c) Old value : It is the value of the data item prior to the write.

(d) New value : It is the value that the data item will have after the write.

Working of Log Based Protocol

- There are some special log records exist to record events during transaction processing, such as the start of a transaction and the commit or abort of a transaction.
- We denote the various types of log records as :

  $< T_n$ start$>$: Transaction T_n has started.

  $< T_n, X_j, V_1, V_2 >$: Transaction T_n; has performed a write on data item X_j. X_j had value V_1 before the write, and will have value V_2 after the write.

  $< T_n$; commit$>$: Transaction T_n has committed.

  $< T_n$ abort$>$: Transaction T_n has aborted.
- Whenever a transaction performs a write operation, it is essential that the log record for that transaction should be created before the database is modified by it.
- Once a log record exists, we can write the modification to the database only if it is desirable.
- Undo operation can be done for modification that has already been done to the database. We undo it by using the old-value field in log records.
- For log records to be useful for recovery from system and disk failures, the log must reside in stable storage.

- We assume that every log record is written to the end of the operation on stable storage as soon as log is created.

Q.2 What is the basis of immediate updates recovery technique ? What does the deterred updates recovery technique involve? **SPPU - May 19, 6 Marks**

Ans. : Immediate-Modification Technique (UNDO Algorithm)

1. Introduction

- The **immediate-modification technique** allows database modifications to be written to the database when the transaction is still in the active state. (Running state)
- Data modifications written by active transactions are called **uncommitted modifications.**
- In the event of a crash or a transaction failure, the system must use the old-value field of the log records to restore the modified data items to the value had prior to the start of the transaction.

2. UNDO operation

- Before a transaction T_n starts its execution, the system writes the record <T_n start> to the log.
- During transaction execution, any WRITE**(X) operation by T_n** is preceded by the writing of the appropriate new update record to the log.
- When **T_n partially commits**, the system writes the record <T_n commit> to the log.
- Since the information in the log is used in restoring the original database state, we cannot allow the actual update to the database to take place before the corresponding log record is written out to stable storage.
- We therefore require that, before the execution of an output (B) operation its log records is written to stable storage.
- This procedure is defined as follows :

(1) Use two lists of transactions maintained by the system;

(a) Committed transactions since the last checkpoint

(b) Active transactions

(2) Undo all the WRITE operations of the active transaction from the log, using the UNDO procedure.

(3) Redo the WRITE operations of the committed transaction from the log in the order in which they were written in the log, using the REDO procedure.

UNDO operation

- Undoing a WRITE operation, consists of examination its log entry of write T reading Old value, new value and setting the value of item X in the database to old value which is the before image.
- Undoing a number of WRITE operations from one or more transactions from the log must proceed in the reverse order from the order in which the operations were written in the log.

Example :

- Consider a simple banking system, with transactions T_0 and T_1 executed one after the other in the order T_0 followed by T_1.
- Figure below shows possible order of execution in both the database and the log as a result of the execution of T_0 and T_1

<T_0 start>
<T_0, A, 100, 50>
<T_0, B, 100, 150>
< T_0 commit>
<T_1 start>
<T_1, C, 100, 200>
<T_1 commit>

Log	Database
<T_0 start> <T_0, A, 1200, 950> <T_0, B, 2000,2050>	
	A = 950 B = 2050
<T_0 commit> <T_1 start> <T_1, C, 700, 800> < T_1 commit >	 C = 800

- Using the log file, the system can handle any failure that does not result in the loss of information in non volatile storage.
- The recovery scheme uses two recovery procedures :

Method 1 :

(a) Undo(T_n) restores the value of all data items updated by transaction T_n to the old values and redo(T_n) sets the value of all data items updated by transaction T_n to the new values.

(b) The set of data items updated by T_n and their respective old and new values can be found in the log.

(c) The undo and redo operations must guarantee to correct behaviour, even if a failure occurs during the recovery process.

(d) After a failure has occurred, the recovery scheme consults the log to determine which transactions need to be redone, which need to be undone:

- Transaction T_n needs to be undone if the log contains the record < T_n start>, but does not contain the record <T_n commit>.
- Transaction T_n needs to be redone if the log contains both the record < T_n start> and the record <T_n commit>.
- In our banking example, with transaction T_0 and T_1 executed one after the other in the order T_0 followed by T_1.

<T_0 start>
<T_0, A, 100, 50>
<T_0, B, 100, 150>

Fig. 14.1

Method 2 :

(a) If system crashes before the completion of the transactions

- First, let us assume that the crash occurs just after the log record for the step of transaction T_0 has been written to stable storage.
- When the system comes back up, it finds the record <T_0 start> in the log, but no corresponding <T_0 commit> record.
- Thus, transaction T_0 must be undone, so an undo (T_0) is performed.
- As a result, the values in accounts A and B (on the disk) are restored to Rs.100 and Rs.100, respectively.

