18K-1256 Sec 'D'

→ Assignment #03:-

→ Q 20.1:-

Concurrent execution of database transaction in a multi-user system is where many number of user can access or use the same data at the same time.

This execution is used in order to deal with the inconsistency in the database.

Let's consider an example, where the two administrators are trying to access the same data, for example, one try to delete the particular data or want to update it, and other also want to access the same data at the same data, then the final result of that particular execution will not be correct if there is no proper way to deal with the simultaneous access of this data.

x ——— x ——— x

Q 20.7

A schedule (or history) $T_1, T_2 \ldots T_n$ is an ordering of the operations of the transaction. Operations from different transactions can be interleaved in the schedule S. Schedule has two types; serial and interleaved scheduling.

→ Recoverable schedule:-

A schedule is said to be recoverable as it should be recoverable, reads operation are allowed before write operations.

S1: R1(x), W1(x), R2(x), R1(y), R2(y), W2(x), W1(y), C1, C2;

In the given example, T_1 executed before T_2. All read operations of the transaction are operated before T_2 (write operations).

⇒ Cascadeless :-

In this scheduling, when no read or write-write occurs before execution of transaction then the schedule is called cascadeless schedule.

S: R1(x), R2(y), R1(z), W1(y), W2(y), C1, C2;

here W1(y) and W2(y) occurs and overwrite and there is no read operation, therefore, it will be called a cascadeless schedule.

⇒ Strict Schedule :-

In this scheduling, if no read or write operation take place before commit, then it is called strict scheduling.

S: R1(x), R2(x), R1(z), R3(x), R3(y), W1(x), C1, W3(y), C3, R2(y), W2(z), W2(y), C2,

In this, no read or write conflict occur arises before commit hence it is called strict scheduling.

Recoverability :-

It should be clear that when transaction is committed, it should not be roll back, this ensures the durability property of transactions. The schedule that reaches this criteria is called recoverable schedules. A schedule where a committed transactions may have to roll back during recovery is called nonrecoverable and should not be permitted by DBMS.

Q 20.8

Conflict Equivalence :-

Two schedule said to be conflict if the order of any two conflicting operation is same in both the schedule.

View schedule :-

The properties of a view schedule can be defined as :-

1) both schedules have same set of transactions.

2) If in a schedule, a read operation $r_1[x]$ of transaction T_1 reads the value of x written by a write operation $w_2[x]$ of transaction T_2 or reads the original value of x, then this must also be in the case of other schedule.

All conflict serializible schedule are view serializable.

Q

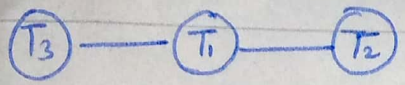
20.23:-

S1:-

This schedule is serializable because T_1 only reads (x) ($r_1(x)$) which is not completely modefied by T_2 and T_3.

$\Rightarrow$ T_3 reads x $r_3(x)$ before T_1 modefies it, $w_1(x)$, T_2 reads y($r_2(y)$) and writes it ($w_2(y)$) only after T_3 has written to it $w_3(y)$.

The Graph of the Serializibility can be given as:-

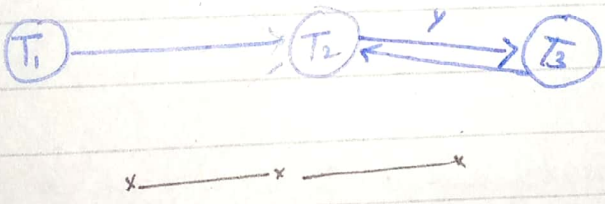


S2:-

This schedule is not serizible because T_2 reads (y) [$r_2(y)$] which is afterwards modified or read by T_3 ($w_3(y)$).

and also T_3 reads y ($r_3(y)$), which then modified before T_2 modifies Y ($w_2(y)$).

