TRANSACTION

WHY WE STUDY TRANSACTION?

- According to general computation principle (operating system) we may have partially executed program, as the level of atomicity is instruction i.e. either an instruction is executed completely or not
- But in DBMS view, user perform a logical work(operation) which is always atomic in nature i.e. either operation is execute or not executed, there is no concept like partial execution.
- In this transaction if a failure occurs after Read(B) then the final statue of the system will be inconsistent as 100 units are debited from account A but not credited in account B, this will generate inconsistency.
- Here for 'consistency' before (A + B) == after (A + B)"

WHAT IS TRANSACTION

- To remove this partial execution problem, we increase the level of atomicity and bundle all the instruction of a logical operation into a unit called transaction.
- So formally 'A transaction is a Set of logically related instructions to perform a logical unit of work'.

As here we are only concerned with DBMS so we well only two basic operation on database

• **READ (X) -** Accessing the database item x from disk (where database stored data) to memory variable also name as X.

• **WRITE (X)** - Writing the data item from memory variable X to disk.

Q A transaction can include following basic database access operations:

(NET-JUNE-2011)

A) Read_item(X) (B) Write_item(X) (C) Both (A) and (B) (D) None of these

DESIRABLE PROPERTIES OF TRANSACTION

• Now as the smallest unit which have atomicity in DBMS view is transaction, so if want that our data should be consistent then instead of concentrating on data

base, we must concentrate on the transaction for our data to be consistent.

• Transactions should possess several properties, often called the ACID properties;

to provide integrity and consistency of the data in the database. The following are the ACID properties:

1) **Atomicity** - A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all. It is the responsibility of recovery control manager / transaction control manager of DBMS to ensure atomicity
2) **Consistency** - A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another.
   - The definition of consistency may change from one system to another.The preservation of consistency of database is the responsibility of programmers(users) or the DBMS modules that enforces integrity constraints.
3) **Isolation -** A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently.
   - That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.
   - The isolation property of database is the responsibility of concurrency control manager of database.
4) **Durability** - The changes applied to the database by a committed transaction must persist in the database.
   - These changes mu st not be lost because of any failure. It is the responsibility of recovery control manager of DBMS.

Q NOT a part of the ACID properties of database transactions? (GATE- 2016) (1 Marks)

(a) Atomicity (b) Consistency (c) Isolation (d) Deadlock-freedom

Q Consider the following transaction involving two bank accounts x and y. (GATE - 2015) (1 Marks)

read(x);	write(x);	y: = y + 50;
x: = x – 50;	read(y);	write(y)

The constraint that the sum of the accounts x and y should remain constant is that of

(A) Atomicity (B) Consistency (C) Isolation (D) Durability

Q All Oracle transactions obey the basic properties of a database transaction. What

is the name of the following property?‘All tasks of a transaction are performed or none of them are. There are no partial transactions.’ [Asked in Wipro NLTH 2021]

a) Atomicity b) Durability c) Consistency d) Isolation

TRANSACTION STATES

• **ACTIVE** - It is the initial state. Transaction remains in this state while it is executing operations.

