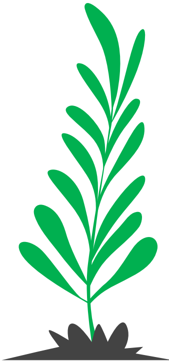
NSCET
E-LEARNING
PRESENTATION

LISTEN … LEARN… LEAD…

II YEAR / IV SEMESTER

CS8492 – DATABASE MANAGEMENT SYSTEMS



A.AMBIKA,M.E,MISTE
ASSISTANT PROFESSOR
Nadar Saraswathi College of Engineering & Technology,
Vadapudupatti, Annanji (po), Theni – 625531.

UNIT- III

TRANSACTIONS

CONTENTS

TRANSACTIONS

- Transaction Concepts
- ACID Properties
- Schedules
- Serializability
- Concurrency Control
- Need for Concurrency
- Locking Protocols

CONTENTS

- Two Phase Locking
- Deadlock
- Transaction Recovery
- Save Points
- Isolation Levels
- SQL Facilities for Concurrency and Recovery.

TRANSACTION CONCEPTS

- A transaction is a unit of program execution that accesses and possibly updates various data items.
- A transaction must see a consistent database.
- During transaction execution the database may be inconsistent.
- When the transaction is committed, the database must be consistent.
- Two main issues to deal with:

Failures of various kinds, such as hardware failures and system crashes Concurrent execution of multiple transactions.

Nadar Saraswathi College of Engineering & Technology

ACID PROPERTIES

ACID Properties:

Atomicity

Either all operations of the transaction are properly reflected in the database or none are.

Consistency

Execution of a transaction in isolation preserves the consistency of the database. !

Nadar Saraswathi College of Engineering & Technology

ACID PROPERTIES

Isolation:

Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.

That is, for every pair of transactions Ti and Tj , it appears to Ti that either Tj , finished execution before Ti started, or Tj started execution after Ti finished. !

Nadar Saraswathi College of Engineering & Technology

ACID PROPERTIES

Durability:

After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures. To preserve integrity of data, the database system failures

EXAMPLE OF FUND TRANSFER

Example of Fund Transfer

Transaction to transfer $50 from account A to account

1. read(A)
2. A := A – 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)

EXAMPLE OF FUND TRANSFER

Consistency requirement:

The sum of A and B is unchanged by the execution of the transaction.

Atomicity requirement:

If the transaction fails after step 3 and before step 6, the system should ensure that its updates are not reflected in the database, else an inconsistency will result.

EXAMPLE OF FUND TRANSFER

Durability requirement:

once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place), the updates to the database by the transaction must persist despite failures.

EXAMPLE OF FUND TRANSFER

Isolation requirement:

if between steps 3 and 6, another transaction is allowed to access the partially updated database, it will see an inconsistent database (the sum A + B will be less than it should be). Can be ensured trivially by running transactions serially, that is one after the other.

However, executing multiple transactions concurrently has significant benefits, as we will see

TRANSACTION STATE

Active, the initial state; the transaction stays in this state while it is executing

Partially committed, after the final statement has been executed.

Failed, after the discovery that normal execution can no longer proceed.