<T_0 start>
<T_0, A, 100, 50>
<T_0, B, 100, 150>
<T_0 commit>
<T_1 start>
<T_1, C, 300, 200>

Fig. 14.2

(b) If crash comes occurs after the log record for the step write (C) of transaction T_1 has been written to stable storage.

- When the system comes back up, two recovery actions need to be taken.
- The operation UNDO(T_1) must be performed, since the record <T_1 start> appears in the log, but there is no record <T_1 commit>.
- The operation REDO(T_0) must be performed, since the log contains both the record <T_0 start> and the record <T_0 commit>.
- At the end of the entire recovery procedure, the values of accounts A, B, and C are Rs.50, Rs.150, and Rs.300, respectively. Note that the UNDO(T_1) operation is performed before the redo(T_0).
- In this example, the same outcome would result if the order were reversed.
- However, the order of doing undo operations first, and then redo operations, is important for the recovery algorithm.

<T_0 start>
<T_0, A, 100, 50>
<T_0, B, 100, 150>
<T_0 commit>
<T_1 start>
<T_1, C, 200, 300>
< T_1 commit>

Fig. 14.3

(c) If crash occurs just after the log record <T_1 commit> has been written to stable storage.

- When the system comes to backup, both T_0 and T_1 need to be redone, since the records <T_0 start> and <T_0 commit> appear in the log, as do the records <T_1 start> and <T_1 commit>.
- After the system performs the recovery procedures REDO(T_0) and REDO(T_1), the values in accounts A, B, and C, are Rs. 50, Rs. 150, and Rs. 300, respectively.

Q.3 What is a checkpoint? Explain the operations performed by a system during checkpoint. Explain the recovery mechanism during system crash.

SPPU - May 18, Dec.18

Ans. : Recovery Related Structures – Checkpoint

Introduction

- A database checkpoint is where all committed transactions are written to the redo/audit logs.
- The database administrator determines the frequency of the checkpoints based on volume of transactions.
- When a system failure occurs, we must consult the log to determine those transactions that need to be redone and those that need to be undone using above log files.
- Too frequent checkpoints can affect the performance.

Problems in This Approach

- The search process is time-consuming.
- Most of the transactions that, according to our algorithm, need to be redone as they have already written their updates into the database.

Need of Checkpoints

- To reduce these types of overhead, we introduce checkpoints.
- During execution, the system maintains the log, using one of the two techniques.
- The system periodically performs checkpoints, with following sequence of actions :
  1. Output all log records onto stable storage which are currently stored in main memory.
  2. Output to the disk all modified buffer blocks.
  3. Output onto stable storage a log record <checkpoint>.
- Transactions are not allowed to perform any update actions, such as writing to a buffer block or writing a log record, while a checkpoint is in working state.
- The presence of a <checkpoint> record in the log allows the system to restructure its recovery procedure.

Working

- Consider a transaction T_n that committed prior to the checkpoint.
- For such a transaction, the <T_n commit> record appears in the log before the <checkpoint> record.
- Any database modifications made by transaction T_n must have been written to the database either prior to the checkpoint or as part of the checkpoint itself. Thus, at recovery time, there is no need to perform a redo operation on T_n.
- After a database failure has occurred, the recovery scheme examines the log to determine the most recent transaction T_n that started executing before the most recent checkpoint took place.
- It can find such a transaction by searching the log backward, from the end of the log, until it finds the first <checkpoint> record (since we are searching backward, the record found is the final <checkpoint> record in the log); then it continues the search backward until it finds the next <T_n start> record. This record identifies a transaction T_n.
- Once the system has identified respective transaction T_n, the redo and undo operations need to be applied to only for transaction T_n and all transactions that started executing after transaction T_n.
- The exact recovery operations to be performed depend on the modification technique being used.
- For the immediate-modification technique, the recovery operations are :
  - For all transactions T_k in T that have no <Tk commit> record in the log, execute UNDO(Tk).

- For all transactions Tk in T such that the record <Tk commit> appears in the log, execute REDO(T_k).

- Obviously, the undo operation does not need to be applied when the deferred-modification technique is being employed.
- Consider the set of transactions {T_0, T_1 .... , T_{10}} executed in the order of the subscripts. Suppose that the recent checkpoint took place during the execution of transaction T_6. Thus, only transactions T_6, T_7, ... , T_{10} need to be considered during the recovery scheme. Each of them needs to be redone if it has committed; otherwise, it needs to be undone.