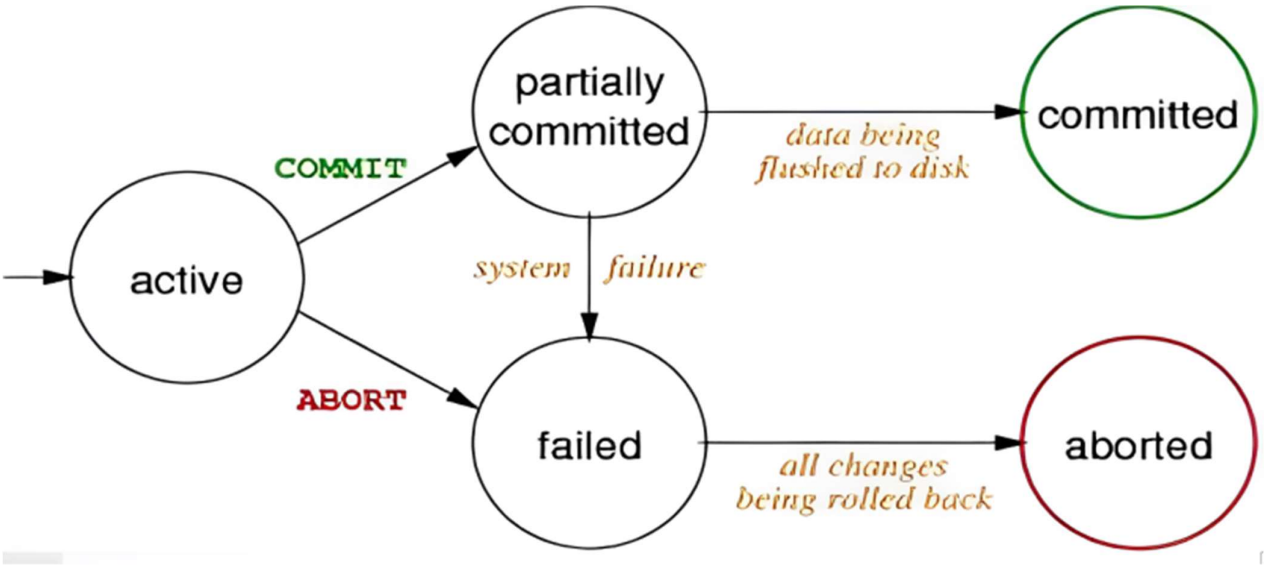
Transaction states

• **PARTIALLY COMMITTED -** After the final statement of a transaction has been executed, the state of transaction is partially committed as it is still possible that it may have to be aborted (due to any failure) since the actual output may still be temporarily residing in main memory and not to disk.

• **FAILED** - After the discovery that the transaction can no longer proceed (because of hardware /logical errors). Such a transaction must be rolled back

• **ABORTED** - A transaction is said to be in aborted state when the when the transaction has been rolled back and the database has been restored to its state prior to the start of execution.

• **COMMITTED -** A transaction enters committed state after successful completion of a transaction and final updation in the database



Q if the transaction is in which of the state that we can guarantee that data base is in consistent state

a) aborted b) committed c) both aborted & committed d) none

Q Match:

a) 1-a, 2–a, 3-b, 4–c

b) 1-a, 2-b,3-b,4-c

c) 1-a,2-a,3-b,4-b

d) none of these

Column I	Column II
1. Atomicity	A. Recovery Manager
2. Durability	B. Concurrency control manager
3. Isolation	C. Programmer
4. Consistency	

WHY WE NEED CONCURRENT EXECUTION

• Concurrent execution is necessary because-

• It leads to good database performance , less weighting time.

• Overlapping I/O activity with CPU increases throughput and response time.

PROBLEMS DUE TO CONCURRENT EXECUTION OF TRANSACTION

• But interleaving of instructions between transactions may also lead to many problems that can lead to inconsistent database.

• Sometimes it is possible that even though individual transaction are satisfying the acid properties even though the final statues of the system will be inconsistent.

1) Lost update problem / Write - Write problem-

T_1	T_2
Read(A)	
Write(A)	
	Write(A)
	Commit
Commit	

• If there is any two write operation of different transaction on same data value, and between them there is no read operations, then the second write over writes the first write.

2) Dirty read problem/ Read -Write problem

T_1	T_2
Read(A)	
Write(A)	
	Read(A)
	Commit
Abort	

• In this problem, the transaction reads a data item updated by another uncommitted transaction, this transaction may in future be aborted or failed.

• The reading transactions end with incorrect results.

