Creating safe schedules

Topic 6, Lesson 2 – Serializable and recoverable schedules

Creating safe schedules

Avoiding the:

- Lost Update problem
- Uncommitted Dependency (Dirty Read) problem
- Inconsistent Analysis problem
- Nonrepeatable Read problem
- Phantom Read problem

Scheduling database operations

Schedule: Sequence of reads/writes over a specific time interval performed by a set of transactions.

Objective of a concurrency control protocol is to schedule transactions in such a way as to avoid any interference.

Simplest solution: could run transactions serially, but this limits the degree of concurrency or parallelism in system.

Serial Schedule: Schedule where operations of each transaction are executed consecutively without any interleaved operations from other transactions.

Serializability identifies those executions of transactions guaranteed to ensure consistency of the database.

Nonserial schedule

Schedule where operations from a set of concurrent transactions are interleaved.

Objective of serializability is to find nonserial schedules that allow transactions to execute concurrently without interfering with one another.

In other words, we want to find nonserial schedules that are equivalent to the result of **some** serial schedule. Such a schedule is called **serializable.**

Serializability

- A **serializable schedule** is a schedule whose effect on any consistent database instance is guaranteed to be identical to that of some serial schedule
- In serializability, ordering of read/writes is important:
  - If two transactions only read a data item, they do not conflict, so order is not important.
  - If two transactions either read or write separate data items, they do not conflict, so order is not important.
  - If one transaction **writes a data item and another reads or writes the same data item**, order of execution is important

Conflicting operations in a schedule

Two operations are said to be in conflict, if they satisfy all the following three conditions:

1. Both the operations should belong to different transactions
2. Both the operations are working on the same data item.
3. At least one of the operation is a write operation.

Example of serializability

	Schedule 1		Schedule 2		Schedule 3	
Time	T_7	T_8	T_7	T_8	T_7	T_8
t_1	begin_transaction		begin_transaction		begin_transaction	
t_2	read(bal_x)		read(bal_x)		read(bal_x)	
t_3	write(bal_x)		write(bal_x)		write(bal_x)	
t_4		begin_transaction		begin_transaction	read(bal_y)	
t_5		read(bal_x)		read(bal_x)	write(bal_y)	
t_6		write(bal_x)	read(bal_y)		commit	
t_7	read(bal_y)			write(bal_x)		begin_transaction
t_8	write(bal_y)		write(bal_y)			read(bal_x)
t_9	commit		commit			write(bal_x)
t_{10}		read(bal_y)		read(bal_y)		read(bal_y)
t_{11}		write(bal_y)		write(bal_y)		write(bal_y)
t_{12}		commit		commit		commit
	(a)		(b)		(c)	

Conflict serializability

- A conflict serializable schedule orders any conflicting operations in the same way as some serial execution.
- A schedule is called conflict serializable if we can convert it into a serial schedule after swapping its **non-conflicting operations**.
- Under the *constrained write rule* , a transaction updates a data item based on its old value, which is determined at the first read
- Use a *precedence graph* to test for conflict serializability.

Serializable versus Conflict serializable

Example of a serializable schedule that is not conflict serializable

Time	Transaction 1	Transaction 2	Transaction 3
t_1	R(A)		
t_2		W(A)	
t_3		Commit	
t_4	W(A)		
t_5	Commit		
t_7			W(A)
t_8			Commit

The schedule is serializable since the effect of running this schedule is equivalent to running T1, T2, T3 but the order of the conflicting operations are not in the correct order

Precedence graph

- Create a graph (N,E):
  - where each node is a transaction;
  - a directed edge $T_i \rightarrow T_j$, if T_j reads the value of an item written by T_i;
  - a directed edge $T_i \rightarrow T_j$, if T_j writes a value into an item after it has been read by T_i.
  - a directed edge $T_i \rightarrow T_j$, if T_j writes a value into an item after it has been written by T_i.
- **If precedence graph contains a cycle, then the schedule is not conflict serializable.**

Not conflict serializable schedule

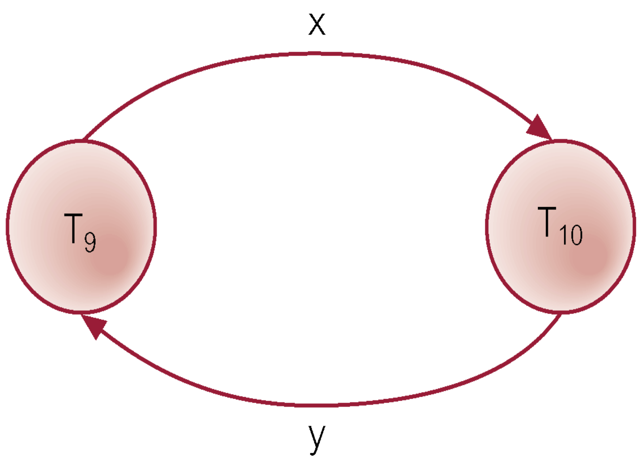
T_9 is transferring £100 from one account with balance bal_x to another account with balance bal_y.

