Transcation processing

Hi! In this chapter I am going to discuss with you about Transcation processing.

What is concurrency?

Concurrency in terms of databases means allowing multiple users to access the data contained within a database at the same time. If concurrent access is not managed by the Database Management System (DBMS) so that simultaneous operations don't interfere with one another problems can occur when various transactions interleave, resulting in an inconsistent database.

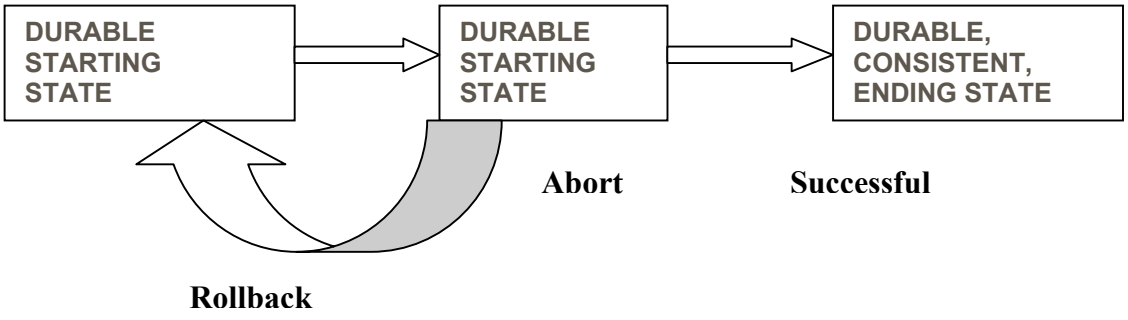
Concurrency is achieved by the DBMS, which interleaves actions (reads/writes of DB objects) of various transactions. Each transaction must leave the database in a consistent state if the DB is consistent when the transaction begins.

Concurrent execution of user programs is essential for good DBMS performance. Because disk accesses are frequent, and relatively slow, it is important to keep the CPU humming by working on several user programs concurrently. Interleaving actions of different user programs can lead to inconsistency: e.g., check is cleared while account balance is being computed. DBMS ensures such problems don't arise: users can pretend they are using a single-user system.

Define Transaction

A transaction is a sequence of read and write operations on data items that logically functions as one unit of work

- It should either be done entirely or not at all
- If it succeeds, the effects of write operations persist (commit); if it fails, no effects of write operations persist (abort)
- These guarantees are made despite concurrent activity in the system, and despite failures that may occur



ACID Properties of Transaction

- **Atomic**
  Process all of a transaction or none of it; transaction cannot be further subdivided (like an atom)
- **Consistent**
  Data on all systems reflects the same state
- **Isolated**
  Transactions do not interact/interfere with one another; transactions act as if they are independent
- **Durable**
  Effects of a completed transaction are persistent

Concurrent Execution

You know there are good reasons for allowing concurrency:-

1. Improved throughput and resource utilization.

(THROUGHPUT = Number of Transactions executed per unit of time.)

The CPU and the Disk can operate in parallel. When a Transaction

Read/Write the Disk another Transaction can be running in the CPU.

The CPU and Disk utilization also increases.

2. Reduced waiting time.

In a serial processing a short Transaction may have to wait for a long transaction to complete. Concurrent execution reduces the average response time; the average time for a Transaction to be completed.

What is concurrency control?

Concurrency control is needed to handle problems that can occur when transactions execute concurrently. The following are the concurrency issues:-

Lost Update: an update to an object by some transaction is overwritten by another interleaved transaction without knowledge of the initial update.

Lost Update Example:-

T_A	T_B
read (R)	
Add 50 to R	
	Read (R)
	Add 100 to R
Write (R)	
	write (R)

Transaction A's update is lost

Uncommitted Dependency: a transaction reads an object updated by another transaction that later falls.

Uncommitted Dependency Example:-

T_A	T_B
read (R)	
Subtract 50 from R	
write (R)	
	read (R)
	Add 75 to R
	write (R)
ROLLBACK	

Transaction B reads an uncommitted value for R

Inconsistent Analysis: a transaction calculating an aggregate function uses some but not all updated objects of another transaction.

Inconsistent Analysis Example:-

T_A	T_B
read (ACC1)	
SUM =30	
read (ACC2)	
SUM = 70	
	read (ACC1)
	ADD 10 to ACC1
	read (ACC3)
	SUBTRACT 10 from ACC3
	COMMIT
read (ACC3)	
SUM = 110 (not 120)	

The value in SUM will be inconsistent

Main goals of Database Concurrency Control

- To point out problem areas in earlier performance analyses
- To introduce queuing network models to evaluate the baseline performance of transaction processing systems
- To provide insights into the relative performance of transaction processing systems
- To illustrate the application of basic analytic methods to the performance analysis of various concurrency control methods
- To review transaction models which are intended to relieve the effect of lock contention
- To provide guidelines for improving the performance of transaction processing systems due to concurrency control; and to point out areas for further investigation.

