CPS221 Lecture - ACID Transactions

Last Revised 10/27/14

Objectives:

1. To introduce the notion of a transaction and the ACID properties of a transaction
2. To introduce the notion of the state of a transaction

Materials:

1. Projectable of transaction states

I. Introduction

A. One vital property of many systems is preserving the integrity of data by dealing with issues that could otherwise lead to its corruption. Two such issues are

1. CONCURRENCY CONTROL: ensuring that the integrity of the data is preserved when it is being simultaneously accessed by more than one user - something we have discussed in some detail in the Operating Systems portion of the course

2. CRASH CONTROL: ensuring that the integrity of the data is preserved even if a system crash (perhaps arising from factors outside the control of the system - e.g. hardware, power, or network failure) should occur while changes are being made.

B. Though these seem like two rather diverse issues, it turns out that a key concept is at the heart of measures for dealing with both of them - the concept of a TRANSACTION.

We will develop this concept in the context of database systems - which is where it first emerged - but as the Hailperin reading pointed out it is also critical in other contexts as well.

C. A preliminary definition: We say that a system is in a CONSISTENT state if there are no contradictions between data stored in it.

1. However, during the course of routine processing, it is sometimes necessary for a system to momentarily enter an inconsistent state

   Example:

   In a banking database, a customer requests a transfer of funds from a checking account to his savings account. Note that this operation results in a change to the balances in both accounts; however, the SUM of the two balances is not changed. Thus, the database is consistent if the sum of the two balances is correct. However, in processing the transfer, it will necessarily be the case that one of the two new balances is written to disk before the other (they can't both be written at exactly the same time if they are stored in two different locations on the same disk.) Thus, during the brief interval between the two write operations the database on disk is actually in an inconsistent state.

2. The system must take measures to ensure that this momentary inconsistency does not become permanent or even visible to other users.

   a) A failure or a crash at this instant could "freeze" that inconsistency.

   b) If another operation were to access the data at this point, that operation would see inconsistent data; and if the operation were performing an operation that updated the database, the inconsistent data might be incorporated into that update.

      Example: suppose the transfer of funds occurs at the exact same moment another process is posting interest to savings accounts. The following might occur:

Transfer Transaction	Interest Posting
Reads savings balance	
	Reads savings balance
Adds transfer amount	
	Computes and adds interest
Writes updated balance	
	Writes updated balance

What happens in this case?

ASK

3. Actually, it is also possible for a pair concurrent transactions to be executed in two different ways that are both consistent, yet produce different results.

   Example: Suppose we do the transfer transaction completely, then the interest posting. In this case, the balance on which the interest is computed includes the transferred amount. On the other hand, if the order is interest posting first, then transfer, the interest balance does not include the transferred amount. Thought these two results differ, we would regard both as consistent, since the difference depends on the relative order of external events.

II. **The Transaction Concept**

A. At the heart of strategies for preventing problems like these is that we conceive of the system's work as basically involving the processing of a series of TRANSACTIONS.

   Each transaction begins with the system in a consistent state, and ends with the database in a consistent state - but may momentarily place the database into an inconsistent state due to the necessity of performing updates one after another.

B. To preserve system consistency, we must guarantee that each transaction satisfies four requirements. These are called the ACID properties, after the first letters of their names.

1. ATOMICITY: We must guarantee that each transaction is processed ATOMICALLY - i.e. either none of it is done, or all of it is done. It must NOT be possible for only part of a transaction to be carried out.

   a) This means that if a transaction is aborted for any reason (due to a logical error in the data or a request by the user, then all effects of the transaction must be removed from the data in the system and the data must be restored to the state it was in before the transaction was begun.

   b) This also means that if a system crash occurs in the middle of processing a transaction, then either:

      (1) Upon system restart, the system must be restored to its state before the transaction was started (in which case the transaction can be restarted from scratch.)

      or

      (2) Upon system restart, the work that was not done because of the crash is completed before any new work is begun.

2. CONSISTENCY: If a transaction is executed in isolation (with no other transactions executing concurrently), and the database is in a consistent state when the transaction starts, then it will still be in a consistent state when it is finished.