TRANSACTION STATE

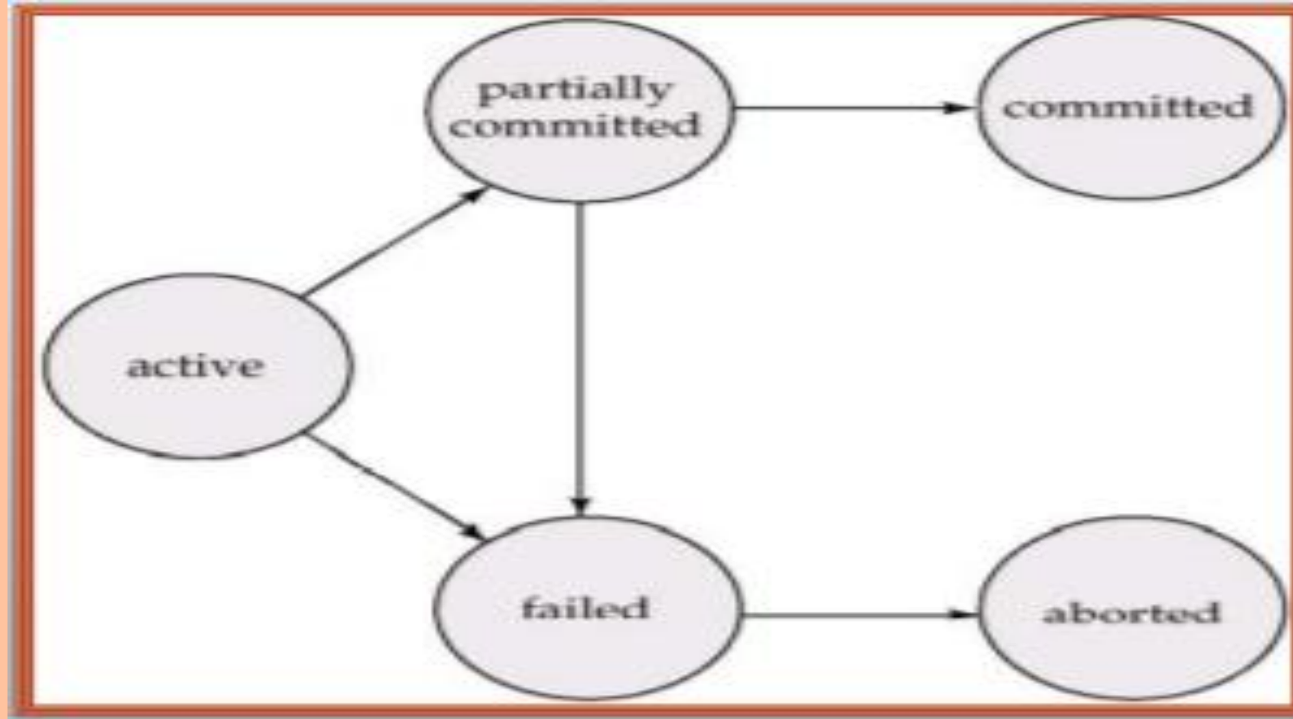
Aborted, After the transaction has been rolled back and the database restored to its state prior to the start of the transaction.

Two options after it has been aborted:

- Restart the transaction – only if no internal logical error
- kill the transaction

Committed, after successful completion

TRANSACTION STATE



IMPLEMENTATION OF ATOMICITY AND DURABILITY

- The recovery-management component of a database system implements the support for atomicity and durability.

The shadow-database scheme:

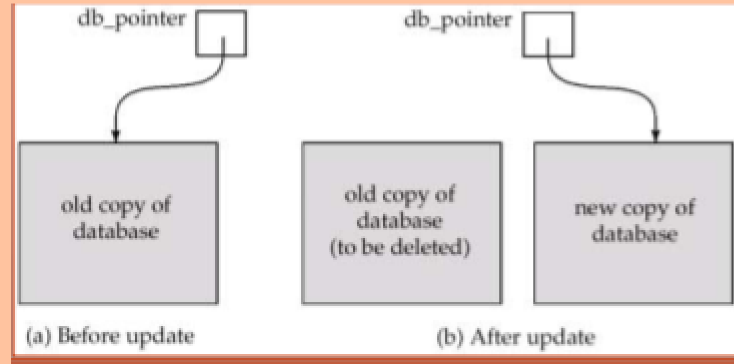
- Assume that only one transaction is active at a time.
- A pointer called db_pointer always points to the current consistent copy of the database.

IMPLEMENTATION OF ATOMICITY AND DURABILITY

- All updates are made on a shadow copy of the database, and db_pointer is made to point to the updated shadow copy only after the transaction reaches partial commit and all updated pages have been flushed to disk.
- In case transaction fails, old consistent copy pointed to by db_pointer can be used, and the shadow copy can be deleted.

IMPLEMENTATION OF ATOMICITY AND DURABILITY

The shadow-database scheme:



- Assumes disks to not fail.
- Useful for text editors, but extremely inefficient for large databases: executing a single transaction requires copying the entire database.

