WEEK 10

TRANSACTION – CONCURRENCY CONTROL

STUDENT OBJECTIVES

- Upon completion of this video, you should be able to:
  - Identify the 3 main problems with interleaving schedules
  - Distinguish between a serial and a non serial schedule
  - Determine if a schedule is serializable

CONCURRENCY CONTROL

- We want to allow multiple transactions to get at all the data and resources at the same time.
- Objective is to ensure **serializability** of transactions in multi-user database environment (interleaving of transactions where the resulting state is the same as one the states that would occur if the transactions were done serially in some order).
- **Schedule:** A schedule S of n transactions T1, T2, … TN is an ordering of the operations of the transactions. Each operation of an individual transaction Ti in S must appear in the same order that it appears in Ti. The scheduler interleaves the execution of database operations to ensure serializablity

SAMPLE SCHEDULES:

Schedule 1:

Time	Transaction	Step	Stored Value
1	T1	Read Bal	35
2	T1	Bal = 35+100	
3	T1	Write Bal	135
4	T1	Commit	
5	T2	Read Bal	135
6	T2	Bal = 135-30	
7	T2	Write Bal	105
8	T2	Commit	

Schedule 2:

Time	Transaction	Step	Stored Value
1	T2	Read Bal	35
2	T2	Bal = 35-30	
3	T2	Write Bal	5
4	T2	Commit	
5	T1	Read Bal	5
6	T1	Bal = 135-30	
7	T1	Write Bal	105
8	T1	Commit	

Schedule 3:

Time	Transaction	Step	Stored Value
1	T1	Read Bal	35
2	T2	Read Bal	35
3	T1	Bal = 35+100	
4	T2	Bal = 35-30	
5	T1	Write Bal	135
6	T2	Write Bal	5
7	T1	Commit	
8	T2	Commit	

NOTICE THE PROBLEM IN SCHEDULE 3!

CONFLICT

- 2 operations in a schedule ***conflict*** if
  - 1) they belong to different transactions,
  - 2) they access the same data item X
  - 3) at least one of the transactions issues a WRITE(X).

- 3 Main Problems:
  - Lost Updates
  - Uncommitted Data
  - Inconsistent Retrievals

EXAMPLE: LOST UPDATE

- T1: Homer puts in 100 dollars to the savings account (Bal = Bal + 100)
- T2: Marge takes out 30 dollars from the savings account (Bal = Bal - 30)

Correct Schedule

Time	Transaction	Step	Stored Value
1	T1	Read Bal	35
2	T1	Bal = 35+100	
3	T1	Write Bal	135
4	T1	Commit	
5	T2	Read Bal	135
6	T2	Bal = 135-30	
7	T2	Write Bal	105
8	T2	Commit	

Incorrect Schedule:

Time	Transaction	Step	Stored Value
1	T1	Read Bal	35
2	T2	Read Bal	35
3	T1	Bal = 35+100	
4	T2	Bal = 35-30	
5	T1	Write Bal	135 ← Lost Update
6	T2	Write Bal	5
7	T1	Commit	
8	T2	Commit	

COMMITTED DATA

Something has happened here, maybe the ATM rollers broke before the money was sucked in

- hen 2 transactions T1 and T2 are executed, T1 is rolled uncommitted data (violates the Isolation property)
- action starts reading before the rollback is completed

Correct Schedule

Time	Transaction	Step	Stored Value
1	T1	Read Bal	35
2	T1	Bal = 35+100	
3	T1	Write Bal	135
4	T1	Rollback	35
5	T2	Read Bal	35
6	T2	Bal = 35-30	
7	T2	Write Bal	5
8	T2	Commit	

Incorrect Schedule:

Time	Transaction	Step	Stored Value
1	T1	Read Bal	35
2	T1	Bal = 35+100	
3	T1	Write Bal	135
4	T2	Read Bal	135← Uncommitted Data
5	T2	Bal = 135-30	
6	T1	Rollback	35
7	T2	Write Bal	105
8	T2	Commit	

INCONSISTENT RETRIEVALS

- Occurs when a transaction reads some data before they are changed and other data after they are changed.
- For example if T1 calculates the sum of all inventory:

  SELECT SUM(Quantity_on_hand) FROM inventory

  at the same time as T2 is update inventory for some items, the total at the end will be wrong.