Q The problem that occurs when one transaction updates a database item and then the transaction fails for some reason is _____. (NETJUNE-2012)

(A) Temporary Select Problem

(B) Temporary Modify Problem

(C) Dirty Read Problem

(D) None

3) Unrepeatable read problem

• When a transaction tries to read a value of a data item twice, and another transaction updates the data item in between, then the result of the two read operation of the first transaction will differ, this problem is called, Non-repeatable read problem

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

4) Phantom read problem

T_1	T_2
Read(A)	
	Delete(A)
Read(A)	

Q The following schedule is suffering from

a) Lost update problem

b) Unrepeatable read problem

c) Both A and B

d) Neither A and B

T_1	T_2
R(y)	
	R(x) R(y) y = x + y w(y)
R(y)	

Q Which of the following scenarios may lead to an irrecoverable error in a database system? (GATE – 2008) (1 Marks)

(A) A transaction writes a data item after it is read by an uncommitted transaction

(B) A transaction reads a data item after it is read by an uncommitted transaction

(C) A transaction reads a data item after it is written by a committed transaction

(D) A transaction reads a data item after it is written by an uncommitted transaction

SOLUTION IS SCHEDULE

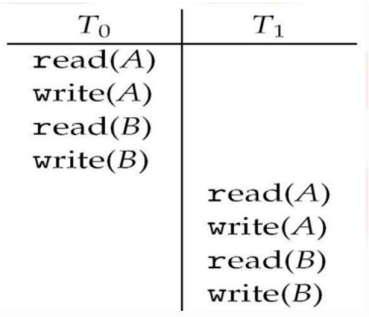
• When two or more transaction executed together or one after another then they can be

bundled up into a higher unit of execution called schedule.

• A schedule of n transactions T1, T2, ..., Tn is an ordering of the operations of the transactions. Operations from different transactions can be interleaved in the schedule S.

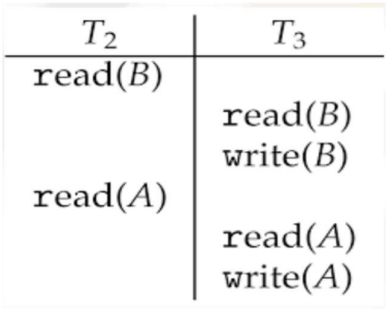
• However, schedule for a set of transaction must contain all the instruction of those transaction, and for each transaction Ti that participates in the schedule S, the operations of Ti in S must appear in the same order in which they occur in Ti

1) **Serial schedule** - A serial schedule consists of sequence of instruction belonging to different transactions, where instructions belonging to one single transaction appear together. Before complete execution of one transaction another transaction cannot be started.



• For a set of n transactions, there exist n! different valid serial schedules. Every serial schedule lead database into consistent state. Throughput of system is less.

2) **Non-serial schedule** - A schedule in which sequence of instructions of a transaction appear in the same order as they appear in individual transaction but the instructions may be interleaved with the instructions of different transactions i.e. concurrent execution of transactions takes place.



• So the number of schedules for n different transaction T1,T2,T3,---------TN where each transaction conations n1, n2, n3, ----------,n4 respectively will be.

• {(n1 + n2+ n3 +----------+n_n)!}/ (n1! n2! n3! ----------n_n!)}

• {(n1 + n2+ n3 +----------+n_n)!}/ (n1! n2! n3! ----------n_n!)} - n!

Conclusion of schedules

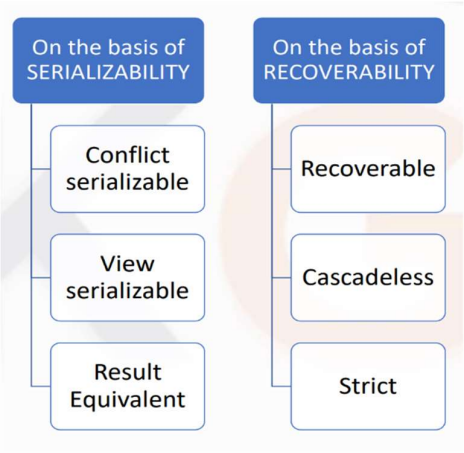
• We do not have any method to proof that a schedule is consistent, but from the above discussion we understand that a serial schedule will always be consistent, so if somehow we proof that a non-serial schedule will also have same effects as of a serial schedule that

we get a proof that, this particular non-serial schedule will also be consistent “find those

schedules that are logically equal to serial schedules”.

