COSC 460 Lecture 15: Transactions 2

Professor Michael Hay
Fall 2018

Transactions

- **Transaction** a sequence of SQL operations treated as a unit.
- Transactions appear to run in isolation
- Transaction either runs to completion or not at all
- ACID Properties
  - Atomicity
  - Consistency
  - Isolation
  - Durability

```
BEGIN TRANSACTION;
insert into archive
(select *
 from apply
 where decision = 'n');
delete from apply
where decision = 'n';
COMMIT;
```

Isolation

- Goal: develop ways to **I**solation. We will worry about A,C, and D later.
- Plan:

    1. See what isolation looks like (serializable schedules, confict-serializable schedules, …)

    2. See how to ensure isolation (locking protocols)

Schedules

- A **schedule** specifies a sequence (ordering) of statements across a set of transactions.
  - Statements *within same* transaction must execute in order.
  - Statements from *different* transactions may be interleaved
- **Serial** schedule: no interleaving

S1

Serial:
T1, T2

T1	T2	Intermediate results
		A = B =1000
A = A+100		A = 1100
B = B-100		B = 900
	A = A*1.01	A = 1111
	B = B*1.01	B = 909

End result:
A = 1111; B = 909

A+B = 2020

S2

Serial:
T2, T1

T1	T2	Intermediate results
		A = B =1000
	A = A*1.01	A = 1010
	B = B*1.01	B = 1010
A = A+100		A = 1110
B = B-100		B = 910

End result:
A = 1110; B = 910

A+B = 2020

Serializable schedules

- A **schedule** specifies a sequence (ordering) of statements across a set of transactions.

- Schedule S' is **equivalent** to schedule S if the result of executing S’ on D is the same as the result of executing S on D.

- Schedule S’ is **serializable** if it is equivalent to some serial schedule S.

S3

Serializable: equivalent to T1, T2

T1	T2	Intermediate results
		A = B =1000
A = A+100		A = 1100
	A = A*1.01	A = 1111
B = B-100		B = 900
	B = B*1.01	B = 909

End result:
A = 1111; B = 909

A+B = 2020

S4

Not *serializable*

T1	T2	Intermediate results
		A = B =1000
A = A+100		A = 1100
	A = A*1.01	A = 1111
	B = B*1.01	B = 1010
B = B-100		B = 910

End result:
A = 1111; B = 910

A+B = 2021!!

Abstract Schedule: Reads and Writes

S4

A = A+100

A = A*1.01

B = B*1.01

B = B-100

Abstract version

R(A)

W(A)

R(A)

W(A)

R(B)

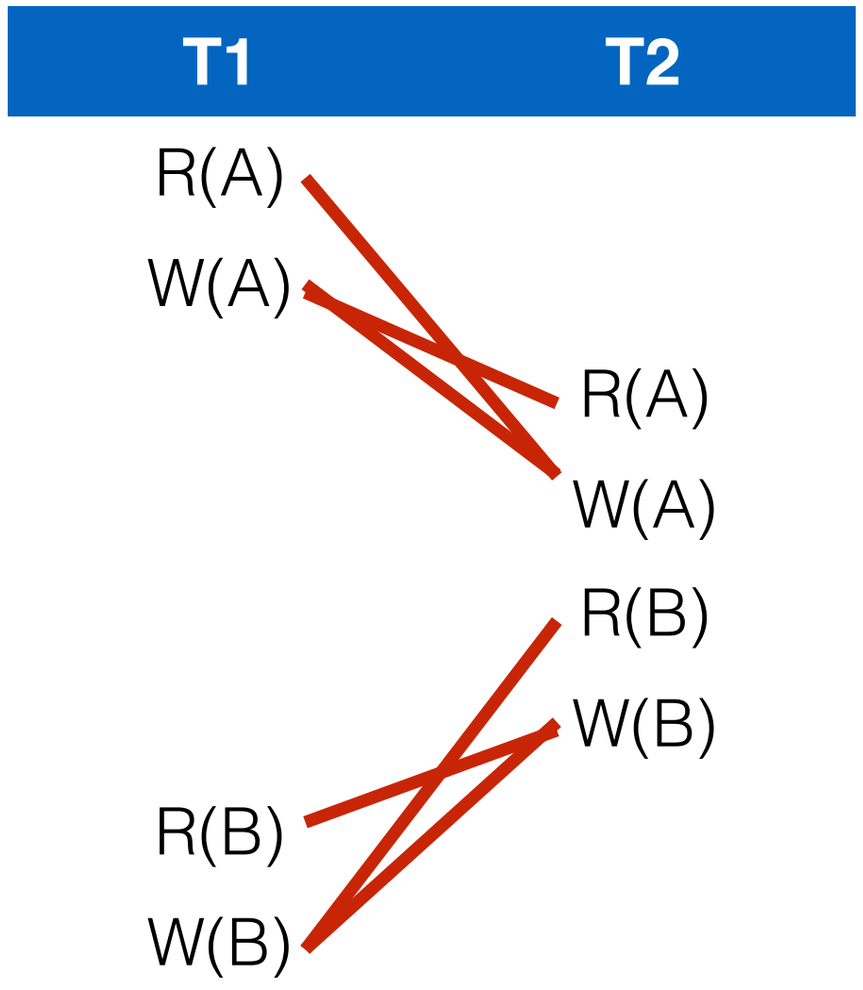
W(B)

