Transaction

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the content of the database.

Example: Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction ~~cent~~ contains several low-level tasks:

X's Account

Open- Account(X)
Old_Balance = X.Balance
New-Balance = Old_Balance - 800
X. balance = New_Balance
Close- Account(X)

Y's Account

Open_ Account(Y)
Old_ Balance = Y.Balonce
New- Balonce = Old-Balance + 800
New- Balance = Old-Balance + 800
Y. balance = New_Balance.
Close- Account(Y)

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

(1.) R(x);

(2) X = X - 500;

(3) W(x);

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of x by 500. so buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

Rollback: It is used to undo the work done.

Transaction Property

The transaction has the four ~~pr~~ properties. These are used to maintain consistency in a database, before and after the transaction.

Properties of Transaction

1.) Atomicity → means either all successful or none

2.) Consistency → ensures bringing the database form one consistent state to another consistent state. ~~ensures bringing the database from one consistent state~~

3.) Durability → means once a transaction has been committed, it will ~~are~~ remain so, even in the event of errors, power loss etc.

4.) Isolation → ensures the transaction is isolated from other transaction.

Atomicity

* It states that all operations of the transaction take place at once if not, the transaction is aborted.

* There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort : If a transaction aborts then all the changes made are not visible.

Commit : If a transaction commits then all the changes made are visible.

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of ₹600 and B consists of ₹300. Transfer ₹100 from account A to account B.

T1	T2
Read (A)	Read (B)
A:= A-100	Y:=Y+100
Write (A)	write(B)

After completion of the transaction, A consists of ₹500 and B consists of ₹400.

If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will have a database in either its prior stable state or a new stable state.

- The transaction is used to transform the database from one consistent state to another consistent state. For example: The total amount must be maintained before or after the transaction.

Total before T occurs = 600 + 300 = 900

Total after T occurs = 500 + 400 = 900.

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, the inconsistency will occur.

Isolation

- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

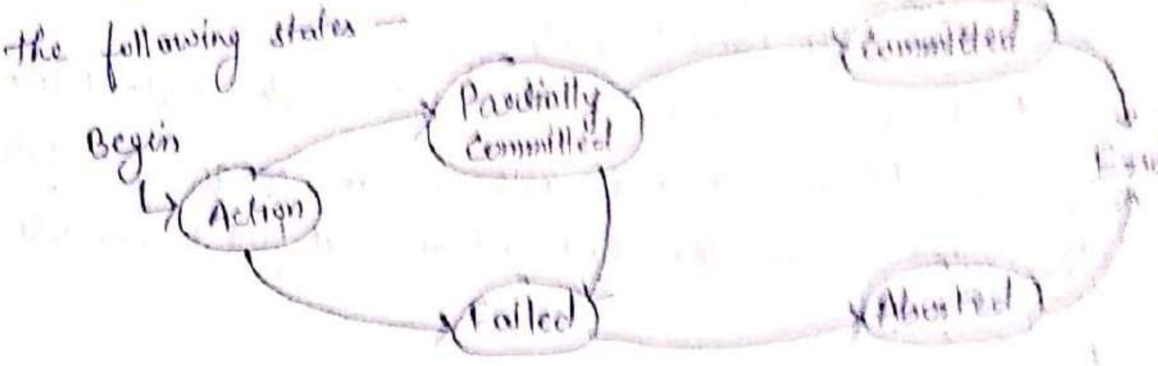
- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the

system failure. when a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.

- The recovery of the DBMS has the responsibility of Durability property.

States of Transaction

In a database, the transaction can be in one of the following states –



Active State

1) • The active state is the first state of every transaction. In this state, the transaction is being executed.

• For example: Insertion ~~and~~ or deletion or updating a record is done here. But all the records are are not saved to the database

Partially Committed

2) • In the partially committed state, a transaction executes its final operation but the data is still not saved to the database.

• In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed

A transaction is said to be in committed state if it executes all its operation 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 the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

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 ~~pre~~ previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:

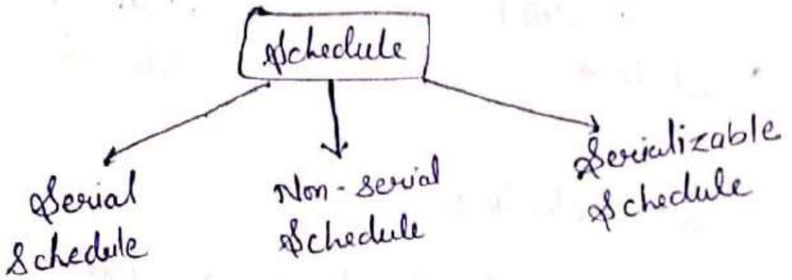
(1) Re-start the transaction

(2) Kill the transaction.

