CMPSC 174A: Section 8 Transactions

March 3rd, 2021

Administrivia

HW5 Deadline: March 9th, 2021 11:00 PM PT

HW5 - Concurrent

```
from concurrent.futures import ProcessPoolExecutor
import os

def task():
    print("Executing our Task on Process: {}".format(os.getpid()))

def main():
##    executor = ProcessPoolExecutor(max_workers=3)
    with ProcessPoolExecutor(max_workers=3) as executor:
        task1 = executor.submit(task)
        task2 = executor.submit(task)

main()
```

HW5 - apsw Transaction

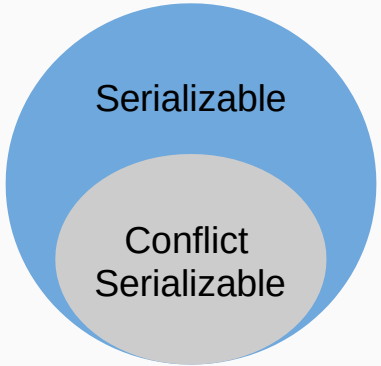
```
# this will be 3 separate transactions
db.cursor().execute("INSERT ...")
db.cursor().execute("INSERT ...")
db.cursor().execute("INSERT …")


# this will be one transaction
db.cursor().execute(“BEGIN”)      // <<<-
db.cursor().execute("INSERT ...")
db.cursor().execute("INSERT ...")
db.cursor().execute("INSERT ...")
db.cursor().execute(“COMMIT”)    //. <<<-
```

Serializability

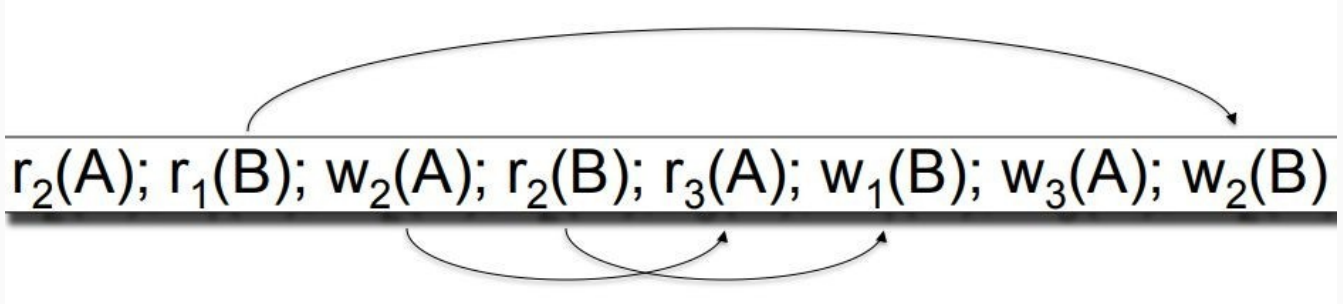
Conflict serializable is stricter than serializable

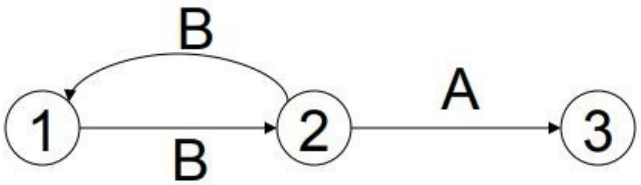
I.e. Any schedule that is conflict serializable must be serializable.



Serializability

Checking for conflict serializability -> precedence graph and cycle checking





Serializability

S1: w1(Y); w2(Y); w1(X); w2(X); w3(X)

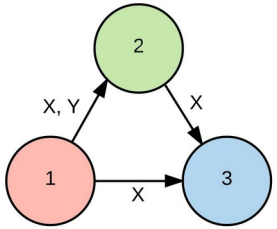
Are these serializable?
Conflict serializable?