In the last lecture we discussed about concurrency and transactions. Here I would like to discuss with you in detail regarding the transaction management and concurrency control.

Transaction properties

To ensure data integrity the DBMS, should maintain the following transaction properties- atomicity, consistency, isolation and durability. These properties often referred to as **acid properties** an acronym derived from the first letter of the properties. In the last lecture we have introduced the above terms, now we will see their implementations.

We will consider the banking example to gain a better understanding of the acid properties and why are they important. We will consider a banking system that contains several accounts and a set of transactions that accesses and updates accounts. Access to a database is accomplished by two operations given below:-

1. Read(x)-This operation transfers the data item x from the database to a local buffer belonging to the transaction that executed the read operation
2. Write(x)-the write operation transfers the data item x from the local buffer of the transaction that executed the write operation to the database.

Now suppose that Ti is a transaction that transfers RS. 2000/- from account CA2090 to SB2359. This transaction is defined as follows:-

```
Ti:
Read(CA2090);
CA2090:=CA2090-2000;
Write(CA2090);
Read(SB2359);
SB2359:=SB2359+2000;
Write(SB2359);
```

We will now consider the acid properties..

Implementing Atomicity

Let's assume that before the transaction take place the balances in the account is Rs. 50000/- and that in the account SB2359 is Rs. 35000/-. Now suppose that during the execution of the transaction a failure(for example, a power failure) occurred that prevented the successful completion of the transaction. The failure occurred after the Write(CA2090); operation was executed, but before the execution of Write(SB2359); in this case the value of the accounts CA2090 and SB2359 are reflected in the database are Rs. 48,000/- and Rs. 35000/- respectively. The Rs. 200/- that we have taken from the account is lost. Thus the failure has created a problem. The state of the database no longer reflects a real state of the world that the database is supposed to capture. Such a state is called an inconsistent state. The database system should ensure that such inconsistencies are not visible in a database system. It should be noted that even during the successful execution of a transaction there exists points at which the system is in an inconsistent state. But the difference in the case of a successful transaction is that the period for which the database is in an inconsistent state is very short and once the transaction is over the system will be brought back to a consistent state. So if a transaction never started or is completed successfully, the inconsistent states would not be visible except during the execution of the transaction. This is the reason for the atomicity requirement. If the atomicity property provided all actions of the transaction are reflected in the database of none are. The mechanism of maintaining atomicity is as follows The DBMS keeps tracks of the old values of any data on which a transaction performs a Write and if the transaction does not complete its execution, old values are restored o make it appear as though the transaction never took place. The transaction management component of the DBMS ensures the atomicity of each transaction.

Implementing Consistencies

The consistency requirement in the above eg is that the sum of CA2090 and SB2359 be unchanged by the execution of the transaction. Before the execution of the transaction the amounts in the accounts in CA2090 and SB2359 are 50,000 and 35,000 respectively. After the execution the amounts become 48,000 and 37,000. In both cases the sum of the amounts is 85,000 thus maintaining consistency. Ensuring the consistency for an individual transaction is the responsibility of the application programmer who codes the transaction.

Implementing the Isolation

Even if the atomicity and consistency properties are ensured for each transaction there can be problems if several transactions are executed concurrently. The different transactions interfere

with one another and cause undesirable results. Suppose we are executing the above transaction Ti. We saw that the database is temporarily inconsistent while the transaction is being executed. Suppose that the transaction has performed the Write(CA2090) operation, during this time another transaction is reading the balances of different accounts. It checks the account CA2090 and finds the account balance at 48,000.

Suppose that it reads the account balance of the other account(account SB2359, before the first transaction has got a chance to update the account.

So the account balance in the account Sb2359 is 35000. After the second transaction has read the account balances, the first transaction reads the account balance of the account SB2359 and updates it to 37000.But here we are left with a problem. The first transaction has executed successfully and the database is back to a consistent state. But while it was in an inconsistent state, another transaction performed some operations(May be updated the total account balances). This has left the database in an inconsistent state even after both the transactions have been executed successfully. On solution to the situation(concurrent execution of transactions) is to execute the transactions serially- one after the other. This can create many problems. Suppose long transactions are being executed first. Then all other transactions will have to wait in the queue. There might be many transactions that are independent(or that do not interfere with one another). There is no need for such transactions to wait in the queue. Also concurrent executions of transactions have significant performance advantages. So the DBMS have found solutions to allow multiple transactions to execute concurrency with out any problem. The isolation property of a transaction ensures that the concurrent execution of transactions result in a system state that is equivalent to a state that could have been obtained if the transactions were executed one after another. Ensuring isolation property is the responsibility of the concurrency-control component of the DBMS.