Advantages

(1) A Database storage checkpoint keeps track of block change information and thereby enables incremental database backup at the block level.

(2) A Database Storage Checkpoint helps recover data from incorrectly modified files.

(3) A Database Storage Checkpoint can be mounted, allowing regular file system operations to be performed. Mountable Database Storage Checkpoints can be used for a wide range of application solutions that include backup, investigations into data integrity, staging upgrades or database modifications, and data replication solutions.

Disadvantages

(1) Database Storage Checkpoints can only be used to restore from logical errors (E.g. a human error).

(2) Because all the data blocks are on the same physical device, Database Storage Checkpoints cannot be used to restore files due to a media failure.

Q.4 Explain shadow paging with proper example.

SPPU - Dec. 17, 4 Marks

Ans. : Shadow Paging

Introduction

- It is not always convenient to maintain logs of all transactions for the purposes of recovery. An alternative is to use a system of shadow paging.
- This is where the database is divided into pages that may be stored in any order on the disk.
- In order to identify the location of any given page, we use something called a **page table.**

Overview

- This method considers the database is made up of a number of fixed size pages.
- A directory with n entries is constructed which the entry points to the database page on disk.
- This directory is kept in the main memory.
- It is not too large and all references read and write to database pages on disk.

Method

(a) During the life of a transaction two page tables are maintained as below,

(i) Shadow page table

(ii) Current page table.

(b) When a transaction begins both of these page tables point to the same locations (are identical).

(c) During the lifetime of a transaction the shadow page table doesn't change at all.

(d) However during the lifetime of a transaction update values etc. may be changed.

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

(f) So whenever we update a page in the database we always write the updated page to a new location.

(g) This means that when we update our current page table it reflect the changes that have been made by that transaction.

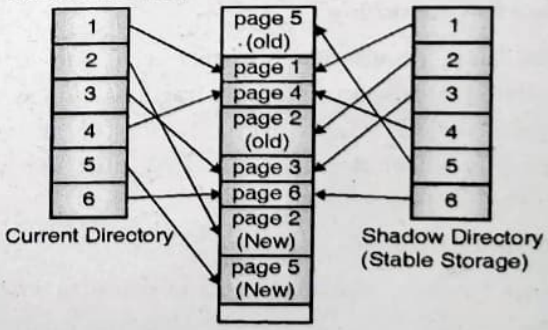

Fig. 14.4 : Page table -shadow paging

(h) Looking at below diagram we see how these tables appear during a transaction. As we can see the

shadow page table shows the state of the database just prior to a transaction, and the current page table shows the state of the database during or after a transaction has been completed.

Process of Recovery

- We now have a system whereby if we ever want to undo the actions of a transaction all we have to do is recover the shadow page table to be the current page table.
- As such this method makes the shadow page table particularly important, and so it must always be stored on stable storage.
- On disk we store a single pointer location that points to the address of the shadow page table.
- This means that to swap the shadow table for the current page table (commiting the data) we just need to update this single pointer (very unlikely to fail during this very short fast operation).
- In case of no failure means while committing transaction just discard the shadow directory.
- In case of multi-user environment with concurrent transaction, logs and checkpoints must be incorporated in shadow paging.

Advantages

- Shadow page method does not require any Undo or Redo algorithm for recovery purpose.
- Recovery using this method will be faster.
- No Overhead for writing log records.

Disadvantages

(1) Data fragmentation

The main disadvantage of this technique is the updated Data will suffer from fragmentation as the data is divided up into pages that may or not be in linear order for large sets of related hence, complex storage management strategies.

(2) Commit overhead

If the directory size is large, the overhead of writing shadow directories to disk as transaction commit is significant.

(3) Garbage collection

- Garbage will accumulate in the pages on the disk as data is updated and pages loose any references. For example if i have a page that contains a data item X that is replaced with a new value then a new page will be created. Once the shadow page table is updated nothing will reference the old value of X.
- The operation to migrate between current and shadow directories must be implemented as an atomic mode.

Q.5 Explain Query optimization with respect to SQL databases. **SPPU - Dec. 18, May 19, 6 Marks**

Ans. : Query Optimization in SQL

(1) **Query optimization** is an essential for many DBMS in which best query execution plan, out of multiple alternative query execution plans is to be identified for solving a query efficiently.

(2) Query optimization is nothing but selecting the efficient query execution plan among the available query plans for solving a given query.

(3) The query plan is constructed based on multiple cost factors.

(4) Query optimization tries to select query plan which is lower processing cost (but not always low cost plans are selected as other factors are also included in query evaluation).

(5) Query optimization come into picture when we want to develop a good system to minimize the cost of query evaluation.

