CHAPTER 12

Module VI

Transaction Processing

Syllabus

Transaction : Transaction concept, State Diagram, ACID Properties, Transaction Control Commands, Concurrent Executions, Serializability – Conflict and View.

12.1 Concept of Transaction

Q. What is transaction? Explain concept of transaction using example.

- 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. 12.1.1)

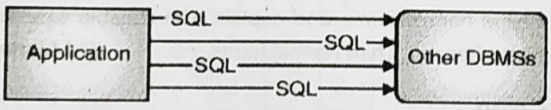


Fig. 12.1.1 : Executing single operation in DBMS

- In place of sending one by one SQL command to server we can combine multiple operations that are logically similar and send to serve 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.

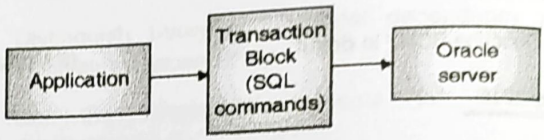


Fig. 12.1.2 : Executing transaction in DBMS

Types of operations

Types of operations that can be done inside transaction,

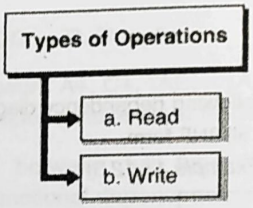


Fig. 12.1.3 : Types of operations

1. 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
```

2. 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. 12.1.4 : Sample implementation of transaction in SQL

12.1.1 Transaction Structure and Boundaries

1. 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 start monitoring the transaction status.
- All operations done after a BEGIN command is treated as a single large operation.

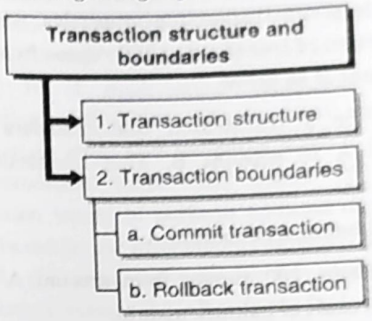
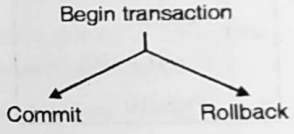


Fig. 12.1.5 : Transaction structure and boundaries

2. 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.



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.

12.2 Fundamental Properties of Transaction / ACID Properties

Q. Explain ACID properties of transaction.

- To understand transaction properties, we consider a transaction of transferring 100 rupees from account A to account B as below.
- Let T_i be a transaction that transfers 100 from account A to account B. This transaction can be defined as,
  a. Read balance of account A.
  b. Withdraw 100 rupees from account A and write back result of balance update.
  c. Read balance of account B.
  d. Deposit 100 rupees to account B and write back result of balance update.

T	Read (A);
i	A := A – 100;
	Write(A);
	Read(B);
	B := B + 100;
	Write(B)

Fig. 12.2.1 : Sample transaction

Transaction Properties are as follows :

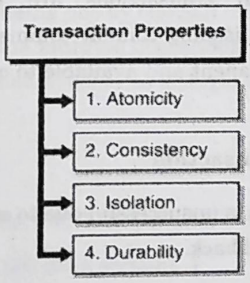


Fig. 12.2.2 : Transaction Properties

1. Atomicity

- Transaction must be treated as a single unit of operation.
- That means when a sequence of operations is performed in a single transaction, they are treated as a single large operation.

Examples

a. Withdrawing money from your account.
b. Making an airline reservation.

- The term atomic means thing that cannot be divided in parts as in atomic physics.
- Execution of a transaction should be either complete or nothing should be executed at all.
- No partial transaction executions are allowed. (No half done transactions)

Example 1 : Money transfer in above example

Suppose some type of failure occurs after Write (A) but before Write (B) then system may lose 100 rupees in calculation which may cause error as sum of original balance(A+B) in accounts A and B is not preserved (such situation is called as inconsistency explained later), In such case database should automatically restore original value of data items.

Example 2 : Making an airline reservation

a. Check availability of seats in desired flight.
b. Airline confirms your reservation
c. Reduces number of available seats
d. Charges your credit card (deduct amount from your balance)
e. Increases number of meals loaded on flight (Sometimes)
   - In above case either all above changes are made to database or nothing should be done as half-done transaction may leave data as incomplete state.
   - If one part of the transaction fails, the entire transaction fails and the database state is left unchanged.