Implementing durability

The durability property guarantees that, once a transaction completes successfully, all updates that it carried out on the database persist, even if there is a system failure after the transaction completes execution. We can guarantee durability by ensuring that either the updates carried out by the transaction have been written to the disk before the transaction completes or information about the updates that are carried out by the transaction and written to the disk are sufficient for

the data base to reconstruct the updates when the data base is restarted after the failure. Ensuring durability is the responsibility of the recovery- management component of the DBMS

Picture

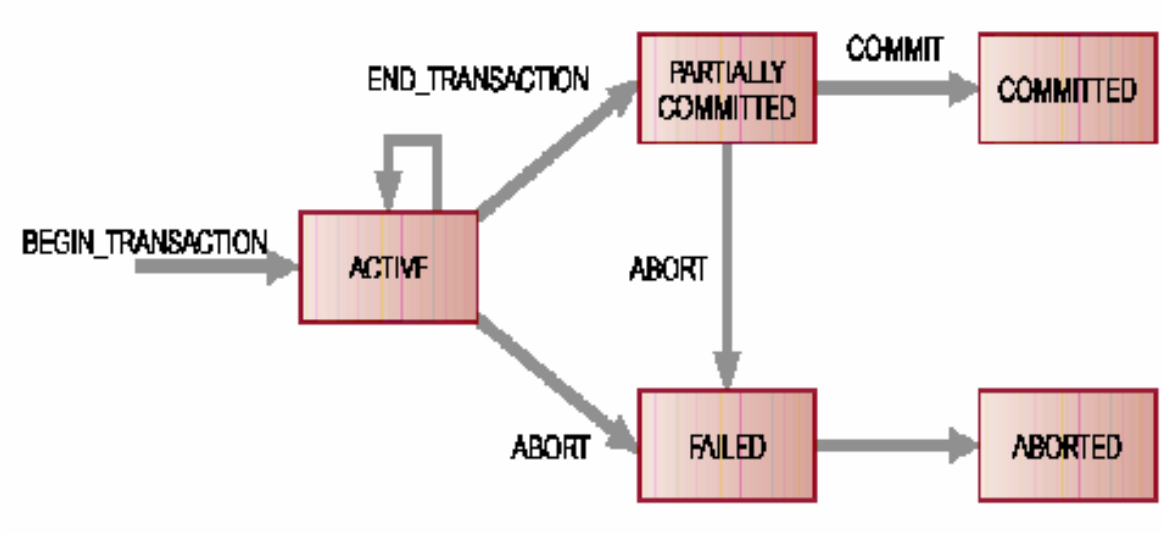
Transaction management and concurrency control components of a DBMS

Transaction States

Once a transaction is committed, we cannot undo the changes made by the transactions by rolling back the transaction. Only way to undo the effects of a committed transaction is to execute a compensating transaction. The creating of a compensating transaction can be quite complex and so the task is left to the user and it is not handled by the DBMS.

The transaction must be in one of the following states:-

1. active:- This is a initial state, the transaction stays in this state while it is executing
2. Partially committed:- The transaction is in this state when it has executed the final statement
3. Failed: - A transaction is in this state once the normal execution of the transaction cannot proceed.
4. Aborted: - A transaction is said to be aborted when the transaction has rolled back and the database is being restored to the consistent state prior to the start of the transaction.
5. Committed: - a transaction is in this committed state once it has been successfully executed and the database is transformed in to a new consistent state. Different transactions states arte given in following figure.



State transition diagram for a Transaction.

Concurrency Control

Concurrency control in database management systems permits many users (assumed to be interactive) to access a database in a multiprogrammed environment while preserving the illusion that each user has sole access to the system. Control is needed to coordinate concurrent accesses to a DBMS so that the overall correctness of the database is maintained. For example, users *A* and *B* both may wish to read and update the same record in the database at about the same time. The relative timing of the two transactions may have an impact on the state of the database at the end of the transactions. The end result may be an inconsistent database.

Why Concurrent Control is needed?

- Several problems can occur when concurrent transactions execute in an uncontrolled manner.
  - The lost update problem : This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of same database item incorrect.
  - The temporary update (or dirty read) problem : This 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.
  - The incorrect summary problem : If one transaction is calculating an aggregate function on a number of records while other transaction is updating some of these

records, the aggregate function may calculate some values before they are updated and others after they are updated.

- Whenever a transaction is submitted to a DBMS for execution, the system must make sure that :
  - All the operations in the transaction are completed successfully and their effect is recorded permanently in the database; or
  - the transaction has no effect whatever on the database or on the other transactions in the case of that a transaction fails after executing some of operations but before executing all of them.

Review Questions

1. Why Concurrent Control is needed?
2. Main goals of Database Concurrency Control.
3. What is concurrency?