(6) There is a trade off between the **amount of time spent to find out the best plan** and the **amount of time required for running the plan**. Different DBMS have different ways for balancing these two factors.

Goals of Query Optimization

(a) Eliminate all unwanted data

Query optimization tries to eliminate unwanted tuples, or rows from given relation.

Example :

```
SELECT *
FROM EMPLOYEES
```

If we require only employee Ids then select only employee_Id column from table. No need to select all the columns from table by specifying * in select clause.

So, Optimized query will be

```
SELECT  Emp_ID
FROM    EMPLOYEES
```

(b) Speed up queries

Query optimization try to find out query which gives result very fast.

(c) Increase query performance by performance tuning

- Break up the single complex SQL statement into several simple SQL statements which will increase performance of execution of query.
- In many cases, you can build an index on the temporary table (such tables used to store intermediate results of query) in order to speed up the query performance.

(d) Select best query plan out of alternative query plan

Consider following query to "Find all employees having age above 25 and working for department HR."

```
SELECT  e.Emp_Name
FROM    EMPLOYEES e
        JOIN
        DEPARTMENTS d
        ON      e.did=d.did
WHERE     e.age > 25
AND             d.dname= 'HR';
```

For above query we can write alternative relational queries like below,

Option 1

$$\pi_{e.Emp\_Name}(\sigma_{e.Age>25\ AND\ d.dname='HR'}(EMPLOYEES\ e \bowtie_{e.did=d.did} Departments\ d))$$

Option 2

$$\pi_{e.Emp\_Name}((\sigma_{e.Age>25}(EMPLOYEES\ e)) \bowtie_{e.did=d.did} (\sigma_{d.dname='HR'}(Departments\ d)))$$

Now, which of above query option is to be selected is identified by query optimizer.

Query Optimization Approaches / Performance Tuning

Relational approach

In RDBMS system attempts to find query expression which is equivalent to the given query expression, but execute in more efficient way.

Algorithm approach

Selecting query by choosing algorithm for operation or choosing specific indices to use and so on.

Other commonly used strategies for optimization

- **Use index**
  - Using an index is strategy that can used to speed up a query evaluation.
  - This strategy is very important for query optimization.
- **Aggregate table**
  - Tables which stores only higher level data so fewer amounts of data need to be parsed.
  - So, Query evaluation becomes faster.
- **Vertical partitioning**
  - Slicing the table vertically using columns.
  - Vertical partitioning will decrease the amount of data required for query processing.
- **Horizontal partitioning**
  - Partition the table by data value, or by row wise.
  - This method will decrease the amount of data query needs to process.
- **Denormalization**
  - The process of combining multiple tables into a single table is called as denormalization.
  - This speeds up query performance because less number of joins is required.
- **Server tuning**

  Each server has its own parameters, and often tuning server parameters so that it can fully take advantage of the hardware resources and significantly speed up the query performance.

Query Optimizers

- Query optimization is one of the most important tasks of a relational DBMS.
- The Query optimizer generates alternative plans and chooses the best plan with the least estimated cost.
- Query optimizer is responsible for identifying a best execution plan for evaluating the query.
- The optimizer generates multiple query plans and selects the plan with the least estimated cost.
- To estimate the cost of every plan, the optimizer uses system catalogs.
- The system catalogs contain the information needed by the optimizer to choose between alternate plans for a given query.

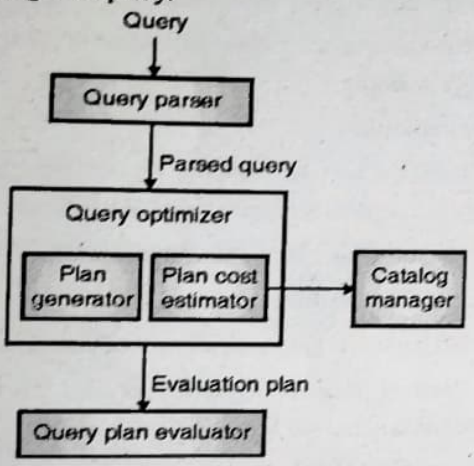

Fig. 14.5 : Query parsing, optimization and execution

- Query evaluation plan can have a remarkable impact on query execution time.

Query Optimization in NoSQL

(1) In case of NoSQL data, efficiency of read operations can be improved by reducing the amount of data that query operations need to process.

(2) If NoSQL queries a set of fields, then an index on such queried field can prevent the query from scanning entire data set.

(3) Governing methods to optimize query performance is done by Performance Tuning.

(4) Query selectivity can be used for performance tuning which refers to filtering out unrelated documents in a collection.

(5) Many of above SQL Query optimization approaches are applicable for NoSQL database also.

❑❑❑