R(B)

W(B)

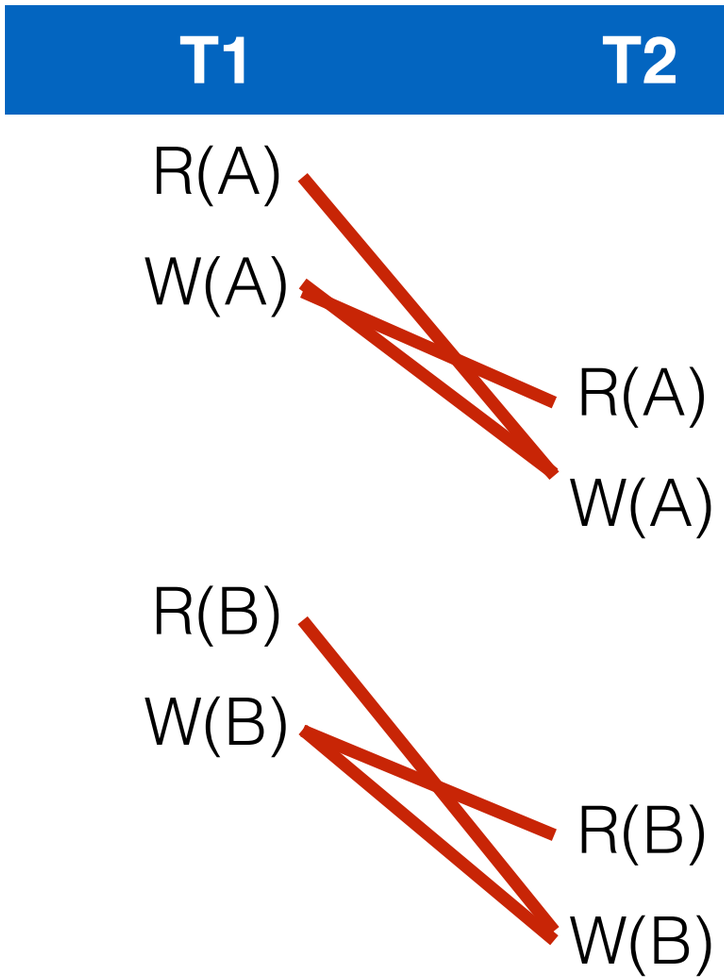
Conflicting statements

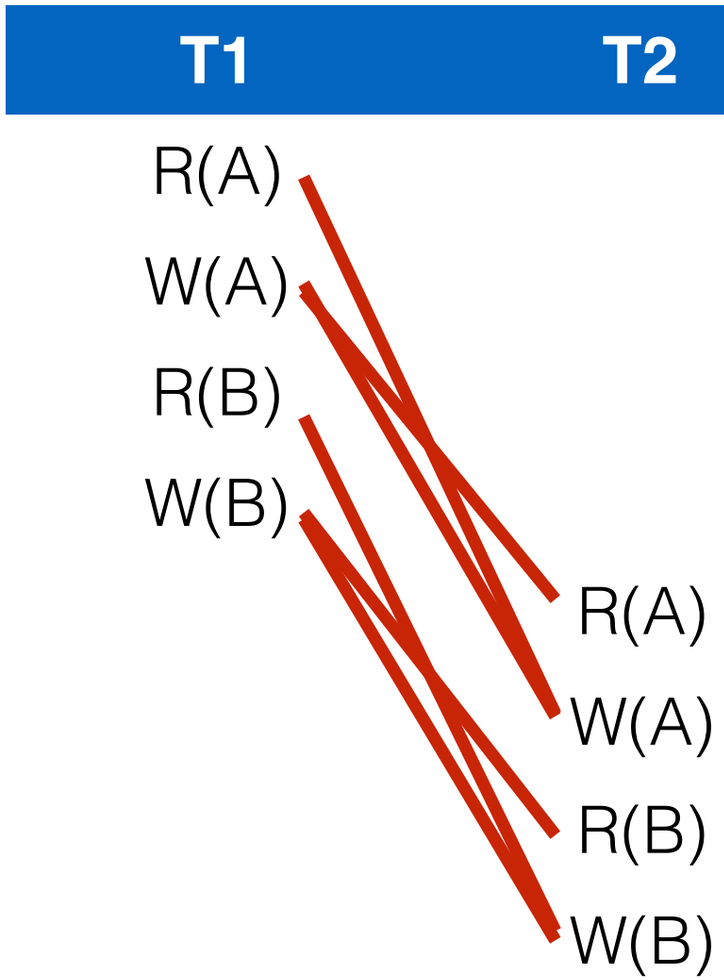
- Two statements **conflict** if….
  - They are from different transactions
  - They access same object
  - At least one is a write



Conflict equivalent

- Schedule S is **conflict equivalent** to S’ if every pair of conflicting statements is ordered in the same way





Question

- Two statements **conflict** if….
  - They are from different transactions
  - They access same object
  - At least one is a write

- Are these conflict equivalent schedules?

Instructions: ~1 minute to think/answer on your own; then discuss with neighbors; then I will call on one of you

Conflict serializable

- Recall: schedule S' is serializable if it is *equivalent* to some serial schedule S.

- A schedule S' is **conflict serializable** if it is *conflict equivalent* to serial schedule S.

Question

Instructions: ~1 minute to think/ answer on your own; then discuss with neighbors; then I will call on one of you

Let's think about the relationship between *conflict serializability* and *serializability.*

Consider the schedule shown here.

- Is it *conflict serializable*?
- Is it *serializable*?

Checking for conflict serializability

