Transactions

Unit 4.1

Concurrency using Transactions

The goal in a ‘concurrent’ DBMS is to allow multiple users to access the database simultaneously without interfering with each other.

A problem with multiple users using the DBMS is that it may be possible for two users to try and change data in the database simultaneously. If this type of action is not carefully controlled, inconsistencies are possible.

To control data access, we first need a concept to allow us to encapsulate database accesses. Such encapsulation is called a ‘Transaction’.

Transactions

- Transaction (ACID)
  - unit of logical work and recovery
  - atomicity (for integrity)
  - consistency preservation
  - isolation
  - durability
- Available in SQL
- Some applications require nested or long transactions

Transactions cont...

After work is performed in a transaction, two outcomes are possible:

- Commit - Any changes made during the transaction by this transaction are committed to the database.
- Abort - All the changes made during the transaction by this transaction are not made to the database. The result of this is as if the transaction was never started.

Transaction Schedules

A transaction schedule is a tabular representation of how a set of transactions were executed over time. This is useful when examining problem scenarios. Within the diagrams various nomenclatures are used:

- READ(*a*) - This is a read action on an attribute or data item called 'a'.
- WRITE(*a*) - This is a write action on an attribute or data item called 'a'.
- WRITE(*a*[*x*]) - This is a write action on an attribute or data item called 'a', where the value 'x' is written into 'a'.
- tn (e.g. t1,t2,t10) - This indicates the time at which something occurred. The units are not important, but t*n* always occurs before t*n+1*.

Schedules cont...

Consider transaction A, which loads in a bank account balance X (initially 20) and adds 10 pounds to it. Such a schedule would look like this:

Time	Transaction A
t1	TOTAL:=READ(X)
t2	TOTAL:=TOTAL+10
t3	WRITE(X[30])

Schedules cont...

Now consider that, at the same time as trans A runs, trans B runs. Transaction B gives all accounts a 10% increase. Will X be 32 or 33?

Time	Transaction A	Transaction A
t1		BALANCE:=READ(X)
t2	TOTAL:=READ(X)	
t3	TOTAL:=TOTAL+10	
t4	WRITE(X[30])	
t5		BONUS:=BONUS*110%
t6		WRITE(X[BONUS])

Woops… X is 22! Depending on the interleaving, X can also be 32, 33, or 30. Lets classify erroneous scenarios.

Lost Update scenario

Time	Transaction A	Transaction A
t1	READ(R)	
t2		READ(R)
t3	WRITE(R)	
t4		WRITE(R)

Transactopn A's update is lost at t4, because Transaction B overwrites it. B missed A's update at t4 as it got the value of R at t2.

Uncommitted Dependency

Time	Transaction A	Transaction A
t1		WRITE(R)
t2	READ(R)	
t3		**ABORT**

Transaction A is allowed to READ (or WRITE) item R which has been updated by another transaction but not committed (and in this case ABORTed).

Inconsistency Scenario

Time	X	Y	Z	Transaction A		Transaction B
				Action	SUM	
t1	40	50	30	SUM:=READ(X)	40	
t2	40	50	30	SUM+=READ(Y)	90	
t3	40	50	30			READ(Z)
t4	40	50	20			WRITE(Z[20])
t5	40	50	20			READ(X)
t6	50	50	20			WRITE(X[50])
t7	50	50	20			COMMIT
t8	50	50	20	SUM+=READ(Z)	110	
				SUM should have been 120		

Serializability

- A ‘schedule’ is the actual execution sequence of two or more concurrent transactions.
- A schedule of two transactions T1 and T2 is ‘serializable’ if and only if executing this schedule has the same effect as either T1;T2 or T2;T1.

Precedence Graph

In order to know that a particular transaction schedule can be serialized, we can draw a precedence graph. This is a graph of nodes and vertices, where the nodes are the transaction names and the vertices are attribute collisions.

The schedule is said to be serialised if and only if there are no cycles in the resulting diagram.

Precedence Graph : Method

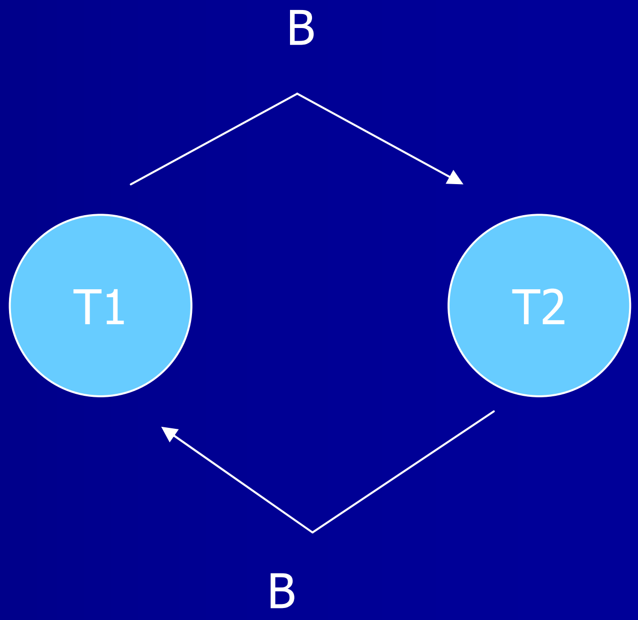
To draw one;

1. Draw a node for each transaction in the schedule
2. Where transaction A writes to an attribute which transaction B has read from, draw an line pointing from B to A.
3. Where transaction A writes to an attribute which transaction B has written to, draw a line pointing from B to A.
4. Where transaction A reads from an attribute which transaction B has written to, draw a line pointing from B to A.

Example 1

Consider the following Schedule:

Time	**T1**	**T2**
t1	READ(A)	
t2	READ(B)	
t3		READ(A)
t4		READ(B)
t5	WRITE(B)	
t6		WRITE(B)



Example 2

Consider the following Schedule:

Time	T1	T2	T3
t1	READ(A)		
t2	READ(B)		
t3		READ(A)	
t4		READ(B)	
t5			WRITE(A)
t6	WRITE(C)		
t7	WRITE(B)		
t8			WRITE(C)

$$\begin{tikzcd}
T1 \arrow[dd, bend right, "C"'] \arrow[dd, "A"] & \\
& T2 \arrow[ul, "B"'] \arrow[dl, "A"] \\
T3 &
\end{tikzcd}$$