SCHEDULER

- Want to ensure that once a transaction T is committed, it should never be necessary to roll back T. (A recoverable schedule)

- Recoverable schedules may have cascading rollbacks that allows uncommitted transactions to be rolled back because it read an item from a transaction that aborted (try to avoid this because it is time consuming!)

- **Scheduler:** A scheduler sets the order that concurrent transactions are executed. It interweaves the operations to ensure serialization.
- A schedule can use a number of methods, we will look at 3: **locking, time stamping** and **optimistic.**
  - It MUST preserve the order of operations within the original transactions
  - It makes some system component schedules
  - It can't really make a planned schedule because it doesn't know what transactions the users are going to submit and in what order.

QUESTION: If the scheduler just had to ensure serialization, what would it always do to ensure this?

ANSWER: Never ever let them mix, always finish one transaction before doing the next transaction

SERIALIZABILITY

- Suppose 2 users (e.g. airline clerks) submit the 2 transactions: T1 and T2 at approximately the same time: 3 resolutions:
  - Execute all the transaction of T1 in sequence followed by T2
  - Execute all the transactions of T2 in sequence followed by T1
  - Some interleaving of operations may be allowed

EXAMPLE

- Say Account A has 1000 and Account B has 2000
  - T1: Transfer $50 from account A to account B
  - T2: Transfer 10% of account A to account B
- NOTE: Even if we do the transactions as atomic units we get 2 different answers depending on the order we execute them!!
- **THAT IS OKAY,** either order: T1 then T2 OR T2 then T1 produces a serial schedule

- **Serializability Theory**: determines which schedules are "correct" and those that are not and tries to allow only correct schedules
- **Serial:** A schedule is serial if transactions are executed consecutively with out interleaving. Every serial schedule is considered correct
- **Non-serial:** A schedule is non-serial if it allows interleaving of transactions

QUESTION: Why not make all schedules Serial?

ANSWER: Too slow, we MUST allow some interleaving

- A schedule of transactions is serializable if it is equivalent to some serial schedule.

QUESTION: Is *result equivalence* (a non-serial schedule that produces the same final database state as a serial schedule) enough?

ANSWER: No! It might have just been by accident!

Example:

If X starts at 100, everything is fine with both schedules but not with other numbers!

Example 1

Schedule 1	Schedule 2
Read X	Read X
X = X + 10	X = X - 6
Write x	Write X
Read X	Read X
X = X - 5	X = X + 11
Write X	Write X

Example 2

Schedule 1	Schedule 2
Read X	Read X
X = X + 10	X = X * 1.1
Write X	Write X

CONFLICT EQUIVALENCE

- **Conflict equivalence:** if the order of any 2 conflicting operations (operations from different transactions, using the same data and one of them is a write operation) is the same in both schedules
- **Testing for Conflict Serializability** (using a serialization graph)
  - for each transaction Ti participating in schedule S create a node labeled Ti in the graph
  - for each case in S where Tj executes a read (X) after a write(X) executed by Ti create an edge (Ti→Tj)
  - for each case in S where Tj executes write(X) after Ti executes a read(X) create an edge (Ti → Tj)
  - for each case in S where Tj executes a write (X) after Ti executes a write(X) create an edge (Ti → Tj)
  - The schedule is serializable if and only if the precedence graph has no cycles