- Construct **precedence graph**
  - Node for every transaction
    - Edge $T_i \rightarrow T_j$ if T_i has statement *a* that conflicts with a statement *b* in T_j and *a* comes before *b*
- If precedence graph is acyclic, then a schedule is conflict serializable.

Examples schedules

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
R(B)	
W(B)	

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

Question

Instructions: ~1 minute to think/ answer on your own; then discuss with neighbors; then I will call on one of you

Construct the *precedence graph* for the schedule shown here.

Is this schedule *conflict serializable?*

Example

- T1 transfers between accounts
- T2 displays total account balance
- Is this schedule *serializable*?

T1	T2
L(A)	
	L(A)
A=A+100	|
U(A)	|
	L(B)
	print(A+B)
	U(A), U(B)
L(B)	
B = B - 100	
U(B)	

Locking alone is not enough! Need ***locking protocol***.

Example

Instructions: ~1 minute to think/ answer on your own; then discuss with neighbors; then I will call on one of you

- Is this schedule feasible under the *2PL protocol*?
- If not, why not?
- If so, is it a *serializable schedule*?

T1	T2
	L(A), L(B)
	temp = A
	U(A)
L(A)	
A=A+100	
U(A)	
	temp += B
	U(B)
L(B)	
B = B - 100	
U(B)	
	print(temp)