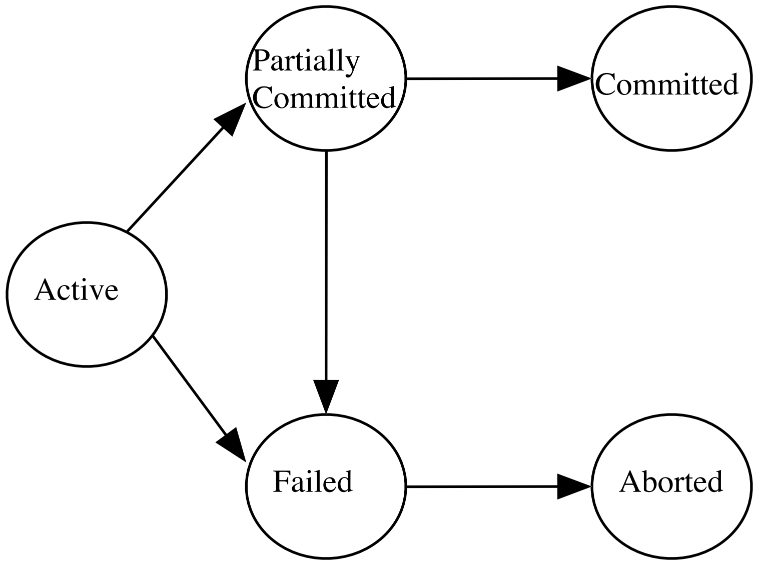
3. ISOLATION: Even if transactions are executing concurrently, the overall result is the same as if they executed serially - i.e. as if each transaction executed in isolation, with one transaction completing before the next begins.

a) This was the problem with our earlier example about transferring money from a checking account to a savings account at the same time interest was being posted to savings accounts. Each transaction was consistent in isolation, but they interacted in such a way as to produce inconsistency.

b) Note that we consider the isolation property to be satisfied if the result is equivalent to ANY serial ordering of the transactions being processed - e.g. as we already noted execution of both a transfer and interest posting would give two different final balances in the savings account if the transfer was done before interest was posted or after it was posted - but either result is acceptable (as long as it applies consistently to all the accounts involved.)

4. DURABILITY: Once a transaction is completed, its effects on the data must persist, even if there is a subsequent system crash. This may mean restoring some data that was destroyed by a crash upon restart.)

C. As a transaction is being processed, it passes through a series of STATES.



PROJECT

1. Active: from the time it starts, until it either fails or reaches its last statement.

2. Partially committed: its last statement has executed, but its changes to system data have not yet been made permanent.

3. Committed: its changes to the database have been made permanent. A soon as a transaction has partially committed, the system attempts to move it to the committed state - though there is no guarantee it will be able to successfully do so. Once a transaction has reached the committed state, the system is obligated preserve its results, even if there is a crash

4. Failed: a logical error or user abort has precluded completion, so any changes it has made to the database must be undone.

5. Aborted: all effects of the transaction have been removed from the system.

6. Some further points to note

   a) There is a one-way connection from partially committed to failed - a partially committed transaction can still fail; but a failed transaction must end up aborted

   b) Externally visible effects of the transaction (those seen by a user) should be deferred until after the transaction is fully committed. These include:

      (1) The writing of messages to the user terminal such as "Transaction complete."

      (2) Changes to data seen by other users concurrently accessing the database.

D. The implementation of the ACID properties in database managment systems is a subject considered in detail in the DBMS - we will not pursue it here beyond three notes:

1. Mechanisms for dealing with concurrency - such as we dealt with earlier in the course - are, of course, vital to guaranteeing isolation.
2. A typical approach to achieving durability is to make use of a LOG - a record of data write operations and the decision to commit a transaction that the system maintains in some form of write-only stable storage. This log can be used to restore data values for a committed transaction in the event of a system crash.
3. Guaranteeing these properties becomes especially complex in a distributed system. We will deal with one aspect of this problem when we talk about distributed algorithms.

E. In practice, because insisting on ACID transactions has performance implications, it is common for some systems to not require that transactions be ACID. This topic is considered in the database course.