T_{10} is increasing balance of these two accounts by 10%.

Precedence graph has a cycle and so is not serializable.

Example: precedence graph

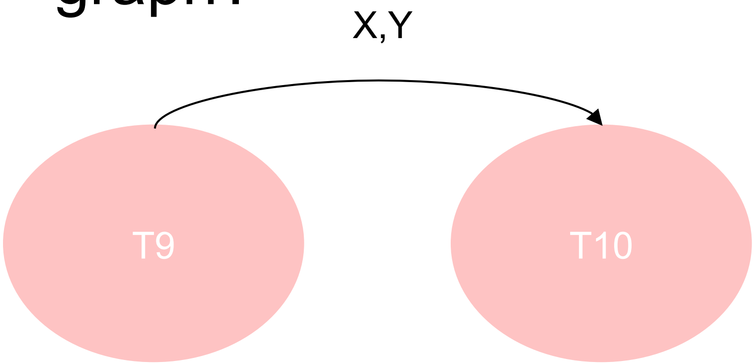
Time	T_9	T_{10}
t_1	begin_transaction	
t_2	read(bal_x)	
t_3	$bal_x = bal_x + 100$	
t_4	write(bal_x)	begin_transaction
t_5		read(bal_x)
t_6		$bal_x = bal_x * 1.1$
t_7		write(bal_x)
t_8		read(bal_y)
t_9		$bal_y = bal_y * 1.1$
t_{10}		write(bal_y)
t_{11}	read(bal_y)	commit
t_{12}	$bal_y = bal_y - 100$	
t_{13}	write(bal_y)	
t_{14}	commit	



Same transactions, different schedule

Time	Transaction 9	Transaction 10
t_1	Begin transaction	
t_2	Read(bal_X)	
t_3	$bal_X = bal_X + 10$	
t_4	Write(bal_X)	Begin transaction
t_5		Read(bal_x)
t_6		$bal_X = bal_x + .1$
t_7		Write(bal_x)
t_8	Read(bal_y)	
t_9	$bal_y = bal_y + 10$	
t_{10}	Write(bal_y)	
t_{11}	Commit	Read(bal_y)
t_{12}		$bal_y = bal_y + .1$
t_{13}		Write(bal_y)
t_{14}		Commit

What is the precedence graph?

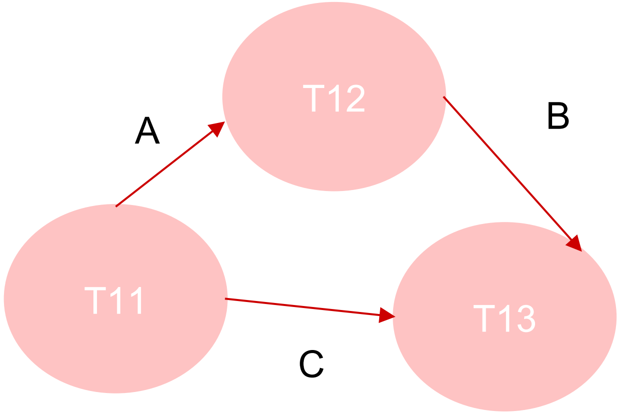


Is this schedule conflict serializable?

Precedence Graph example

Time	Transaction 11	Transaction 12	Transaction 13
t_1	Read(A)		
t_2	A = A + 100		
t_3	Read(C)		
t_4	Write(A)		
t_5	A = A - 40		
t_6		Read(B)	
t_7	Write(C)		
t_8		Read(A)	
t_9			Read(C)
t_{10}		B = B + 40	
t_{11}		Write(B)	
t_{12}			C = C*20
t_{13}			Read(B)
t_{14}			Write(C)
t_{15}		A = A - 2	
t_{16}		Write(A)	
t_{17}			B = B - 8
t_{18}			Write(B)

What is the precedence graph?



Is it conflict serializable?

A recoverable schedule

Serializability identifies schedules that maintain database consistency, **assuming no transaction fails**.

Could also examine recoverability of transactions within a schedule.