CONCURRENT EXECUTIONS

Multiple transactions are allowed to run concurrently in the system. Advantages are:

- Increased processor and disk utilization,

leading to better transaction throughput: one transaction can be using the CPU while another is reading from or writing to the disk

- Reduced average response time for transactions: short transactions need not wait behind long ones.

CONCURRENT EXECUTIONS

Concurrency control schemes:

Mechanisms to achieve isolation, i.e., to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

SCHEDULES

Sequences that indicate the chronological order in which instructions of concurrent transactions are executed

- A schedule for a set of transactions must consist of all instructions of those transactions.
- Must preserve the order in which the instructions appear in each individual transaction.

EXAMPLE SCHEDULES

Let T1 transfer $50 from A to B, and T2 transfer 10% of the balance from A to B.The following is a serial schedule (Schedule 1 in the text), in which T1 is followed by T2.

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

EXAMPLE SCHEDULE

Example Schedule (Cont.) ! Let T1 and T2 be the transactions defined previously. The following schedule (Schedule 3 in the text) is not a serial schedule, but it is equivalent to Schedule 1.

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

In both Schedule 1 and 3, the sum A + B is preserved.

EXAMPLE SCHEDULE

The following concurrent schedule (Schedule 4 in the text) does not preserve the value of the the sum A + B.

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

SERIALIZABILITY

- Basic Assumption – Each transaction preserves database consistency.
- Thus serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule.

SERIALIZABILITY

Different forms of schedule equivalence give rise to the notions of:

1. conflict serializability
2. view serializability

- We ignore operations other than read and write instructions, and we assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes. Our simplified schedules consist of only read and write instructions

CONFLICT SERIALIZABILITY

- Instructions I i and I j of transactions Ti and Tj respectively, conflict if and only if there exists some item Q accessed by both I i and I j , and at least one of these instructions wrote Q.

  1. I i = read(Q), Ij = read(Q). I i and I j don't conflict.
  2. I i = read(Q), Ij = write(Q). They conflict.
  3. I i = write(Q), Ij = read(Q). They conflict
  4. I i = write(Q), Ij = write(Q). They conflict

Nadar Saraswathi College of Engineering & Technology

CONFLICT SERIALIZABILITY

- Intuitively, a conflict between I i and I j forces a (logical) temporal order between them. If I i and I j are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.
- 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

CONFLICT SERIALIZABILITY

- We say that a schedule S is conflict serializable if it is conflict equivalent to a serial schedule
- Example of a schedule that is not conflict serializable:

T_3	T_4
read(Q)	
	write(Q)
write(Q)	

- We are unable to swap instructions in the above schedule to obtain either the serial schedule < T3, T4 >, or the serial schedule < T4, T3 >.

CONFLICT SERIALIZABILITY

- Schedule 3 below can be transformed into Schedule 1, a serial schedule where T2 follows T1, by series of swaps of non-conflicting instructions. Therefore Schedule 3 is conflict serializable.

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

VIEW SERIALIZABILITY

- Let S and S´ be two schedules with the same set of transactions. S and S´ are view equivalent if the following three conditions are met:

1. For each data item Q, if transaction Ti reads the initial value of Q in schedule S, then transaction Ti must, in schedule S´, also read the initial value of Q.

2. For each data item Q if transaction Ti executes read(Q) in schedule S, and that value was produced by transaction Tj (if any), then transaction Ti must in schedule S´ also read the value of Q that was produced by transaction Tj .

Nadar Saraswathi College of Engineering & Technology

VIEW SERIALIZABILITY

3. For each data item Q, the transaction (if any) that performs the final write(Q) operation in schedule S must perform the final write(Q) operation in schedule S´.

As can be seen, view equivalence is also based purely on reads and writes alone.

VIEW SERIALIZABILITY

A schedule S is view serializable it is view equivalent to a serial schedule.

- Every conflict serializable schedule is also view serializable.
- Schedule 9 (from text) — a schedule which is view-serializable but not conflict serializable.

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- Every view serializable schedule that is not conflict serializable has blind writes.

Nadar Saraswathi College of Engineering & Technology

OTHER NOTIONS OF SERIALIZABILITY

- Schedule 8 (from text) given below produces same outcome as the serial schedule < T1, T5 >, yet is not conflict equivalent or view equivalent to it.