Lecture No-34

Schedule

A series of operation form one transaction to another transaction is known as schedule. It is used to preserve the order of the operation in each of the individual transattion.



1.) Serial Schedule

The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

For example:- Suppose there are two transaction T1 and T2 which have some operation. If it has no interleaving of operations, then there are the following two possible outcomes

1. Execute all the operations of T1 which was followed by all the operations of T2.

2. Execute all the operations of T1 which was followed by all the operations of T2.

- In figure (a), Schedule A shows the serial schedule where T1 followed by T2.
- In figure (b), Schedule B shows the serial

schedule where T2 followed by T1.

(a)

Time

T1	T2
redd (A)	
A:= A-N	
write (A);	
read (B);	
B:= B+N;	
write(B);	
	read(A);
	A:= A+M;
	write (A);

Schedule A

(b)

Time

T1	T2
	read (A);
	A:= A+M;
	write (A);
read(A);	
A:= A-N;	
write (A);	
read (B);	
B:= B+N;	
write (B);	

Schedule B.

⇒ (2) <u>Non-serial Schedule</u>

- If interleaving of operations is allowed, then there will be non-serial schedule.
- It contains many possible orders in which the system can execute the individuals operations of the transactions.
- In figure (c) and (d), Schedule C and Schedule D are non-serial schedules. It has interleaving of operations.

(c)

Time

T1	T2
read(A);	
A:= A-N;	
	read (A);
	A:= A+M;
write (A);	
read (B);	
	write(A);
B:= B+N;	
write(B);	

Schedule C

(d)

Time

T1	T2

read (A);
A := A-N;
write (A);	

	read (A);
	A := A+M;
	write (A);

read (B);
B := B+N;
write(B);	

(B) Serializable Schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule will be serializable if it its result is equal to the result of its transactions executed serially.

Testing of Serializability

Serialization Graph is used to test the ~~Seriable~~ serializability of a schedule.

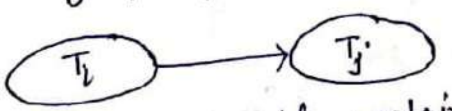
Assume a schedule S. For S, we construct a graph known as precedence graph. This graph has a pair G = (V, E), where V consist a set of vertices, and E consists a set of edges. The set of vertices is used to contain all the transactions participating in the schedule. The set of edges is used to contain all edges $T_i \rightarrow T_j$ for which one of the three conditions holds:

1. Create a node $T_i \rightarrow T_j$ if T_i executes write(Q) before T_j executes read(Q).

2. Create a node $T_i \rightarrow T_j$ if T_i executes ~~red~~ read(Q) before T_j executes write(Q).

3. Create a node $T_i \rightarrow T_j$ if T_i executes write(Q) before T_j executes write(Q).

Precedence graph for schedule S



- If a precedence graph contains a single edge $T_i \rightarrow T_j$, then all the instructions of T_i are executed before the first instruction of T_j is executed.
- If a precedence graph for schedule S contains a cycle, then S is non-serializable. If the precedence graph has no cycle, then S is known as serializable.

For example:

~~Exa~~ Explanation:

Read(A): In T1, no subsequent writes to A, so no new edges.

Read(B): In T2, no subsequent writes to B, so no new edges.

Read(C): In T3, no subsequent writes to C, so no new edges.

write(B): B is subsequently read by T3, so add edge T2→T3.

write(C): C is subsequently read by T1, so add edge T3→T1

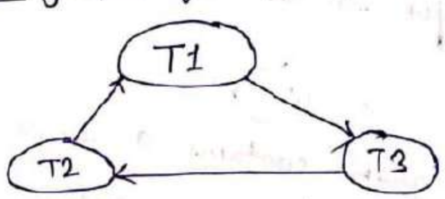
write(A): A is subsequently read by T2, so add edge T1→T2.

write(A): In T2, no subsequently reads to A, so no new edges.

write(C): In T1, no subsequently reads to C, so no new edges.

write(B): In T3, no subsequently reads to B, so no new edges.

