TRANSACTIONS

week 6

- sequence of actions
- either all or none

Autocommit off → first SQL query starts txn

```
Start transaction
[SQL]
COMMIT or ROLLBACK (=ABORT)
```

→ if app cannot continue because of integrity cond., dbms will rollback

ABORT reasons → ctrl+c
→ condition fails
→ deadlock (system can abort)

Without transactions

Lost Updates

```
Upd Cust
Set rental+=1
where name="Fred"
```

Q1

```
Some query
in the left
```

Q2

We want at the end rental → rental+2 but it might be +1.

Unrepeatable Read

```
x = Sel. rentals ---
if (x<5){
  upd Cust
  set rentals+=1
  ----
}
```

in here, it might read 4
However, another txn might set it to 5
here
Then, in this statement, we will have rentals = 6 instead of 5.
This read should be 4 again, but it is updated.

Inconsistent Read

```
Upd Pro.
Set Quantity+=5
where pro=gizmo.


upd Pro
set Quantity-=5
where product=gadget
```

select sum(quantity)
from Product.

Here, we will read 5 more.

Dirty Read

```
Client1  100$ acc1→acc2

X= acc1.balance
Acc2.balance +=100

if (X>100) acc1.balance -=100

else        //Rollback
     acc2balance = -100
```

```
Client2 100$ acc2→acc3

Y=acc2 balance
acc3.balance +=100

if (y>100) acc2.balance -=100

else  //Rollback
  acc3.balance -=100
```

Dirty read ≅ Inconsistent read

Crash

```
.upd  acc
set   bal -=100
  none see
CRASH
upd   acc
set   bal+=100
  none fixed
```

ACID

Atomicity → all or none

Consistency → our responsability
- → writing correct txn
- → integrity rules

Isolation → concurrency

Durability → will survive after failure (write to disk)
- → by dbms

Recap:

Dirty Reads — write-read conflict

T1	T2
W(A)	
	R(A)
Abort	

Inconsistent Read

T1	T2
A=20, B=20	
W(A)	
	R(A)
	R(B) → it will not read B0
W(B)	

write-read conflict

Unrepeatable Read — read-write conflict

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

Lost Update — write-write conflict

T1	T2
R(A)	
	R(A)
A+5	
	A*1.3
W(A)	→ lost
	W(A)

Isolation

Serial

T1 | T2

A+100
B+100

A*2
B*2

Serializable

A schedule is serializable, if it is equivalent to a serial schedule

T1 | T2

R(A,t)
t+=100
W(A,t)

R(A,S)
S=*2
W(A,S)

R(B,t)
t+100
W(B,t)

R(B,S)
S=*2
W(B,S)

This is serializable schedule
not serial schedule

Non Serializable

T1 | T2

R(A)
A+100
W(A)

R(A)
**2
W(A)
R(B)
B*2
W(B)

R(B)
B+100
W(B)

A=100, B=10

A=120

A=220

B=20

B=120

They both should be 220.

edge case

T1	T2	
R(A)		A=10 B=10
A+100		
W(A)		
	R(B)	
	B+200	
	W(B)	
	R(A)	
	A+200	
	W(A)	
R(B)		
B+100		
W(B)		A=310 B=310

it is serializable schedule
- T1 then T2 is equivalent to this schedule.
- However, we dont expect scheduler to find this.

Conflict Serializability

Conflicts

$r_i(X), w_j(Y)$ → Two actions by same txn.

$w_i(X), w_j(X)$ → Two writes on same element by diff txn

$w_i(X), r_j(X)$
$r_i(X), w_j(X)$ → read/write on same element by diff txn

Defn: A schedule is conflict serializable if it can be transformed into a serial schedule by a series of swappings of adjacent non conflicting actions

ex:
$r_1(A)\, w_1(A)\, r_2(A)\, w_2(A), r_1(B)\, w_1(B)\, r_2(B)\, w_2(B)$

$r_2(A)\, r_1(B), w_2(A)$

$r_1(B)\, r_2(A), w_2(A)\, w_1(B)$

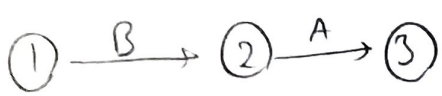
⇓

$r_1(A)\, w_1(A)\, r_1(B)\, w_1(B), r_2(A)\, w_2(A)\, r_2(B)\, w_2(B)$

Use precedence graph

ex.

