ECS 165: Database Systems

Project's Third Milestone

Winter 2025

UCDAVIS
UNIVERSITY OF CALIFORNIA

Two-Phase Locking (2PL)

Transaction 1: transfer $100 from A to B

```
x1 = read (A)
write (A, x1-100)
y1 = read (B)
write (B, y1+100)
```

Transaction 2: transfer $50 from B to A

```
x2 = read (B)
write (B, x2-50)
y2 = read (A)
write (A, y2+50)
```

Two-Phase Locking (2PL)

x1 = read (A)	
write (A, x1-100)	
y1 = read (B)	
write (B, y1+100)	
	x2 = read (B)
	write (B, x2-50)
	y2 = read (A)
	write (A, y2+50)

If Transaction 1 and Transaction 2 are executed serially:

A_new = A_old - 50
B_new = B_old + 50

UCDAVIS

Two-Phase Locking (2PL)

x1 = read (A)	
write (A, x1-100)	
y1 = read (B)	
	x2 = read (B)
	write (B, x2-50)
	y2 = read (A)
write (B, y1+100)	
	write (A, y2+50)

If Transaction 1 and Transaction 2 are not executed serially:
A_new = A_old - 50
B_new = B_old + 100

We need to prevent concurrent transactions from accessing the same data simultaneously, which can lead to inconsistency.

Two Phase Locking (2PL)

- T1 reads a data item that T2 reads? **Allowed**
- T1 writes a data item that T2 reads? **Not Allowed**
- T1 reads a data item that T2 writes? **Not Allowed**
- T1 writes a data item that T2 writes? **Not Allowed**

Two Phase Locking (2PL)

- T1 reads a data item that T2 reads? **Allowed**
- T1 writes a data item that T2 reads? **Not Allowed**
- T1 reads a data item that T2 writes? **Not Allowed**
- T1 writes a data item that T2 writes? **Not Allowed**

    - Shared-Lock/Read-Lock: allow multiple transactions to read the same data simultaneously, but prevent other transactions from writing(updating) the same data (acquiring the exclusive-lock)
    - Exclusive-Lock/Write-Lock: allow one transactions to write(update) the same data, but prevent other transactions from reading or writing(updating) the same data (acquiring the shared-lock or exclusive-lock)

Two Phase Locking (2PL)

- T1 reads a data item that T2 reads? **Allowed**
- T1 writes a data item that T2 reads? **Not Allowed**
- T1 reads a data item that T2 writes? **Not Allowed**
- T1 writes a data item that T2 writes? **Not Allowed**

Lock compatibility table

Lock type	read-lock	write-lock
read-lock	✔	X
write-lock	X	X

- Shared-Lock/Read-Lock: allow multiple transactions to read the same data simultaneously, but prevent other transactions from writing(updating) the same data (acquiring the exclusive-lock)
- Exclusive-Lock/Write-Lock: allow one transactions to write(update) the same data, but prevent other transactions from reading or writing(updating) the same data (acquiring the shared-lock or exclusive-lock)

Two Phase Locking (2PL)

1. **Acquiring**: acquire all locks needed by a transaction
2. Execute the transaction
3. **Releasing**: release all locks acquired

In **Acquiring**, a transaction should release all acquired locks once it cannot acquire some lock. We call this **aborting** a transaction.

You should keep attempting to commit an aborted transaction until it is committed.

Two Phase Locking (2PL) Implementation

- How to implement shared lock?
- How to implement exclusive lock?
- How to prevent two threads acquiring the same exclusive lock?
- How to prevent one thread acquiring the shared lock and another thread acquiring the exclusive lock?
- How to abort a transaction immediately if it cannot acquire some lock?

Two Phase Locking (2PL) Implementation

- How to implement shared lock? **threading.RLock()**
- How to implement exclusive lock? **threading.Lock()**
- How to prevent two threads acquiring the same exclusive lock?
- How to prevent one thread acquiring the shared lock and another thread acquiring the exclusive lock?
- How to abort a transaction immediately if it cannot acquire some lock?

Two Phase Locking (2PL) Implementation

- How to implement shared lock? **threading.RLock()**
- How to implement exclusive lock? **threading.Lock()**
- How to prevent two threads acquiring the same exclusive lock?
- How to prevent one thread acquiring the shared lock and another thread acquiring the exclusive lock?
- How to abort a transaction immediately if it cannot acquire some lock?

threading.RLock() and threading.Lock() do not solve the last 2 problems.

Design and implement your own Lock class.