T_1	T_5
read(A) $A := A - 50$ write(A)	
	read(B) $B := B - 10$ write(B)
read(B) $B := B + 50$ write(B)	
	read(A) $A := A + 10$ write(A)

Determining such equivalence requires analysis of operations other than read and write.

Nadar Saraswathi College of Engineering & Technology

RECOVERABILITY

Need to address the effect of transaction failures on concurrently running transactions.

Recoverable schedule: if a transaction Tj reads a data items previously written by a transaction Ti , the commit operation of Ti appears before the commit operation of Tj .

The following schedule (Schedule 11) is not recoverable if T9 commits immediately after the read

RECOVERABILITY

T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

If T8 should abort, T9 would have read (and possibly shown to the user) an inconsistent database state. Hence database must ensure that schedules are recoverable.

RECOVERABILITY

Cascading rollback – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable).

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

If T10 fails, T11 and T12 must also be rolled back. ! Can lead to the undoing of a significant amount of work

RECOVERABILITY

Cascadeless schedules: Cascading rollbacks cannot occur; for each pair of transactions Ti and Tj such that Tj reads a data item previously written by Ti , the commit operation of Ti appears before the read operation of Tj .

- Every cascadeless schedule is also Recoverable.
- It is desirable to restrict the schedules to those that are cascadeless

CONCURRENCY CONTROL

Process of managing simultaneous execution of transactions in a shared database, to ensure the serializability of transactions, is known as concurrency control.

- Process of managing simultaneous operations on the database without having them interfere with one another.
- Prevents interference when two or more users are accessing database simultaneously and at least one is updating data.

Nadar Saraswathi College of Engineering & Technology

CONCURRENCY CONTROL

- Although two transactions may be correct in themselves, interleaving of operations may produce an incorrect result.
- Simultaneous execution of transactions over a shared database can create several data integrity and consistency problems.
- lost updated problem
- Temporary updated problem
- Incorrect summery problem

Nadar Saraswathi College of Engineering & Technology

NEED FOR CONCURRENCY CONTROL

Lost updated problem

- This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.
- Successfully completed update is overridden by another user.

NEED FOR CONCURRENCY CONTROL

Example:

• T1 withdraws £10 from an account with balx, initially £100.

• T2 deposits £100 into same account.

• Serially, final balance would be £190.

•Loss of T2's update!! • This can be avoided by preventing T1 from reading balx until after update

NEED FOR CONCURRENCY CONTROL

T1	T2
Read(X) X:=X-N	
	Read(X) X:=X+M
Write(X) Read(Y)	
	Write(X)
Y:=Y+N Write(Y)	