T_3 Transaction interferes in the execution of T_1 & T_2, Therefore Graph of this schedule can be given as:-



Q 20.22

Considering that there are T_1, T_2, T_3 Transactions are executing concurrently and produce a Schedule S.

a) The Given schedule is not serializible because x is read by the T_1 ($r_1(x)$) before T_3 and also T_3 reads x before T_1 writes x. Also the T_2 transaction doesn't interupt the T_1 and T_3 execution. In in a serial T_1, T_2 and T_3 the operation $w_1(x)$ comes after $r_3(x)$. which is not happening in the question.

b)
The Given schedule is not serializible because x is read by T_1 before T_3 writes $x(w_3(x))$ before T_1 writes $x(w_1(x))$. T_2 does n't effect the rest transaction, so it has an irrelevant operation.

c) The Given schedule is seriolizable because all conflicting programs of T_3 Transactions happens before all conflicting operations of T_1 & T_2. In this scheduling, T_2 has only one operation which is a read $r_2(x)$ Seriliazible schedule can be given as:

$r_2(x)$; $r_3(x)$; $w_3(x)$; $r_1(x)$; $w_1(x)$ [serial schedule].

d) The Given schedule is not seriziable because T_3 reads $x(r_3(x))$ before T_1 reads $x(r_1(x))$ but $r_1(x)$ happens before T_3 writes $x(w_3(x))$. In a serial, $r_1(x)$ will execute after the execution of $w_3(x)$, which is not the part of this question.

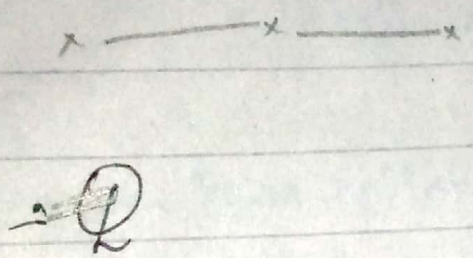
→ Q 20.9:-

Serial Schedule:

Serial schedule can be defined as in which 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.

A schedule is called Serializible whenever the executing the transactions sequentially, in same order, could have left the database in the same data as the actual schedule.

A serializible schedule is accepted as correct because it does not effect the database, by the concurrent execution of the transactions.



: Q 20.12 :

There are four level of isolation that can be defined in SQL as:-

=> Read uncommitted
=> Read commited
=> Repeatable read.
=> Serializable

=> Read uncommitted:- In this level of isolation, read uncommitted has a violation of Dirty Read and nonrepeatable read problem and has a phantom violation.

=> Read committed:- In this level of isolation, there will be no violation of dirty read, and it has a violation of Nonrepeatable read and Phantom.

=> Repeatable Read:- In this level of isolation, there will be no violation of dirty read but will not have a possible violation of non-repeatable read and, Phantom problem.

=> Serializable:- In this level of isolation, there will be no violation of dirty read, have a violation of nonrepeatable read and will also be no violation of Phantom.

Snapshot Isolation:- Another isolation level, known as snapshot isolation, is used in some commerical DBMS, and some concurrency control protocol exist that are based on this concept. The basic definition of snapshot isolation is that a transaction see the data items that it reads based on the commit values of the items in the database snapshot, when the transaction starts.

Snapshot isolation will ensure that the phantom record problem doesn't occur, since the database transaction, or in some cases the database statement, will only see the records that were committed in the database at the time the transaction starts.

x ——— x ——— x

⇒ Q 20.13 :-

The violation caused by dirty read, nonrepeatable, and phantoms can be defined as:-

Dirty read:- Transaction T_1 may read the update of Transaction T_2, which is not committed yet. If T_2 fails and is aborted, then T_1 would have read a value that doesn't exist and is correct.

Non-repeatable read:- There is a violation of read, write-read.
e.g:- if the transaction T_1 reads the value of the data, after that T_2 updates the same value of the data and again T_1 wants to read that value, value will have changed and T_1 will not have a actual value that it read earlier.

Phantoms:- In this violation, if T_1 reads some value of the row that satisfies the given 'where' condition, after that, T_2 enter the new row to the table that also satisfies the 'where' condition then the new enter record would be called Phanton record and it was not there at the start of T_1 but there when the T_1 ends, so T_1 may and may not be able to view it depend on the further transaction.

x ——— x ——— x

20.2(b) T_2
20.3(a), (b)
M=2 N=2

• Q 20.14:-

Answer:-

The only condition when the scenerio would change is the value when $x > 88$.

However, the outcome, however, does obey the implied consistency rule that $x < 90$, since the value of x is not updated if it becomes greater than 90.