2. Consistency

- Consistent state is a database state in which all valid data will be written to the database.
- If a transaction violates some consistency rules, the whole transaction will be rolled back and the database will be restored to its previous consistent state with those rules.
- On the other hand, if a transaction is executed successfully then it will take the database from one consistent state to another consistent state.

- DBMS should handle an inconsistency and also ensure that the database is clean at the end of each transaction.
- Consistency means transaction will never leave your database in a half finished (inconsistent) state.
- If one part of the transaction fails, all of the pending changes made by that transaction are rolled back.

Example : Money transfer in above example

- Initially in total balance in accounts A is 1000 and B is 5000 so sum of balance in both accounts is 6000 and while carrying out above transaction some type of failure occurs after Write (A) but before Write (B) then system may lose 100 rupees in calculation.
- As now sum of balance in both accounts is 5900 (which should be 6000) which is not a consistent result which introduces inconsistency in database.
- This means that during a transaction the database may not be consistent.

3. Isolation

- Isolation property ensures that each transaction must remain unaware of other concurrently executing transactions.
- Isolation property keeps multiple transactions separated from each other until they are completed.
- Operations occurring in a transaction (example, insert or update statements) are invisible to other transactions until the transaction commits (on transaction success) or rolls back (on transaction fails).
- For example, when a transaction changes a bank account balance other transactions cannot see the new balance until the transaction commits.
- Different isolation levels can be set to modify this default behaviour.
- Transaction isolation is generally configurable in a variety of modes. For example, in one mode, a transaction locks until the other transaction finishes.
- Even though many transactions may execute concurrently in the system. System must guarantees that, for every transactions (T_i) all other transactions has finished before transactions (T_i) started, or other transactions are started execution after transactions (T_i) finished.
- That means, each transaction is unaware of other transactions executing in the system simultaneously.

Example : Money transfer in above example.

The database is temporarily inconsistent while above transaction is executing, with the deducted total written to A and the increased total is not written to account B. If some other concurrently running transaction reads balance of account A and B at this intermediate point and computes A+B, it will observe an inconsistent value (Rs. 5900). If that other transaction wants to perform updates on accounts A and B based on the inconsistent values (Rs. 5900) that it read, the database may be left database in an inconsistent state even after both transactions have completed.

A way to avoid the problem of concurrently executing transactions is to execute one transaction at a time.

4. Durability

- The results of a transaction that has been successfully committed to the database will remain unchanged even after database fails.
- Changes made during a transaction are permanent once the transaction commits.
- Even if the database server is fails in the between transaction, it will return to a consistent state when it is restarted.
- The database handles durability by transaction log.
- Once the execution of the above transaction completes successfully, and the user will be notified that the transfer of amount has taken place, if there is no system failure in this transfer of funds.