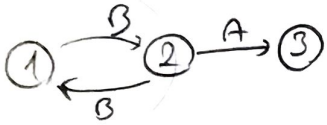
$r_2(A)\ r_1(B)\ w_2(A)\ r_3(A)\ w_1(B)\ w_3(A)\ w_2(B)$



no cycle, conflict serializable

ex:

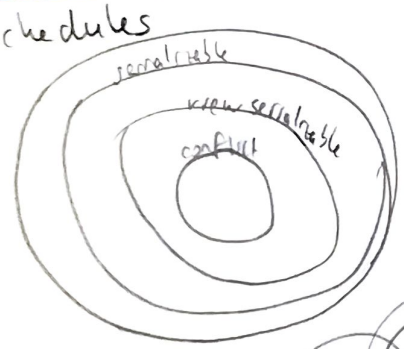
$r_2(A)\ r_1(B)\ w_2(A)\ r_2(B)\ r_3(A)\ w_1(B)\ w_3(A)\ w_2(B)$



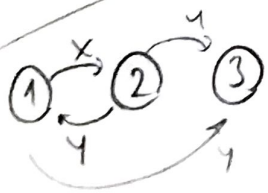
cycle, not conflict serializable

View Equivalence

schedules



$w_1(X)\ w_2(X)\ w_2(Y)\ w_1(Y)\ w_3(Y)$



cycle, not conflict serializable

This write will be lost, because T_1 and T_3 will overwrite.

equivalent

$w_1(X)\ w_1(Y)\ w_2(X)\ w_2(Y)\ w_3(Y)$

not conflict equivalent

ex:

T1	T2	T3
W1(X)		
	W2(X)	
	W2(Y)	
	C	
W1(Y)		
C		
		W3(Y)
		C

$\Rightarrow$

T1	T2	T3
W1(X)		
W1(Y)		
C		
	W2(X)	
	W2(Y)	
	C	
		W3(Y)
		C

lost

Serializable, but not conflict serializable.

Two schedule is view equivalent:

S: reads A initially

S': reads A initially → initial reads

$w_2(A)$ $r_1(A)$ = $w_2(A)$ $r_1(A)$ → T_1 reads T_2's write

$\bar{w}_1(\bar{A})$ $\bar{w}_1(A)$ → final writes

ex:

$r_2(B)\ w_2(A)\ r_1(A)\ r_3(A)\ w_1(B)\ w_2(B)\ w_3(B)$

T_2 reads B first

T_1 reads A first

A

T_2 writes

T_1 reads

T_3 reads

B

T_2 reads

T_1 writes

T_2 writes

T_3 writes

A'ya en son T_2 yaziyor.

$T_2 \xrightarrow{A} T_3$, $T_2 \xrightarrow{A} T_1$

T_1 - T_3 sirasi nasil olmali?