Precedence graph for schedule S1:



The precedence graph for schedule S1 contains a cycle a cycle that's why schedule S1 is non-serializable.

T4	T5	T6
Read(A)		
$A := f_1(A)$		
Read(C)		
$A := f_2(C)$	Read(B)	
write(C)	Read(A)	
		Read(C)

Time ↓

$B := f_3(B)$ Write (B)	
	$C := f_4(C)$ Read (B) Write (C)
$A := f_5(A)$ write (A)	
	$B := f_6(B)$ write (B)

Schedule S_2

Explanation:

Read (A): In T4, no subsequent writes to A, so no new edges.

Read (c): In T4, no subsequent writes to C, so no new edges.

write (A): A is subsequently read by T5, so add edge T4 → T5.

~~[illegible]~~ Read(B): In T5, no subsequent writes to B, so no new edges.

write (C): C is subsequently read by T6, so add edge T4 → T6

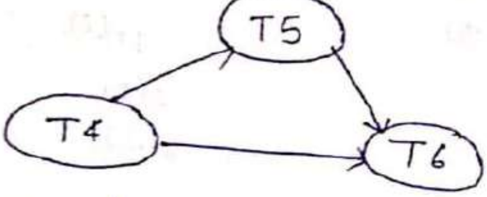
write (B): A is subsequently read by T6 so add edge T5 → T6

write(C): In T6, no subsequent reads to C, so no new edges

write (A): In T5, no subsequent reads to A, so no new edges

write (B): In T6, no subsequent reads to B, so no new edges.

Precedence graph for schedule S2:



The precedence graph ~~of~~ for S2 contains no cycle that's why schedule S2 is serializable.

Lecture No - 36

Conflict Serializable Schedule

• A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.

• The schedule will be a difficult serializable if it is conflict equivalent to a serial schedule.

Conflicting Operations

The two operations become conflicting if all conditions satisfy:

(1) Both belong to seperate transactions.

(2) They have the same data item.

(3) They contain at least one write operation.

Example:

Swapping is possible only if S1 and S2 are logically equal.

1. T1: Read(A) T2: Read(A)

T1	T2
Read(A)	
	Read(A)

Schedule S_1

→

T1	T2
	Read(A)
Read(A)	

Schedule S_2.

Here, S1 = S2. That means it is conflict.

2: T1: Read(A) T2: Write(A)

T1	T2
Read(A)	
	Write(A)

→

T1	T2
	Write(A)
Read(A)	

Schedule S_2.

Here, S1 ≠ S2. That means it is conflict.

Conflict Equivalent

In the conflict equivalent, one can be transformed to another by swapping non-conflicting operations. In the given example, S2 is conflict equivalent S1 (S1 can be converted to S1 by swapping non-conflicting operations).

Two schedules are said to be conflict equivalent if and only if:

1. They contain the same set of the transactions.
2. If each pair of conflict operations are ordered in the same way.

Example:

Non-serial schedule

T1	T2
Read(A)	
write(A)	
	Read(A)
	write(A)
Read(B)	
write(B)	
	Read(B)
	write(B)

schedule S1

Serial Schedule

T1	T2
Read (A)	
write (A)	
Read(B)	
write (B)	
	Read (A)
	write (A)
	Read (B)
	write (B)

Schedule S2

Schedule S2 is a serial schedule because, in this, all operations of T1 are performed before starting any operation of T2. Schedule S1 can be transformed into a serial schedule by swapping non-conflicting operation of S1.

After swapping of non-conflict operations, the schedule S1 becomes:

T1	T2
Read (A)	
Write (A)	
Read (B)	
Write (B)	
	Read (A)
	write (A)
	Read (B)
	write (B)

Since, S1 is conflict serializable.

View Serializability

- A schedule will view serializable if it is view equivalent to ~~be~~ a serial schedule.
- If a schedule is conflict serializable, then it will be view sericalizable.
- The view serializable which does not conflict serializable contains blind writes.

View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following ~~statem~~ conditions.

1. Initial Read

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the date item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	
	write(A)

Schedule S1

T1	T2
	write(A)
Read(A)	

Schedule S2.

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Update Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A

Let's assume there are two transaction 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 can be 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 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 request for a resource which is held by the younger transaction, then older transaction forces younger transaction to kill the transaction and release the resource. After a minute delay, the younger transaction is restarted but with the same time stamp.

- 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.