- The durability property guarantees that, all the changes made by transaction on the database will be available permanently, although there is any type of failure after the transaction completes execution.

12.3 States of Transaction

Q. Explain states of transaction with neat diagram.
Q. Draw transaction state diagram.
Q. Describe state of transaction with neat diagram.

- 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 must be in one of the following states :

1. 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.

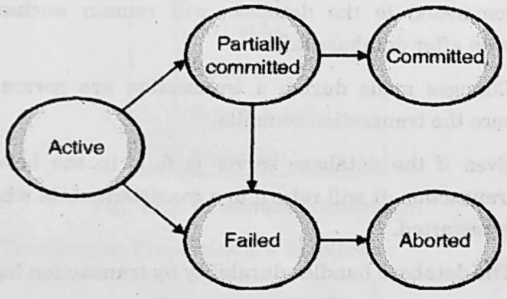


Fig. 12.3.1 : State diagram of a transaction

2. 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.

3. 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.

4. 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 :
  - **Restart the transaction :** A restarted transaction is considered to be a new transaction which may recover from possible failure.
  - 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.

5. 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.

12.4 Transactions Schedules

Q. What is transaction schedule?

1. Introduction

- Schedule is a sequence of instructions that specify the sequential order in which instructions of transactions are executed.
- A schedule for a set of transactions must consist of all instructions present in that transactions, it must save the order in which the instructions appear in each individual transaction.
- A transaction that successfully completes its execution will have to commit all instructions executed by it at the end of execution.
- A transaction that fails to successfully complete its execution will have to abort all instructions executed by transaction at the end of execution.

2. Representation

- We denote this transaction T as,
- $R_T(X)$: Denotes read operation performed by the transaction T on object X.
- $W_T(X)$: Denotes write operation performed by the transaction T on object X.
- Each transaction must specify its final action as commit or abort.

12.5 Serial Executions / Transactions / Schedules

Q. What is serial schedule? Give one example. **(4 Marks)**

(1) Introduction

- This is simple model in which transactions executed in a serial order that means after finishing first transaction second transaction starts its execution.
- This approach specifies first my transaction, then your transaction other transactions should not see preliminary results.

(2) Example

Consider below two transactions T_1 and T_2.

Transaction T_1 : deposits Rs.100 to both accounts A and B.

Transaction T_2 : doubles the balance of accounts A and B.

T_1 :

Read(A)
A ← A + 100
Write(A)
Read(B)
B ← B + 100
Write(B)

T_2 :

Read(A)
A ← A*2
Write(A)
Read(B)
B ← B*2
Write(B)

Serial schedule for above transaction can be represented as below,

Schedule A : A consistent serial schedule

A consistent serial schedule is obtained by executing T_1 right after T_2.

T_1	T_2	A	B	Operations
		25	25	Initial Balance
	Read(A); A ← A*2			
	Write(A)	50	25	
	Read(B); B ← B*2			
	Write(B)	50	50	
Read(A); A ← A + 100				
Write(A)		150	50	
Read(B); B ← B + 100				
Write(B)		150	150	
		150	150	Final Balance

In above Serial schedule for we can first execute transaction T_2 then T_1 which may results in some final values.

Representation : $T_1 \rightarrow T_2$

Schedule B : A consistent serial schedule

A serial schedule that is also consistent is obtained by executing T_2 right after T_1.

T_1	T_2	A	B	Operations
		25	25	Initial Balance
Read(A); A ← A + 100				
Write(A)		125	25	
Read(B); B ← B + 100				
Write(B)		25	125	
	Read(A); A ← A*2			
	Write(A)	250	125	
	Read(B); B ← B*2			
	Write(B)	250	250	
		250	250	Final Balance

Representation : $T_2 \rightarrow T_1$