NEED FOR CONCURRENCY CONTROL

Temporary updated problem

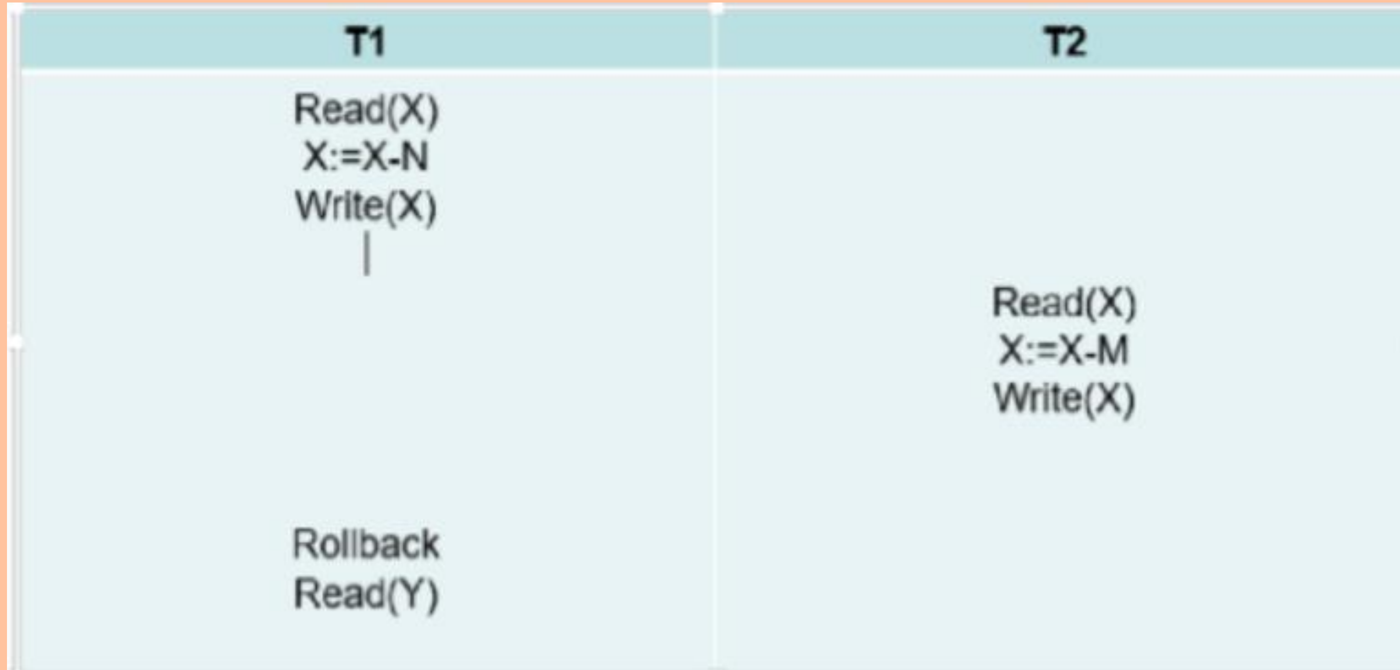
- This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value.
- Occurs when one transaction can see intermediate results of another transaction before it has committed.

NEED FOR CONCURRENCY CONTROL

Example:

- T1 updates balx to £200 but it aborts, so balx should be back at original value of £100.
- T2 has read new value of balx (£200) and uses value as basis of £10 reduction, giving a new balance of £190, instead of £90.
- Problem avoided by preventing T2 from reading balx until after T1 commits or aborts.

NEED FOR CONCURRENCY CONTROL



T1	T2
Read(X)	
X:=X-N	
Write(X)	
	Read(X)
	X:=X-M
	Write(X)
Rollback	
Read(Y)	

NEED FOR CONCURRENCY CONTROL

Incorrect summary problem

- If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.
- Occurs when transaction reads several values but second transaction updates some of them during execution of first

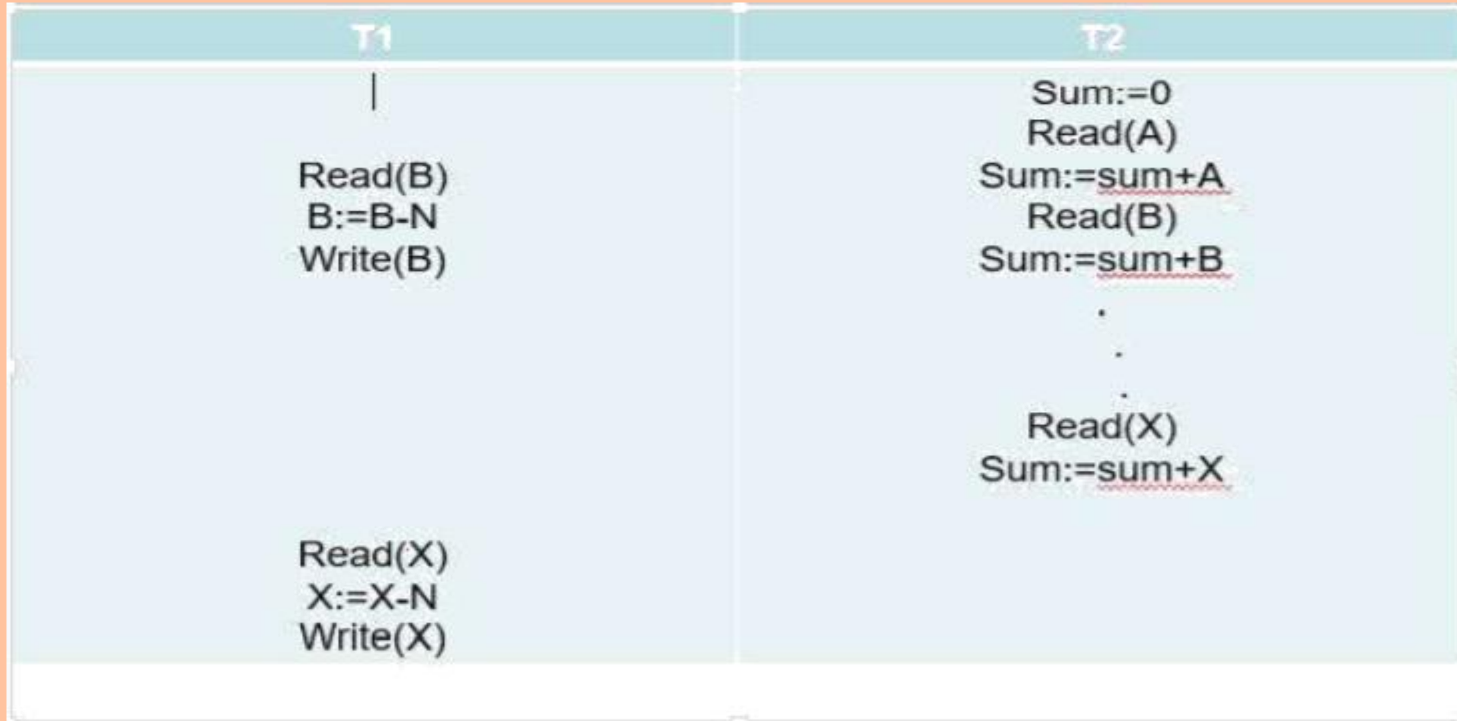
Nadar Saraswathi College of Engineering & Technology