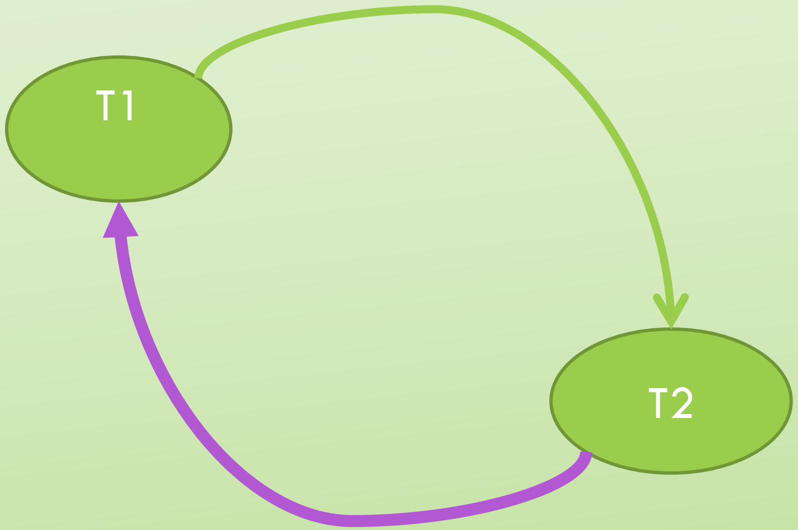
EXAMPLE 1:

Time	Transaction	Graph
T1	Read x	
T1	x = x – n	
T1	Write x	
T1	Read y	
T1	y = y + n	
T1	Write y	
T2	Read x	
T2	x = x + M	
T2	Write x	

Question: Why don’t the red Read and Write of y matter in this case?

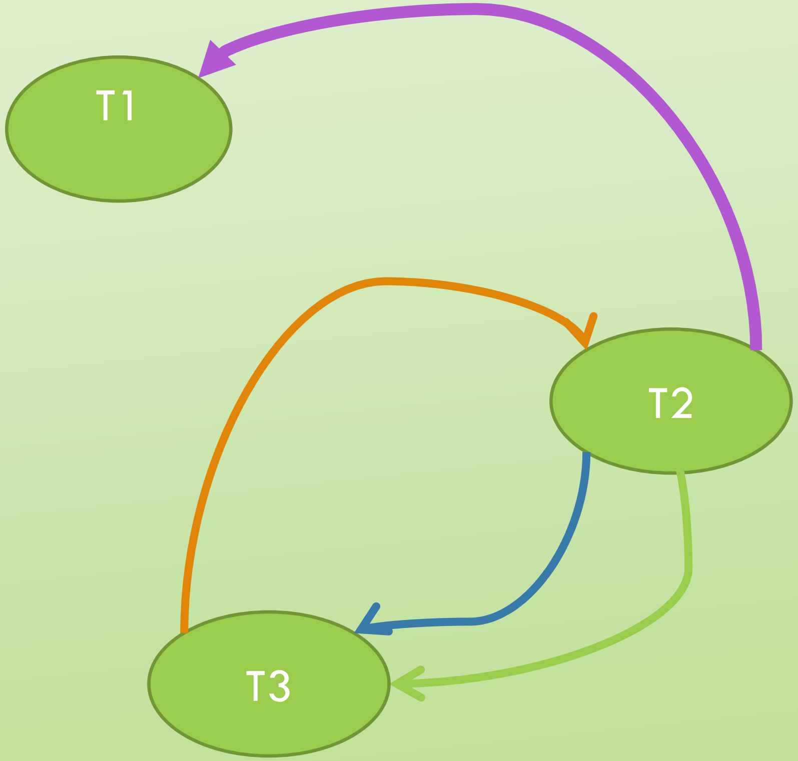
EXAMPLE 2:

Time	Transaction	Graph
T1	Read x	
T1	x = x – n	
T2	Read x	
T2	x = x + M	
T1	Write x	
T1	Read y	
T2	Write x	
T1	y = y + n	
T1	Write y	



EXAMPLE 4:

Time	Transaction	Graph
T2	Read z	
T2	Read y	
T2	Write y	
T3	Read y	
T3	Read z	
T1	Read x	
T1	Write x	
T3	Write y	
T3	Write z	
T2	Read x	
T1	Read y	
T1	Write y	
T2	Write y	



- NOTE: In general, the above test for every schedule is too time consuming, so instead of testing for serializability, we use protocols (sets of rules) that ensure serializability without having to test the schedule, such as 2-phase locking.