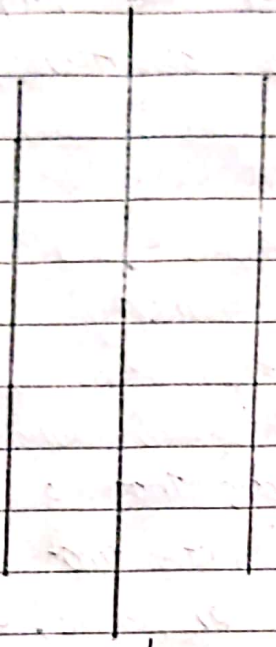
KATHMANDU UNIVERSITY.
Dhulikhel, Kavre.



Assignment 4.

Submitted by :
Ram kaisala.
Roll: 29.
Group: Computer Engineering.

Submitted to :
Dr. Rajani Chulyadyo
Department of Computer
Science and engineering.

Date
Page 1

Q. No. 1.
List the ACID properties. Explain the usefulness of each.

⇒ DBMS is the management of data that should remain integrated when any changes are done on it. It is because if the integrated of data is affected, whole data will get disturbed and corrupted. Therefore, to maintain the integrity of data, there are four properties described in DBMS which are known as ACID properties :-

A → Atomicity
C → Consistency
I → Isolation
D → Durability.

a) Atomicity
This property states that a transaction must be treated as an atomic unit that is either all of its operations are executed or none. ~~Explain~~ Assume that A has initially Rs 500 and B has Rs 300. and A is transferring Rs 100 to B's account. Now it may happen that when A has initiated the transfer, in the midst of transferring from A to B, system fail. Now, balance is deducted from A's account but has not been added to B's account. Here, we need either the transaction executes fully or just revert back to its initial state.

b) Consistency
It ensures that all only valid data following

all rules and constraints is written in database. When a transaction results in invalid data, the database reverts to its previous ~~data~~ state, which abides by all customary rules and constraints. This must be totally ensured by a programmer. Referring to the above example, consistency basically means when a ~~transaction~~ A is rolled back, the sum of balance of both As and B's account are same before and after transaction.

c. Isolation.

It ensures that transaction are securely and independently processed at the same time without interference. For eg, User A withdraws Rs 100 and user Bs withdraw Rs 250 from user C's Account which has a balance of 500. Since, both A and B are withdrawing from C's account, one of the users is required to wait until the other user transaction is completed, avoiding inconsistent data. If they both withdraw at same time, it leads database to an inconsistent state. So, Isolation prevents this from happening by making one transaction wait until other transaction is completed.

d. Durability.

It may happens that after a transaction completes successfully, the changes it has made to the database persist, even if there are system failures. For eg, In above scenario, user B may withdraw $100 only after A's transaction is ~~updated~~ completed and updated in the database. If the system fails before A's transaction

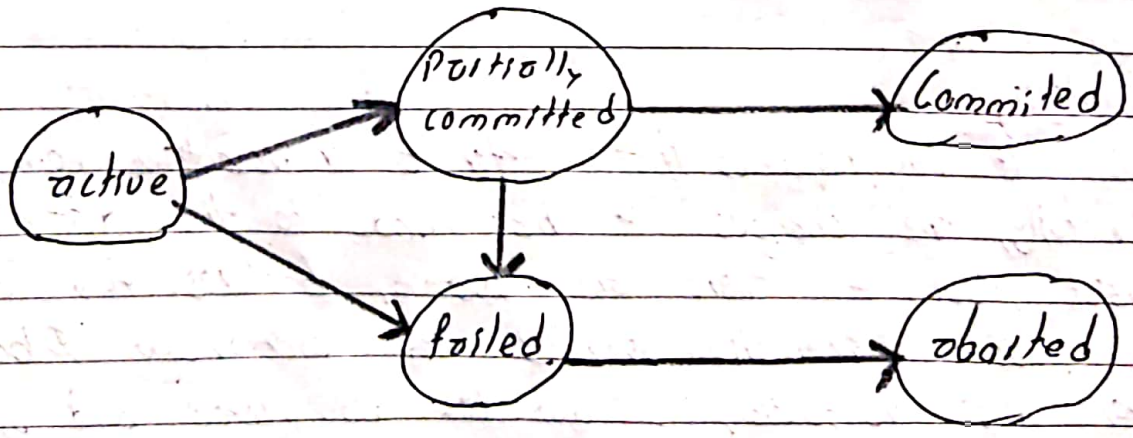
is logged in the database, A cannot withdraw any money and C's amount returns to its previous state. So, durability ensures that changes made to the database transaction that are successfully committed will survive permanently, even in case of system failures.

Q.No.2

List all possible sequence of states through which a transaction may pass. Explain why each state transistion may occur.