In above schedule T_1 is executed entirely then T_2 has started. Assume account A with Rs.25 and B with Rs.25. Then transaction T_1 will update A as 125 and B as 125. Now T_2 will read updated values of A and B. T_2 will update value of A as 250 and B as 250. The consistency constraint is A + B should remain unchanged. So at end of T_2, A + B.

i.e. 250 + 250 = 500 remains unchanged so execution of this schedule keeps database in consistent state.

12.6 Concurrent Executions / Transactions / Schedules

Q. Explain concurrent execution and give example. **(4 Marks)**

1. Introduction

- Transactions executed concurrently, that means operating system executes one transaction for some time then context switches to second transaction and so on.
- Transaction processing can allows multiple transactions to be executed simultaneously on database server.
- Allowing multiple transactions to change data in database concurrently causes several complications with consistency of the data in database.
- It was very simple to maintain consistency in case of serial execution as compare to concurrent execution of transactions.

2. Advantages of concurrency

A. Improved throughput

- Throughput of transaction is defined as the number of transactions executed in a given amount of time.
- If we are executing multiple transactions simultaneously that may increase throughput considerably.

B. Resource utilization

- Resource utilization defined as the processor and disk performing useful work or not (in ideal state).
- The processor and disk **utilization** increase as number of concurrent transactions increases.

C. Reduced waiting time

- There may be some small transaction and some long transactions may be executing on a system.
- If transactions are running serially, a short transaction may have to wait for a earlier long transaction to complete, which can lead to random delays in running a transaction.

3. Example

Consider below two transactions T_1 and T_2,

Transaction T_1 : deposits Rs.100 to both accounts A and B.

T_1 :

Read(A)
A ← A + 100
Write(A)
Read(B)
B ← B + 100
Write(B)

Transaction T_2 : doubles the balance of accounts A and B.

T_2 :

Read(A)
A ← A*2
Write(A)
Read(B)
B ← B*2
Write(B)

Above transaction can be executed concurrently as below,

Schedule C : A schedule that is not serial but is still consistent

Obtained by interleaving the actions of T_1 with those of T_2

Table 12.6.1

T_1	T_2	A	B	Operations
		25	25	Initial Balance
Read(A); A ← A + 100;				
Write(A);		125		

T_1	T_2	A	B	Operations
	Read(A); A ← A*2; Write(A);	250		
Read(B); B ← B + 100; Write(B);			125	
	Read(B); B ← B*2; Write(B);			
		250	250	Final Balance

In above given schedule Part of T_1 is executed which updates A to 125. Then processor switches to T_2 and part of T_2 which updates A to 250 is executed. Then context switch to T_1 and remaining part of T_1 which updates B to 250 is executed. At the end remaining part of T_2 which reads B as 125 and updates it to 250 by multiplying value of B by two. This concurrent schedule also maintains consistency of database as ultimately A + B is 250 + 250 = 500 (unchanged).

The result of above transaction is exactly same as serial schedule shown in above Table 10.6.1. Therefore above schedule can be converted to equivalent serial schedule and hence it is consistent schedule.

12.7 Serializability / Serializable Schedule

Q. Explain the concept of serializability. (6 Marks)

1. Introduction

- The database system must control above concurrent execution of transactions serializability will ensure that the database state remains in consistent state.
- We first need to understand which schedules will ensure consistency, and which schedules will not.
- A schedule is the actual execution sequence of concurrent transactions.
- A schedule of two transactions T_1 and T_2 is 'serializable' if and only if executing this schedule has the same effect as any serial schedule (either $T_1 : T_2$ or $T_2 : T_1$).