Two Phase Locking (2PL) Implementation

```
class MyLock{
  uint reading_count;   // The number of transactions that have acquired the shared lock
  uint writing_count;   // The number of transactions that have acquired the exclusive lock
  mutex mut;            // Prevent multiple threads from accessing a MyLock object simulaneously
                        // Search "mutex in C++" and "thrading.Lock() in Python" for more information
  bool GetSharedLock();
  bool GetExclusiveLock();
  // ...
};
```

Two Phase Locking (2PL) Implementation

```
bool MyLock::GetSharedLock(){
  mut.lock();
  // Ensure no other transaction has acquired the exclusive lock
  if(writing_count == 0){
    reading_count++;
    mut.unlock();
    return true;
  }else{
    mut.unlock();
    return false;
  }
}
```

Two Phase Locking (2PL) Implementation

```
bool MyLock::GetExclusiveLock(){
  mut.lock();
  // Ensure no other transaction has acquired the exclusive lock or shared lock
  if(writing_count == 0 && reading_count==0){
    writing_count++;
    mut.unlock();
    return true;
  }else{
    mut.unlock();
    return false;
  }
}
```

Record-Level Lock and **Page Latch**

- Record-Level Lock: For the same record (a base record and its tail records), only simultaneous reads are allowed.
  - Ex1. T1 reads R1, T2 writes R1
  - Ex2. T1 reads R1, T2 writes R2, both R1 and R2 are in the same page
- Page Latch: For the same page, no two transactions can load it from/write it to disk simultaneously.
  - Ex3. T1, T2 both reads records in the same page that is not in the memory.

Record-Level Lock and **Page Latch**

- Record-Level Lock: For the same record, only simultaneous reads are allowed.
  - Ex1. T1 reads record R1, T2 writes record R1
  - Ex2. T1 reads record R1, T2 writes record R2, both R1 and R2 are in the same page
  - Ex1 is not allowed;
- Page Latch: For the same page, no two transactions can load it from/write it to disk simultaneously.
  - Ex3. T1, T2 both reads records in the same page that is not in the memory.

Record-Level Lock and **Page Latch**

- Record-Level Lock: For the same record, only simultaneous reads are allowed.
  - Ex1. T1 reads record R1, T2 writes record R1
  - Ex2. T1 reads record R1, T2 writes record R2, both R1 and R2 are in the same page
  - Ex1 is not allowed; In Ex1, T2 should be aborted if T1 acquires the lock first.
  - Ex2 is allowed
- Page Latch: For the same page, no two transactions can load it from/write it to disk simultaneously.
  - Ex3. T1, T2 both reads records in the same page that is not in the memory.

Record-Level Lock and **Page Latch**

- Record-Level Lock: For the same record, only simultaneous reads are allowed.
  - Ex1. T1 reads record R1, T2 writes record R1
  - Ex2. T1 reads record R1, T2 writes record R2, both R1 and R2 are in the same page
  - Ex1 is not allowed; In Ex1, T2 should be aborted if T1 acquires the lock first.
  - Ex2 is allowed
- Page Latch: For the same page, no two transactions can load it from/write it to disk simultaneously.
  - Ex3. T1, T2 both reads records in the same page that is not in the memory.
  - Ex3 is allowed. Only one transaction is allowed to load the page into the memory, which can implemented via the Python built-in exclusive lock **threading.Lock()**.

Record-Level Lock and **Page Latch**

- Record-Level Lock: For the same record, only simultaneous reads are allowed.
  - Ex1. T1 reads record R1, T2 writes record R1
  - Ex2. T1 reads record R1, T2 writes record R2, both R1 and R2 are in the same page
  - Ex1 is not allowed; In Ex1, T2 should be aborted if T1 acquires the lock first.
  - Ex2 is allowed.
- Page Latch: For the same page, no two transactions can load it from/write it to disk simultaneously.
  - Ex3. T1, T2 both reads records in the same page that is not in the memory.
  - Ex3 is allowed. Only one transaction is allowed to load the page into the memory, which can implemented via the Python built-in exclusive lock **threading.Lock()**.
  - **If T1 acquires the lock first, T2 waits for T1 to release the lock rather than gets aborted immediately.**

Aborting Transaction vs. Aborting Query

- **Aborting a Transaction**
  - Undo all changes the transaction has made and release all locks the transaction has acquired. Keep trying to execute an aborted transaction until it is committed if it is aborted for failing to acquire some lock.
  - Example: A transaction that tries to acquire a lock that is acquired and not released by another transaction should be aborted.
- **Aborting a Query**
  - Undo all changes the query has made and delete the query.
  - Example: A query that tries to insert a base record whose primary key already exists should be aborted.
  - You should abort the transaction of this aborted query and delete the transaction (no need to try to execute it again).

More Hints...

- Think about the concurrency issue when inserting two records in the same page range
- Think about the concurrency issue when inserting two records with the same primary key
- Don’t Write them to the same offset!