States are the situation and condition in which a transaction in DBMS is currently present. There are several states in the transaction such as active, partially committed, committed and abort. During the execution of transaction, the transaction may goes through this several state until it finally commits or abort. The possible sequence of state in transaction are.

i) Active → partially committed → committed.
ii) Active → partially committed → aborted.
iii) Active → failed → aborted.



i) Active → partially committed → committed.

This is the normal sequence which is followed by any normal transaction. In this sequence, first the transaction is in active state. If all the 'read' and 'write' operations are performed without any error, it goes to partially committed state. After enough recovery information has been written to a disk, the transaction finally enters committed state.

ii) Active → partially committed → aborted.

After executing all the transaction present in active state, the transaction enters partially committed state. But before writing enough recovery information on the disk, a hardware failure may occur destroying the memory content. In this case, the changes which it made to the database are undone, and the transaction enters the aborted state.

iii) Active → failed → aborted.

After the transaction starts, if it is discovered at some point, the normal execution cannot continue either due to internal program errors or external errors, it enters the failed stated. It is then rolled back, after which it enters the aborted state.

Q.No.3.
How is automicity implemented in database systems?

→ The recovery manager is responsible for ensuring automicity of the database. Atomicity is guaranteed by undoing the actions of transactions that did not commit (aborted). To ensure atomicity despite failures, we first output information describing the modifications to stable storage without modifying the database itself. To do this, database usually implement some form of logging to track changes.

The database engines logs all the changes and notes when the transaction started and finished. Each changes in the database system causes an update record to be appended to log which contains the identity of data item modified, identification of transaction that did the modification and an copy of data item before update occured. If the transaction completed successfully, we donot need those logs. But in case of failures or crash, we need to recover the database using logs. To do this, we scan the log backwards. If the transaction first record is an update record rather than commit, this transaction was active at time of crash. So, we have to rolled back those transaction.

So, the log records ensures the automicity of the database.

Q.No.4.

What is conflict-serializability? How do you examine and decide whether it is conflict-serializable or not. Explain with an example.

⇒ If a given non-serial schedule can be converted into a serial schedule by swapping its not conflicting operations, then it is called conflict serializable. Let us take two schedule S_1 and S_2;

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

Schedule S_1.

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

Schedule S_2.

Here, Schedule S_1 is not a serial schedule but schedule S_2 is a serial schedule because, in this, all the operations of T_1 are performed before starting any operation of T_2. The schedule S_1 can be transformed into a serial schedule S_2 by swapping non-conflicting operations of S_1. So, S_1 is conflict-serializability.

To check whether a schedule is conflict-serializable or not, we first make the precedence graph. If the precedence graph is cyclic, a schedule is conflict-serializable otherwise not.
let us take a schedule with four transaction, T_1, T_2, T_3 and T_4.

T_1	T_2	T_3	T_4
			Read (A)
	Read (A)		
		Read (A)	
write (B)			
	Write (A)		
		Read (B)	
	write (B)		

First we determine the dependency between transaction.
$T_4 \rightarrow T_2$ (Read (A), write (A))
$T_3 \rightarrow T_2$ (Read (A), write (A))
$T_1 \rightarrow T_3$ (write (B), read (B))
$T_1 \rightarrow T_2$ (write (B), write (B))
$T_3 \rightarrow T_2$ (Read (B), write (B))
Now,
we draw the precedence graph as below:

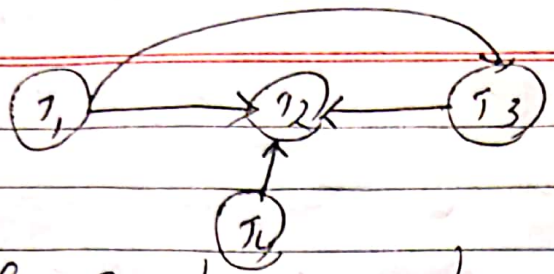


fig: Precedence graph.

Clearly, there exists no cycle, so it is conflict-serializable and all the possible serializable schedule can be found by using topological sort.

1. $T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2$
2. $T_1 \rightarrow T_4 \rightarrow T_3 \rightarrow T_2$.
3. $T_4 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2$.

Q.No.5.
What is recoverable schedule? Why is recoverability of schedules desirable.

⇒ Schedules in which transaction commit only after all transactions whose changes those schedules read commits are called recoverable schedule. In other words, if a transaction T_j reads a data item previously written by a transaction T_i, then the commit operation of T_i appears before the commit operation of T_j.

let us consider two transaction schedules given below:

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

In the above schedule, the transaction T_2 performs dirty read (reading from uncommited transaction) operation on A. The commit operation of T_2 is delayed until transaction T_1 commit or rollback. So, the schedule is recoverable. If the transaction T_1 failed, transaction T_2 has to change to recover by rollback.

Sometimes a transaction may not execute completely due to software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by failed transaction. So, the system is in inconsistent state. So, recoverable schedule are desirable because failure of transaction which otherwise bring the system into an irreversibly inconsistent state.