If a transaction fails, atomicity **requires that the effects of a transaction is undone.**

Durability states that once transaction commits, its **changes cannot be undone** (without running another, compensating, transaction).

Recoverable schedule

A recoverable schedule is a schedule where, for each pair of transactions T_i and T_j, if T_j reads a data item previously written by T_i, then the **commit** operation of T_i precedes the **commit** operation of T_j.

All reads must be completed on (eventual) committed database objects.

Cascading rollback is when a single transaction failure leads to a series of transaction rollbacks

What if the commit does not occur?

To be recoverable: all reads must be completed on (eventual) committed database objects but what if that producer of the data rollbacks instead of commits?

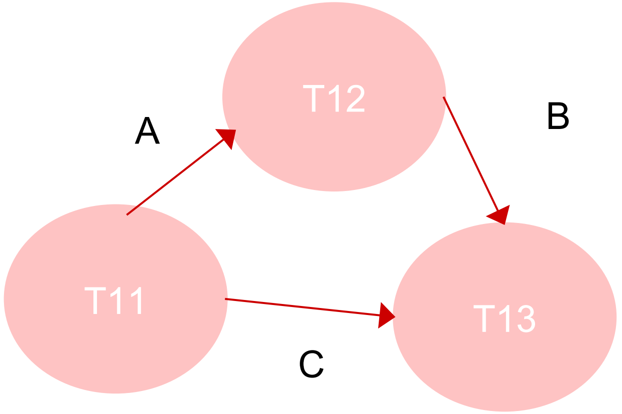
Cascading rollback is when a single transaction failure leads to a series of transaction rollbacks

Cascadeless schedule is one where for each pair of transaction T_i and T_j such that T_j reads data item, previously written by T_i the **commit operation** of T_i appears before the **read operation** of T_j. All cascadeless schedules are recoverable.

Is this Schedule recoverable (A)?

Time	Transaction 11	Transaction 12	Transaction 13
t_1	Read(A)		
t_2	A = A + 100		
t_3	Read(C)		
t_4	Write(A)		
t_5	A = A - 40		
t_6		Read(B)	
t_7	Write(C)		
t_8		Read(A)	
t_9			Read(C)
t_{10}		B = B + 40	
t_{11}	Commit	Write(B)	
t_{12}			C = C*20
t_{13}			Read(B)
t_{14}			Write(C)
t_{15}		A = A - 2	
t_{16}		Write(A)	
t_{17}		Commit	B = B - 8
t_{18}			Write(B)

Precedence graph

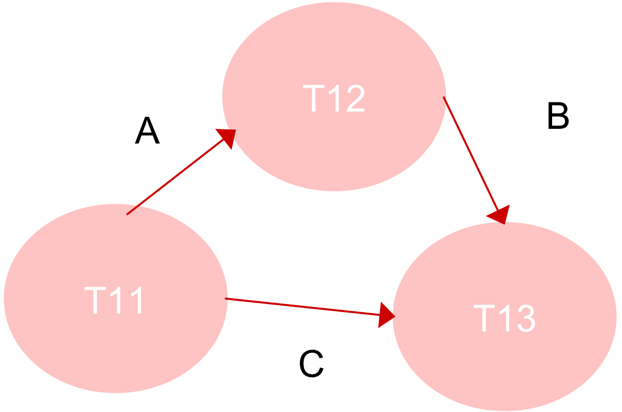


Recoverable
Transaction 11 commits before transaction 12 commits
Transaction 12 commit before transaction 13 commits

Is this Schedule recoverable (B)?

Time	Transaction 11	Transaction 12	Transaction 13
t_1	Read(A)		
t_2	A = A + 100		
t_3	Read(C)		
t_4	Write(A)		
t_5	A = A - 40		
t_6		Read(B)	
t_7	Write(C)		
t_8		Read(A)	
t_9			Read(C)
t_{10}		B = B + 40	
t_{11}		Write(B)	
t_{12}			C = C*20
t_{13}			Read(B)
t_{14}			Write(C)
t_{15}		A = A - 2	
t_{16}		Write(A)	
t_{17}		Commit	B = B - 8
t_{18}	Commit		Write(B)

Precedence graph

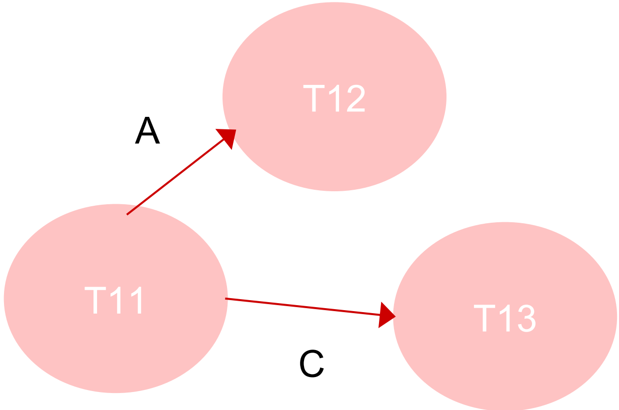


Not recoverable
Transaction 12 commits before transaction 11 commits

Is this Schedule recoverable (C)?

