Shortcuts

I/O = Input Output

Transaction Management

A transaction is defined as any one execution of a user program in a DBMS and differs from a execution of a program outside the DBMS

The Asset Property

A DBMS must ensure four important properties of transactions to maintain data in the phase of conquering access and system failures:-

1. Atomic
2. Consistency
3. Isolation
4. Durability

Atomic

Users should be able to regard the execution of each transaction as Atomic i.e either all actions are carried out or none are

Consistency

Each transaction run by itself with no conquering execution of other transactions must preserve the Consistency of the database

Isolation

Users should be able to understand a transaction without considering the effect of other conquer my executing transactions even if the DBMS interleaves the actions of several transactions for performance reasons this property is referred to as Isolation

Durability

Once the DBMS informs the user that a transaction has been successfully completed its effects should persist even if their system crashes before all its changes are reflected on disk. This is called Durability

Transactions and Schedules

- A transaction is seen by the DBMS as a series or list of actions the actions that can be executed by a transaction includes reads and writes of database

objects

- To keep our motions simple we assume that an object O is always read into a program variable that is also named O. We can therefore denote the action of transaction T reading an object O as RT(O). Similarly we can denote WT(O)
- In addition to reading and writing each transaction must specify as its final action either commit(Complete Successfully) or abort(Terminate and Undo all the actions carried out thus far)
- Abort T denotes the action of T aborting and commit T denotes T committing, we make two important assumptions

1. Transactions interact with each other via only database read and write operations. Ex:- They are not allowed to make messages
2. A database is a fixed independent objects when objects are added or deleted from a database or their are relationship with between database objects that we want to exploit for performance some additional issues arise that's when come Schedules come into actions

Schedules

A Schedule is a list of actions(Read, Write Abort or Commit) from a set of transactions and the order in which to actions of a transaction appear in a Schedule must be same as the order in which they appear in T we emphasize that a Schedule describes a transaction as seen by the DBMS

- In addition to these actions a transaction may carry out other action to other operating files evaluating mathematical expressions however these actions do not affect other transactions

T1	T2
R(A)	
W(A)	
	R(B)
	W(B)
R(C)	
W(C)	

Conquering execution of transactions

Now we have the introduced the concept of a Schedule we have convenient way to describe interleave of a transaction

The actions of different transactions but not all interleave should be allowed

Motivation for conquer execution

The Schedule shown in the figure represents a interleaved execution of two transactions

1. While one transaction is waiting for one page to be from disk the cpu can process another transaction this is because I/O can be done in parallel with cpu activity on a computer overlapping I/O and cpu activity reduces the amount of time disks are idle and increases throughput
2. Interleaved execution offer short transaction with a long transaction usually allows the short transaction to complete quickly
3. In serial execution a short transaction could get stuck behind a long transaction leading to understandable delays in response time or average time taken to complete a transaction

Serializability

- A Serializability Schedule over a set S of committed transactions is a Scheduled whose effect on any consistent database instance is guaranteed to be identical to that of some complete Serial Schedule over S. The database that results after execution the given Schedule is identical to the database instance that results from executing the transactions in some serial order
- As an example the Schedule shown in the Figure below is Serailisble

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
	COMMIT
COMMIT	

Even though the action of T1 and T2 the result of the Schedule is equivalent T1 running T2

Anomalies due to interleave execution

- Two actions on the same data object conflict if at least one of them is a write.

- The three Anomalies situation can be described in the terms of when the actions of two transactions T1 and T2 conflict with each other, in a write read conflict(WR Conflict) T 2 previously returned by T1
- We defined WR and WW similarly

WR Conflicts i.e reading uncommitted data

- This occurs when one transaction updates an item but due to unconditional events that transaction fails but before the transaction performs rollback some other transaction reads the updated value thus creates an inconsistency in the database
- Dirty read problem occurs under the scenario WR conflict between the transactions in the database
- The lost update problem can be illustrated with the below scenario between two transactions T1 and T2

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	commit
R(B)	
W(B)	
commit	

The teacher doesn't even want to stop when I am writing stuff what an amazing teacher

Read Write Conflicts

- Read Write Conflict also known as is an computational Anomaly associated with interleave of transactions
- A Read Write Conflict specifically occurs when a transaction requests to read an entity for which unclose transaction has already made a write request

In this example T1 has read the original value of A and is waiting for T2, T2 also reads the original value of A, overwrites A and commits

T1	T2
R(A)	
	R(A)
	W(A)
	commit
R(A)	
W(A)	
commit	

Write Write Conflicts

- Write Write Conflict is a computational anomaly associated with interleave execution of transactions
- specifically a Write Write Conflict occurs when transaction requests to write an entity for which an enclose transaction has already made a write request

Given a Schedule S

T1	T2
W(A)	
	W(B)
W(B)	
commit	W(A)
	commit

Schedules Invloing

- A Serializability Schedule over a Set S of a transactions is Schedule whose effect on any consistent database instance is guaranteed to be identical to that of some complete serial Schedule over a Set of committed transactions in S
- This defination of Serializability relies of aborted transactions been under completely which may be possible in some situations

For Example suppose

1. A account transfer program T1 deducts 100 dollars from account A then
2. An Interest Depoist program T2 reads the current values of accounts A and B and adds 6% interest to each then commits and then
3. T1 is aborted

This Schedule is shown in the below figure

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	commit
abort	

- Now T2 has read a value for A that should never had been there
- If T2 has not yet committed we could deal with this situation by cascading the abort of T1 and also aborting T2 this process recursiolvy aborts any transaction that read data written by T2

Lock based Conquerency Control

- A DBMS must be able to ensure that only Serializabile, revoverable Schedules are allowed and that no actions of committed are lost while undergoing abort transactions
- A DBMS typically uses a locking protocol to achive this
- A Lock is a small book keeping object associated with a database object
- A Locking protocol is a set of rules to be followed by each transaction to ensure that even though actions of several transactions might be interleaved the net effect is identical to executing all transactions in some serial order
- Different Locking protocols use different type of locks such as Shared locks or Exclusive locks or Strict 2PL protocol or Deadlocks

Strict 2PL protocol

- Strict 2 Phase Locking also called as Strict 2PL Locking protocol is the most widely used locking protocol has two rules

1. If a transaction T wants to read an object if first requests a shared lock on the object, a transaction that requests a lock is suspended until the DBMS is able to grant the requested lock
2. All locks held by a transaction is completed are released when the transaction is completed, requests to acquire and relase locks can be automatically inserted into transactions by the DBMS users need not worry about these details

Deadlocks

- Transaction T1 sets an Exclusive on object A, T2 sets an Exclusive lock on B T1 requests and Exclusive lock on B and is queued and T2 requests an Exclusive lock on A and is queued
- Now T1 is waiting is for T2 to release it's lock and T2 is waiting for T1 to release it's lock such a cycle of transactions waiting for locks to be released is called a Deadlock
- Clearly this transactions will make not longer progress worst , they hold locks that maybe required other transactions
- If a transaction has been waiting to long of a lock we can assume that it is in a Deadlock cycle and abort it