- We consider only two operations: read and write for purpose of computational simplicity.
- Between a read (Q) instruction and a write (Q) instruction on a data item Q, a transaction may perform sequence of operations on the copy of Q that is residing in the local buffer of the transaction.
- Thus, the only important operations of a transaction from a scheduling point of view are its read and write instructions.
- Various forms of schedule equivalence are :

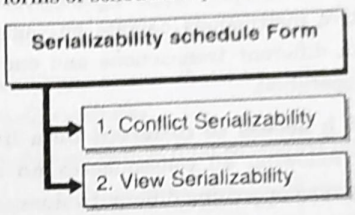


Fig. 12.7.1 : Serializability schedule form

12.7.1 Conflict Serializability

Q. Explain conflict serializability, describe using example. (4 Marks)

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 :

- Ii = READ (D), Ij = READ (D) then no conflict as they only read value.
- This operation is called as non conflicting swap.
- Ii = READ (D), Ij = WRITE (D) then they conflict and cannot be swapped.
- Ii = WRITE (D), Ij = READ (D) then they conflict and cannot be swapped.
- Ii = WRITE (D), Ij = 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. 12.7.2 : Schedule S

- Instruction WRITE(P) of T_1 and READ(P) of T_2 cannot be swapped as they conflict. (as shown in Fig. 12.7.3)

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

Can not Be Swap

Fig. 12.7.3

- Instruction READ(Q) of T_1 and READ(P) of T_2 can be swapped as they operate on different data items so do not conflict. (as shown in Fig. 12.7.4)

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

Can be swapped

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

Fig. 12.7.4

- Instruction WRITE(Q) of T_1 and WRITE(P) of T_2 can be swapped as they operate on different data items so do not conflict.

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

Can be swapped

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

Fig. 12.7.5

- Instruction WRITE (Q) of T_1 and READ (P) of T_2 can be swapped as they operate on different data items so do not conflict.

	T_1	T_2	T_1	T_2
	Read(P)		Read(P)	
	Write(P)		Write(P)	
	Read(Q)		Read(Q)	
can be Swap		**Read(P)**	**Write(Q)**	
As no conflict	**Write(Q)**			**Read(P)**
		Write(P)		Write(P)
		Read(Q)		Read(Q)
		Write(Q)		Write(Q)

Fig. 12.7.6 : Schedule R

- 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>.

12.7.2 View Serializability

Q. Explain view serializability, describe using example. (4 Marks)

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_2 must 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)

Fig. 12.7.7

4. Note

- Every conflict serialisable schedule is view serialisable but not vice versa.
- In above example T_2 and T_3 Writes data without reading value of data item by themselves so they are called as "Blind Writes".

12.8 Precedence Graph for Serializability (Test for Serializability)

Q. Explain method of precedence graph used for checking serializability. **(4 Marks)**

1. Introduction

- A particular transaction schedule can be serialized; we can draw a precedence graph.
- Precedence (or serializability) graph for schedule S is a graphical representation of transactions executed.
- A precedence graph is also known as conflict graph or serializability graph.
- Precedence graph is a graph of nodes and vertices, where the nodes are the transaction names and the vertices are attribute collisions.

2. Algorithm for precedence graph

a. Add a node for each transaction.

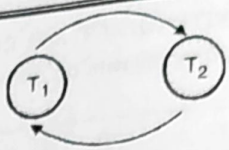
b. Add a directed edge from T_i to T_j , if T_j reads the value of an item written by T_i.

c. Add a directed edge from T_j to T_i, if T_j writes a value in to an item after it has been read by T_i.

3. Example 1

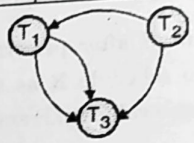
Let us consider Schedule C

T_1	T_2
Read(A); A ← A + 100;	
Write(A);	
	Read(A); A ← A*2;
	Write(A);
Read(B); B ← B + 100;	
Write(B);	
	Read(B); B ← B*2;
	Write(B);



Example 2

T_1	T_2	T_3
READ(A)		
READ(B)		
	READ(A)	
	READ(B)	