S2: w1(Y); w2(Y); w2(X); w1(X); w3(X)

Serializability

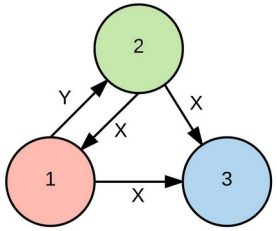
S1: w1(Y); w2(Y); w1(X); w2(X); w3(X)

Conflict Serializable



S2: w1(Y); w2(Y); w2(X); w1(X); w3(X)

Serializable (but not conflict serializable)



2PL v.s. Strict 2PL

2P:

- In every transaction, all lock requests must precede all unlock requests
- Ensure Conflict Serializability
- Might not be able to recover (Dirty Read: Read on some write that gets rolled back)

Strict 2PL:

- Every lock each transaction is held until commit or abort
- Ensure Conflict Serializability
- Recoverable as each transaction does not affect others until commit/abort

2PL v.s. Strict 2PL

A New Problem: Non-recoverable Schedule

T1	T2
$L_1(A)$; $L_1(B)$; READ(A)	
A :=A+100	
WRITE(A); $U_1(A)$	
	$L_2(A)$; READ(A)
	A := A*2
	WRITE(A);
	$L_2(B)$; BLOCKED…
READ(B)	
B :=B+100	
WRITE(B); $U_1(B)$;	
	…GRANTED; READ(B)
	B := B*2
	WRITE(B); $U_2(A)$; $U_2(B)$;
	Commit
Rollback	

Isolation Level: Read Uncommitted

Write Locks? Strict 2PL

Read Locks? No (Immediate Read)

Problem: Dirty-Read

Reading uncommitted data that can be rolled back

Isolation Level: Read Uncommitted

Example Transaction

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

T2 is reading value of A updated by T1's write on A, but T1 has not committed yet.

The value of A read by T2 might not even be in the result.

Then T2's action can be influenced by such uncommitted data.

Isolation Level: Read Committed

Write Locks? Strict 2PL

Read Locks? Obtain before read, release after (No more dirty read)

Problem: Unrepeatable Read

The values of 2 reads on the same tuple can be different in the same transaction

Isolation Level: Read Committed

Example Transaction:

$$S = \begin{bmatrix} T1 & T2 \\ R(A) & \\ & R(A) \\ & W(A) \\ & Com. \\ R(A) & \\ W(A) & \\ Com. & \end{bmatrix}$$

T1’s first R(A) and T1’s second R(A) might have different results.

Updated by T2’s W(A).

Isolation Level: Repeatable Read

Write Locks? Strict 2PL

Read Locks? Strict 2PL (No more unrepeatable read) Same as Serializable if no insert or delete

Problem: Phantom Read

In the same transaction, some tuples appear sometimes and

Isolation Level: Repeatable Read

Suppose there are two blue products, A1, A2:

Phantom Problem

T1	T2
SELECT * FROM Product WHERE color='blue'	
	INSERT INTO Product(name, color) VALUES ('A3','blue')
SELECT * FROM Product WHERE color='blue'	

Isolation Level: Serializable

Not the same thing as Serializable schedule!!!

Write Locks: Strict 2PL

Read Locks: Strict 2PL

Predicate Lock/Table Lock (No Phantom)

Isolation Level: Serializable

Predicate Lock Example:

In Transaction T, we have a statement:

SELECT * FROM People WHERE age > 18;

In this case, the transaction will grab a predicate lock that prevent inserting and deleting tuples that can affect the predicate/statement.

In this case, the lock prevents inserting and deleting tuples with age > 18.

Isolation Level: Summary

Isolation Level	Dirty Reads	Nonrepeatable Reads	Phantoms
Read Uncommitted	Allowed	Allowed	Allowed
Read Committed	Not Allowed	Allowed	Allowed
Repeatable Read	Not Allowed	Not Allowed	Allowed
Serializable	Not Allowed	Not Allowed	Not Allowed