Time	Transaction 11	Transaction 12	Transaction 13
t_1	Read(A)		
t_2	A = A + 100		
t_3	Read(C)		
t_4	Write(A)		
t_5	A = A - 40		
t_6		Read(B)	
t_7	Write(C)		
t_8		Read(A)	
t_9			Read(C)
t_{10}		B = B + 40	
t_{11}		Write(B)	
t_{12}	Rollback		C = C*20
t_{13}			Read(B)
t_{14}			Write(C)
t_{15}		A = A - 2	
t_{16}		Write(A)	
t_{17}		Commit	B = B - 8
t_{18}			Write(B)

Precedence graph

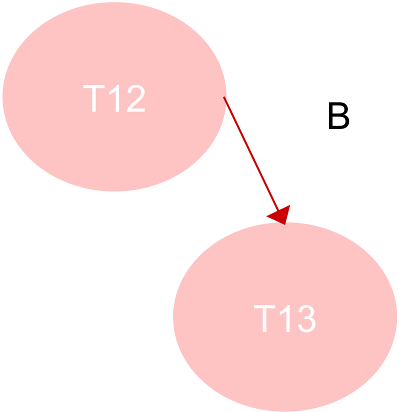


Cascade recoverable
Transaction 11 does not commit so the values of A and B and C are not correct in T12, T13
Remove the results of transaction 11 and restart 12, 13

New schedule for C

Time	Transaction 12	Transaction 13
t_{13}	Read(B)	
t_{24}		
t_{35}	Read(A)	Read(C)
t_{46}		
t_{57}	B = B + 40	
t_{68}	Write(B)	C = C*20
t_{79}		Read(B)
t_{20}		Write(C)
t_{9}		
t_{10}	A = A - 2	
t_{11}	Write(A)	B = B - 8
t_{12}	Commit	Write(B)
t_{13}		
t_{14}		

Precedence graph

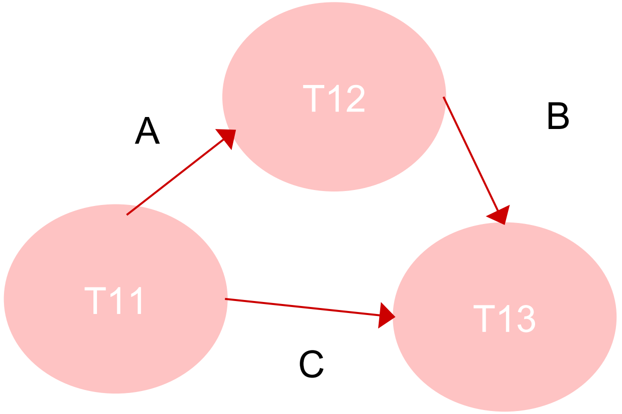


Recoverable
Transaction 12 does commit before T 13

Is this Schedule recoverable ?

Time	Transaction 11	Transaction 12	Transaction 13
t_1	Read(A)		
t_2	A = A + 100		
t_3	Read(C)		
t_4	Write(A)		
t_5	A = A - 40		
t_6		Read(B)	
t_7	Write(C)		
t_8		Read(A)	
t_9			Read(C)
t_{10}		B = B + 40	
t_{11}		Write(B)	
t_{12}			C = C*20
t_{13}			Read(B)
t_{14}			Write(C)
t_{15}		A = A - 2	
t_{16}		Write(A)	
t_{17}		Commit	B = B - 8
t_{18}	Rollback		Write(B)

Precedence graph



Not recoverable or **Irrecoverable – leads to an inconsistent database**
Transaction 11 rollbacks after the transaction 12 commits its changes
Transaction 13 reads C before transaction 11 commits.

Summary

Our goal is to create schedules such that the results of the schedule is as if we ran the transactions in a serial method. This would support the Isolation property.

Conflict serializable schedules can be identified with a precedence graph. If no cycle exists, then the schedule is conflict serializable.

Recoverable schedules require that each data value read from the database has been committed. If a transaction reads a value not committed, then the schedule is not recoverable.