Topic: Introduction to Transaction Management

Assigned readings: Sections 14.1 to 14.6 from Silberschatz book

Consistency

An instance of databases is consistent if it satisfies the requirements defined by application. For example, in case of a bank database transfer of funds involves deducting money from the source account and adding the same amount to the destination account. Doing only one of these actions will leave the database in an inconsistent state. Consistency of database is important as it might gave legal, financial and correctness implications.

Atomicity

Abstraction of transaction is required to maintain consistency of database. A DBMS guarantees atomic execution of a transaction. It means that either the whole transaction will be executed or none. Contents of the transaction are designed by the user. It is the user's responsibility to design a transaction that will move database from one consistent state to the other. However, the DBMS guarantees that while executing the transaction, database will remain consistent even if the transaction fails to execute completely.

Durability

Once the transaction completes its work, DBMS stores the results and details of the operations carried out to the permanent storage. This durability of results enables the DBMS to bring the database back to a consistent state in the event of a failure.

Isolation

While executing transactions, a DBMS can maximize the resource utilization by keeping multiple transactions active. It can move from one transaction to the other. However, a DBMS guarantees that this concurrent execution of multiple transactions will not affect any transaction. This isolation guarantee enables a DBMS to ensure that every transaction is shielded from other active transactions.

We will abstract each transaction in terms of four operations: begin, read, write, commit.

Operation "begin" indicates the starting of transaction. With this operation the transaction moves into "Active" state.

An active transaction can "read" or "write" various data values in the database.

When a transaction finishes its last read or write operations, it moves to a partially committed state.

The last operation of any transaction is the commit operation. Completion of this operation guarantees that the results of the transaction of replicated in the permanent storage.

Sometimes an active or a partially committed transaction can fail for various reasons. The DBMS carries out various recovery procedures to abort such failed transactions.

A schedule describes a sequence of operations carried out by multiple transactions. We assume that only one operation can be performed at a time. In short, we assume that there is concurrency but no parallelism. A serial schedule is one in which one transaction is executed after other. Such schedule guarantees isolation for transactions as only one transaction is active at any given instance. Seral schedule also makes it easy for a DBMS to guarantee atomicity, durability, and consistency. However, serial schedules do not allow for efficient resource utilization. For example, one transaction might be waiting for user input. In such situation, we can execute other transaction. But serial schedule fails to do such optimization.

A concurrent schedule executes operations from multiple transaction in an interleaved manner. However, not all concurrent schedules will guarantee consistency and isolation. We focus on a subset of concurrent schedules called conflict serializable schedules.

Two instructions from different transactions conflict with each other if both of them are dealing with same data item and at least one of them needs write access to data item. If two instructions conflict then order of their execution matters, as it might affect the output.

A concurrent schedule is conflict serializable if it can be converted to a serial schedule by shifting execution order of non-conflicting instructions.

One methods is to try this in brute force way. Other way is to create a precedence graph. In the given concurrent schedule, if T1 executes a conflicting instruction before T2 then we will have directed edge from T1 to T2. Create this precedence graph for whole concurrent schedule. If it has a cycle, then the schedule is not conflict serializable. Otherwise do the topological sorting, to get possible equivalent serial schedules.