NEED FOR CONCURRENCY CONTROL

Example:

- T6 is totaling balances of account x (£100), account y (£50), and account z (£25).
- Meantime, T5 has transferred £10 from balx to balz, so T6 now has wrong result (£10 too high).
- Problem avoided by preventing T6 from reading balx and balz until after T5 completed updates.

NEED FOR CONCURRENCY CONTROL



T1	T2
|	Sum:=0
	Read(A)
Read(B)	Sum:=sum+A
B:=B-N	Read(B)
Write(B)	Sum:=sum+B
	.
	.
	.
	Read(X)
	Sum:=sum+X
Read(X)	
X:=X-N	
Write(X)	

NEED FOR CONCURRENCY CONTROL

Concurrency control techniques Some of the main techniques used to control the concurrent execution of transaction are based on the concept of locking the data items.

LOCKING PROTOCOLS

A lock is a variable associated with a data item that describe the statues of the item with respect to possible operations that can be applied to it.

Locking is an operation which secures

(a) permission to Read

(b) permission to Write a data item for a transaction.

LOCKING PROTOCOLS

Types of lock

- Binary lock
- Read/write(shared / Exclusive) lock

Binary lock it can have two states (or) values 0 and 1.

0 – unlocked

1 - locked

- Lock value is 0 then the data item can accessed when requested.
- When the lock value is 1,the item cannot be accessed when requested

LOCKING PROTOCOLS

Read / write(shared/exclusive) lock:

- Its also called shared-mode lock.
- If a transaction Ti has obtain a shared-mode lock on item X, then Ti can read, but cannot write ,X. - Outer transactions are also allowed to read the data item but cannot write.

TWO PHASE LOCKING PROTOCOL

This protocol requires that each transaction issue lock and unlock request in two phases

- Growing phase
- Shrinking phase

Growing phase

During this phase new locks can be occurred but none can be released

Shrinking phase

During which existing locks can be released and no new locks can be occurred

TWO PHASE LOCKING PROTOCOL

Types

- Strict two phase locking protocol
- Rigorous two phase locking protocol

Strict two phase locking protocol

This protocol requires not only that locking be two phase, but also all exclusive locks taken by a transaction be held until that transaction commits.