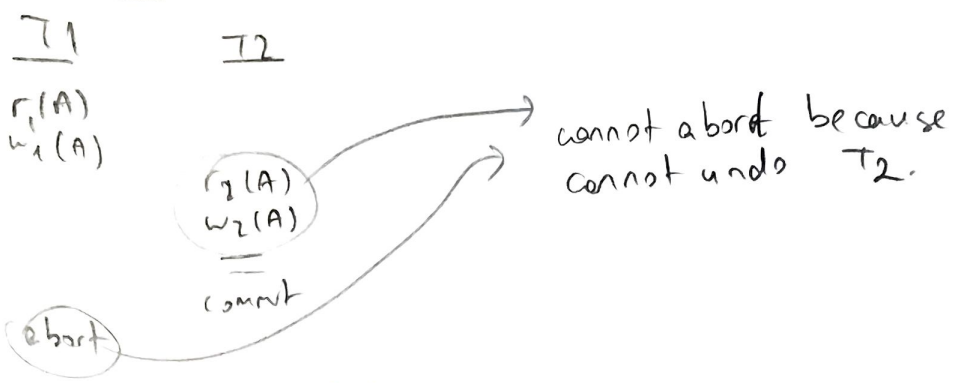
en son T_3 yazdigi icin T_3 olmali

$T_2 - T_1 - T_3$

$r_2(B)\ w_2(A)\ w_2(B)\ r_1(A)\ w_1(B)\ r_3(A)\ w_3(B)$

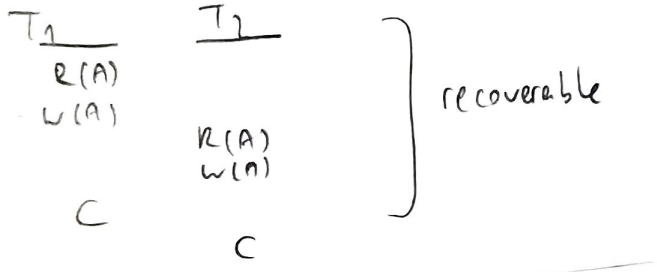
view equivalent

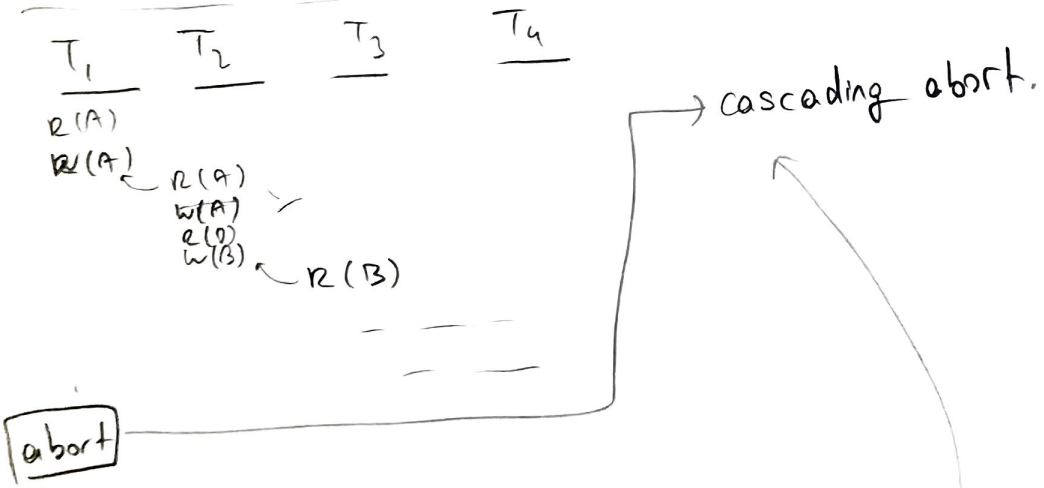
Question

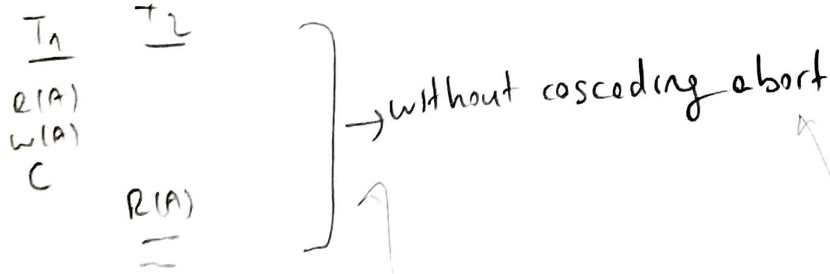


Recoverable Schedules

- We have to read transaction that have committed.
- Conflict serializable







review

Serializability

- Serial
- Serializable
- Conflict serializable
- View serializable

Recoverability

- Recoverable
- Avoid cascading aborts

Locks

- Pessimistic scheduler

$L_i(A) \rightarrow$ Tx; acquires lock for A

$U_i(A) \rightarrow$ Tx; releases lock for A

Non serializable schedule

T1	T2
R(A)	
A+100	
W(A)	
	R(A)
	A*2
	W(A)
	R(B)
	B*2
	W(B)
R(B)	
B+100	
W(B)	

⇒

T_1	T_2
$L_1(A)$	
R(A)	
A+100	
$W(A), U_1(A) L_1(B)$	
	$L_2(A) R_2(A)$
	A*2
	$W_2(A) U_2(A)$
	$L_2(B)$ //DENIED, wait
$R_1(B)$	
B+100	
$W(B) U_1(B)$	
	// Granted
	R(B)
	B*2
	$W(B) U_2(B)$

worst case

conflict serializable ✓

T_1	T_2
$L_1(A)$	
R(A)	
A+100	
W(A)	
$U_1(A)$	
	$L_2(A) R_2(A)$
	A*2
	$W(A) U_2(A)$
	$L_2(B) R_2(B)$
	B*2
	$W(B) U_2(B)$
$L_1(B) R(B)$	
B+100	
$W(B) U_1(B)$	

not conflict serializable!