• For a concurrent schedule to result in consistent state, it should be equivalent to a serial

schedule. i.e. it must be serializable.



SERIALIZABILITY

• Conflicting instructions - Let I and J be two consecutive instructions belonging to two different

transactions Ti and Tj in a schedule S, the possible I and J instruction can be as-

• I= READ(Q), J=READ(Q) ->Non-conflicting

• I= READ(Q), J=WRITE(Q) ->Conflicting

• I= WRITE(Q), J=READ(Q) ->Conflicting

• I= WRITE(Q), J=WRITE(Q) ->Conflicting

• So, the instructions I and J are said to be conflicting, if they are operations by different

transactions on the same data item, and at least one of these instructions is a write operation.

Conflict equivalent – if one schedule can be converted to another schedule by swapping of nonconflicting instruction then they are called conflict equivalent schedule.

T_1	T_2
R(A)	
A=A-50	
	R(B)
	B=B+50
R(B)	
B=B+50	
	R(A)
	A=A+10

T_1	T_2
	R(B)
	B=B+50
R(A)	
A=A-50	
R(B)	
B=B+50	
	R(A)
	A=A+10

Q Consider a schedule of transactions T_1 and T_2:
Here, RX stands for "Read(X)" and WX stands for "Write(X)". Which one of the following schedules is conflict equivalent to the above schedule?**(Gate-2020) (2 Marks)**

S	
T_1	T_2
RA	
	RB
	WB
RC	
	RD
WD	
	WC
WB	
Commit	
	Commit

a)

S	
T_1	T_2
	RB
	WB
	RD
RA	
RC	
WD	
WB	
	WC
Commit	
	Commit

b)

S	
T_1	T_2
RA	
RC	
WD	
WB	
	RB
	WB
	RD
	WC
Commit	
	Commit

c)

S	
T_1	T_2
RA	
RC	
WD	
	RB
	WB
	RD
WB	
	WC
Commit	
	Commit

d)

S	
T_1	T_2
	RB
	WB
	RD
	WC
RA	
RC	
WD	
WB	
Commit	
	Commit

CONFLICT SERIALIZABLE

• The schedules which are conflict equivalent to a serial schedule are called conflict serializable schedule.

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

• If a schedule S can be transformed into a schedule S' by a series of swaps of non- conflicting instructions, we say that S and S' are conflict equivalent.

• A schedule S is conflict serializable, if it is conflict equivalent to a serial schedule.

Procedure for determining conflict serializability of a schedule

• It can be determined using PRECEDENCE GRAPH method:

• A precedence graph consists of a pair G (V, E)

• V= set of vertices consisting of all the transactions participating in the schedule.

• E= set of edges consists of all edges Ti → Tj, for which one of the following conditions holds:

  • Ti executes write(Q) before Tj executes read(Q)

  • Ti executes read(Q) before Tj executes write(Q)

- Ti executes write(Q) before Tj executes write(Q)

• If an edge Ti → Tj exists in the precedence graph, then in any serial schedule S' equivalent to S, Ti must appear before Tj.

• If the precedence graph for S has no cycle, then schedule S is conflict serializable, else it is not. This cycle detection can be done by cycle detection algorithms, one of them based on depth first search takes O(n2) time.

• The serialializibility order of transactions of equivalent serial schedule can be determined using topological order in a precedence graph.

Q Let $r_i(z)$ and $w_i(z)$ denote read and write operations respectively on a data item z by a transaction T_i. Consider the following two schedules.

- $S_1: r_1(x) r_1(y) r_2(x) r_2(y) w_2(y) w_1(x)$
- $S_2: r_1(x) r_2(x) r_2(y) w_2(y) r_1(y) w_1(x)$

Which one of the following options is correct? **(GATE - 2021) (2 Marks)**

a) S_1 is conflict serializable, and S_2 is not conflict serializable

b) S_1 is not conflict serializable, and S_2 is conflict serializable

c) Both S_1 and S_2 are conflict serializable

d) Neither S_1 nor S_2 is conflict serializable