X;
read-item(x)
X:X+M;
write-item(X),
if(x>90)
else exit

M=2 N=2
X = read. item()
X = X+M;
write-item(2)

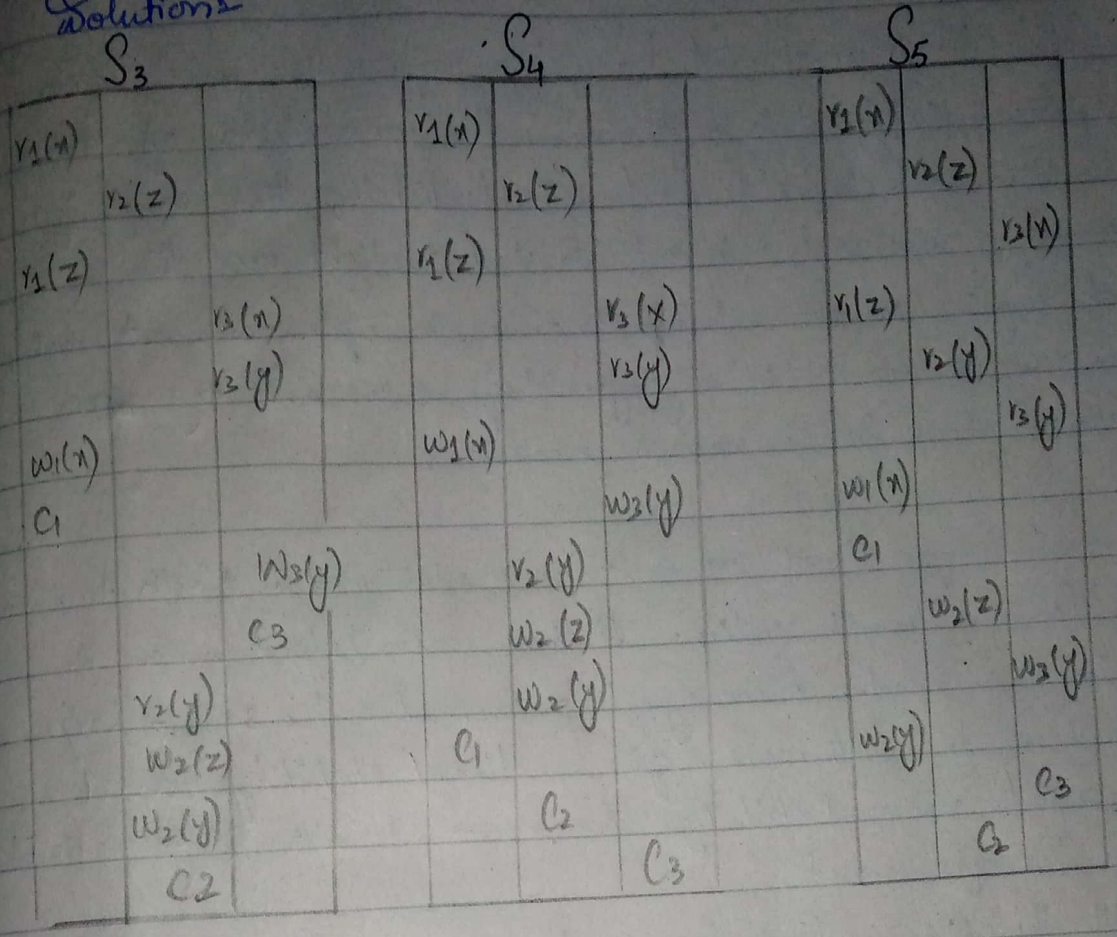
→ Q 20.11 :-

Serializability may be too restrictive, since it requires the enforcement of transaction ordering constraints among all transactions, which may be unnecessary and costly to ensure. Objects which are not yet committed by a transaction should be accessible by other transactions. On the other hand, serializiability doesn't allow synchronization of transaction as a whole.

In database systems, only the interleaving of transactions is synchronized, independently which transactions are executed concurrently.

Q 20.24 :-

Solutions :-



S3 :- recoverable, cascadeless, strict recoverable

S4 :- In this Schedule, there is a dirty read between W3(y) and R2(y) hence not cascadeless, not recoverable because T2 is commiting before T3.

S5 :- No dirty read in this schedule, hence it is recoverable and cascadeless schedule but not strict recoverable.