		WRITE (A)
WRITE (C)		
WRITE (B)		
		WRITE (C)



As no cycle in graph

∴ Schedule is conflict serializable

∴ Corresponding serial schedule is given as,

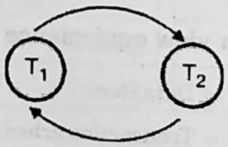
$T_2 \rightarrow T_1 \rightarrow T_3$

4. Test for conflict serializability

1. A schedule is conflict serialzable if it produces an acyclic precedence graph.
2. Example

$r_1(a), w_1(a), r_1(b), r_2(b), w_2(b), w_1(b)$

T_1	T_2
Read(A)	
Write(A)	
Read(B)	
	Read(B)
	Write(B)
Write(B)	



3. Conclusion

Not conflict serializable as there is cycle in above precedence graph.

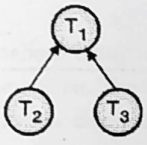
Q. What is the usage of precedence graph? Consider a following schedule with 3 transactions.

T_1	T_2	T_3
Read (A)		
	Write (A)	
Write (A)		
		Write (A)

Draw precedence graph for above. What you judge from the graph? Explain.

Ans. :

Precedence graph for the above graph is as follows;



There is no cycle present in the graph and hence it is not conflict serializable schedule.

12.9 Solved Examples

Example 12.9.1 : A schedule has transactions T_1 and T_2 as given below;

$r_1(x)$, $r_2(z)$, $r_1(z)$, $r_3(x)$, $r_3(y)$, $w_1(x)$, $w_3(y)$, $r_2(y)$, $w_2(z)$, $w_2(y)$

1. Draw precedence graph.
2. Is schedule conflict serialisable or not ? Find respective serial schedule.
3. Is above Schedule view serialisable or not ? **(10 Marks)**

Solution :

1. Step 1 :

$r_1(x)$, $r_2(z)$, $r_1(z)$, $r_3(x)$, $r_3(y)$, $w_1(x)$, $w_3(y)$, $r_2(y)$, $w_2(z)$, $w_2(y)$

Schedule S : Given Schedule

T_1	T_2	T_3
r(x)		
	r(z)	
r(z)		
		r(x)
		r(y)
w(x)		
		w(y)
	r(y)	
	w(z)	
	w(y)	

2. Step 2 : Precedence graph

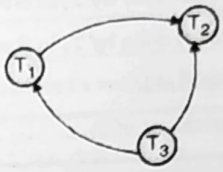


Fig. P. 12.9.1

$\therefore$ Above graph show no cycle

$\therefore$ Schedule s_1, is conflict serializable.

3. Step 3 : Corresponding serial schedule

$T_3 \rightarrow T_1 \rightarrow T_2$

Schedule S_1 : equivalent serial schedule

T_1	T_2	T_3
		r(x)
		r(y)
		w(y)
r(x)		
r(z)		
w(x)		
	r(z)	
	r(y)	
	w(z)	
	w(y)	

$\therefore$ **Serial Schedule S_1 :** $r_3(x)$, $r_3(y)$, $w_3(y)$, $r_1(x)$, $r_1(z)$, $w_1(x)$,$r_2(z)$, $r_2(y)$, $w_2(z)$, $w_2(y)$,

4. Step 4 : Above schedule is view serializable or not.

Conditions for view serializability

1. S and S_1 must have same number of transaction and same number of read write operation.
2. Initial read operation
   - X is read initially by T_3 in S as well as S_1

- Y is read initially by T_3 in S and S_1 both.
- Z is read initially by T_1 in both S and S_1

3. If S reads value of X or Y written by T; then S_1 also reads value of x or y written by same transaction.
   - S_1 reads value Y written by T_3.
   - S also reading value of Y which is written by T_3.
4. Final write operation
   - X-final write (x) done by T_1 in S as well as S_1
   - Y-final write (y) done by T_2 in S and S_2 both.

∴ S is view serializable.

Or S is view equivalent of S_1.

Review Questions

1. What is transaction? Explain concept of transaction using example.
2. Explain ACID properties of transaction.
3. Describe state of transaction with neat diagram.
4. State and explain Transaction structure.
5. Explain with example transaction processing.
6. Discuss the ACID properties of transaction processing.
7. Explain the transaction processing with the help of state diagram.
8. What is transaction ? What are functions of commit and rollback ?
9. Give an example of a schedule of 3 transactions that is not conflict serializable but view serializable ?
10. Discuss how serializability is used to enforce concurrency control. Why is serializability sometimes considered too restrictive as a measure of correctness for schedules ?
11. Explain state transition diagram illustrating the states for transaction execution. What are the desirable properties of transactions ?
12. What is transaction? Draw and explain life cycle of transaction.

❑❑❑