Locks did not enforce that!

Two Phase Locking (2PL)

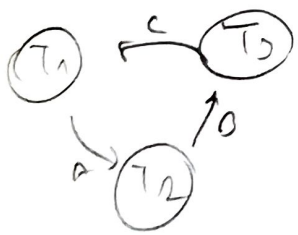
- In every txn, all lock request must precede all unlock

ex:

T_1	T_2
$L_1(A)\ L_1(B)$	
$R(A)$	
$A+100$	
$W(A)\ U_1(A)$	
	$L_2(A)\ R(A)$
	$A*2$
	$W(A)$
	$L_2(B)$ //denied, it waits until the lock is released
$U_2(B)$	
	// granted

Theorem 2PL ensures conflict serializability

Proof: Suppose it does not ensure, then, there is cycle in the graph.



Then, there is temporal cycle in the schedule

$U_1(A) \rightarrow L_2(A)$ // 1 txn, A'yı bıraktınca 2 alacak
$L_2(A) \rightarrow U_2(B)$
$U_2(B) \rightarrow L_3(B)$
$L_3(B) \rightarrow U_3(C)$
$U_3(C) \rightarrow L_1(A)$

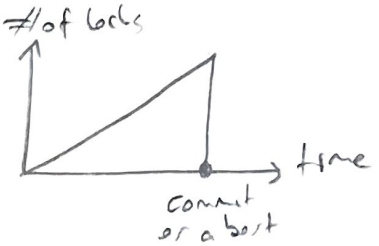
2PL and nonrecoverable Problem

T_1	T_2
$L_1(A)\ L_1(B)$	
	$L_2(A)$
	$L_2(B)$ //denied
abort	commit

Problem!

Strict 2PL

- Locks are released at COMMIT or ROLLBACK
- ensures recoverable
- avoids cascading aborts



No shrinking

T_1	T_2
$L_1(A)\ R(A)$	
$W(A)$	$L_2(A)$ // denied
$L_1(B)\ R(B)$	
$W(B)$	
$U_1(A)\ U_1(B)$	
Rollback ✓	// Granted ✓

- Strict schedule does not allow dirty reads and dirty writes.
- It can still be deadlock!

T_1	T_2
$L_1(A)\ R_1(A)$	
$W(A)$	$L_2(B)\ R(B)$
	$L_2(A)$ // denied
$L_1(B)$ // denied	

- Issues
  - Implementation
  - Lock modes
  - Granularity
  - Deadlocks
  - Performance (looks like serial schedules)

Locking Scheduler

Task 1 → behalf of txn
- → read/write, lock request, strict 2PL

Task 2 → behalf of system
- → Lock table, when a lock is requested check the lock table.
- → when txn abort, release all locks
- → check for deadlocks occasionally.

Fine granularity → rows, tuples
- → high concurrency
- → # locks ↑, high overhead

Coarse grain → tables, predicate locks → sel from R1 where R1.0 > 100
- → less concurrency
- → less overhead
- → many false conflict (we think that it is locked, it is being used but it is not)

Lock Modes

S → shared → read → more than 1 txn can read

X → exclusive → write → only 1 txn can write, none can read/write

Hierarchical Locking

- To place a lock on element, start at the top.
- If element to lock, get S or X.
- If element is in deeper level, leave intentional lock, IS, IX

To get	Must have all ancestors
IS or S	IS or IX
IX, SIX, X	IX or SIX

SIX → I will read all table, update some of them in some condition.

page	Lock	current	wait
P1	S	$\{T_1\}$	$\{T_2\}$

T_1 was reading P1, T_1 changed its mind and want to write on P1.

lock upgrade

P1	X	$\{T_1\}$	$\{T_2\}$

Problem

	$S_1(A)$	$S_2(A)$	$X_2(A)$	$X_1(A)$
P1	S	$\{T_1, T_2\}$	$\{\}$	$S_1(A), S_2(A)$
P1	S	$\{T_1, T_2\}$	$\{T_2\}$	$X_2(A)$
P1	S	$\{T_1, T_2\}$	$\{T_1, T_2\}$	$X_1(A)$ deadlock !