Rigorous two phase locking protocol

This protocol requires that all locks be held until all transaction commits. Consider the two transaction T1 and T2

TWO PHASE LOCKING PROTOCOL

T1 : read(a1);
    read(a2);
    …….
    read(an); write(a1);
T2: read(a1);
    read(a2);
    display(a1+a1);

Lock conversion

- Lock Upgrade
- Lock Downgrade

TWO PHASE LOCKING PROTOCOL

Lock upgrade:

- Conversion of existing read lock to write lock
- Take place in only the growing phase

if Ti has a read-lock (X) and Tj has no read-lock (X) (i!=j)
then convert read-lock (X) to write-lock (X)
else
force Ti to wait until Tj unlocks X

Lock downgrade:

- conversion of existing write lock to read lock
- Take place in only the shrinking phase Ti has a write-lock (X)

DEADLOCK

System is deadlocked if there is a set of transactions such that every transaction in the set is waiting for another transaction in the set.

Consider the following two transactions:

T1: write (A)   T2: write(A)
      write(B)   write(B)

T_1	T_2
lock-X on A write (A)	
	lock-X on B write (B) wait for **lock-X** on A
wait for **lock-X** on B	

DEADLOCK HANDLING

Deadlock prevention protocol

Approach1

– Require that each transaction locks all its data items before it begins execution either all are locked in one step or none are locked.

– Disadvantages

- Hard to predict ,before transaction begins, what data item need to be locked.
- Data item utilization may be very low.

DEADLOCK HANDLING

Approach2 :

–Assign a unique timestamp to each transaction.

–These timestamps only to decide whether a transaction should wait or rollback.

schemes:

- wait-die scheme
- wound-wait scheme

Nadar Saraswathi College of Engineering & Technology

DEADLOCK HANDLING

Deadlock Detection

- Deadlocks can be described as a wait-for graph, which consists of a pair G = (V,E),
  - V is a set of vertices
  - E is a set of edges
- If Ti → Tj is in E, then there is a directed edge from Ti to Tj , implying that Ti is waiting for Tj to release a data item.
- The system is in a deadlock state if and only if the wait-for graph has a cycle. Must invoke a deadlock-detection algorithm periodically to look for cycles.

DEADLOCK HANDLING

Recovery from deadlock

The common solution is to roll back one or more transactions to break the deadlock.

- Three action need to be taken
  - Selection of victim
  - Rollback
  - Starvation

TRANSACTION RECOVERY

Recovery Algorithms

- Recovery algorithms are techniques to ensure database consistency and transaction atomicity and durability despite failures
- Recovery algorithms have two parts

1. Actions taken during normal transaction processing to ensure enough information exists to recover from failures.
2. Actions taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability

TRANSACTION RECOVERY

. Requirement for recovery

- Implicit rollback
- Message handling
- Recovery log
- Statement atomicity
- No nested transaction

Transaction recovery:

Database updates are kept in buffer in main memory and not physically written to disk until commit

Nadar Saraswathi College of Engineering & Technology

TRANSACTION RECOVERY

System recovery

Local failures –affect only the transaction which the failure has actually occurred.

Global failures- affect all the transaction in progress at the time of failure.

System failure – do not physically damage the DB Eg: power shut down Media failure-cause damage to the DB. Eg: head crash ARIES

Nadar Saraswathi College of Engineering & Technology

SAVE POINTS

- It is possible for a transaction to create a savepoint.
- It is used to store intermediate results
- Create: Savepoint <Savepoint name>;
- Rollback: Rollback to <Savepoint name>;
- Drop: Release <Savepoint name> ;

SQL COMMIT: Used to made the changes permanently in the Database.

SAVEPOINT: Used to create a savepoint or a reference point.

ROLLBACK: Similar to the undo operation.

SAVE POINTS

. **ISOLATION LEVEL**

• Degree of interference

• An isolation levels mechanism is used to isolate each transaction in a multi-user environment.

• **Dirty Reads**: This situation occurs when transactions read data that has not been committed.

• **Nonrepeatable Reads**: This situation occurs when a transaction reads the same query multiple times and results are not the same each time.

• **Phantoms**: This situation occurs when a row of data matches the first time but does not match subsequent Times.

SAVE POINTS

Types

Higher isolation level (Repeatable read)

- Less interference
- Lower concurrency
- All schedules are serializable

Lower isolation level(cursor stability)

- More interference
- Higher concurrency
- Not a serializable

THANK YOU