01

ECS 165A Milestone 3

Group 4: Zhengyu Wu, Leran Chen, Tedder Lao, Xingyu Pan, Jiaming Mai

What was accomplished and how?

Goal 1: Transaction Semantics:
To create the concept of the multi-statement transaction

Goal 2: Concurrency Control:
To create a multi-threaded environment
Running multiple transactions, 2PL

Presentation Highlights

Our Design & Solution

Transactions

Transactions: consisting of a set of read and write operations

- Abort

When to abort？
Acquire an unavailable lock
E.g.

1. More than one transaction try to access the same mutex lock.
2. A transaction try to set mutex lock while there is already a shared lock set by another transaction

What to abort?

1. Aborting all changes made in the transaction and roll back to the previous commit.
2. No retry on the same transaction.

Our Design & Solution

Transactions

Transactions: consisting of a set of read and write operations

- Commit

When to commit？
Write to database successful with no failure in acquiring locks

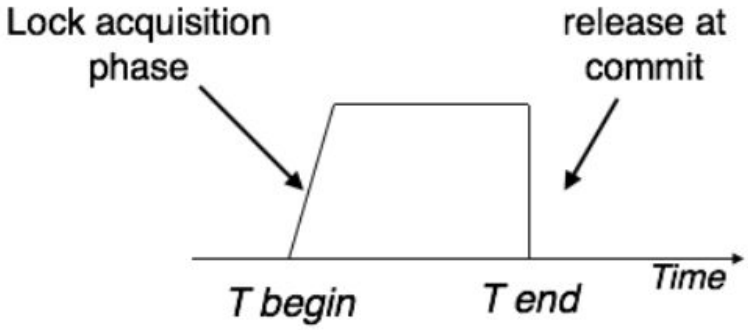
What to commit?
- Change state from "Start" to "Commit"
- Update become visible to other transactions
- All locks are released

Our Design & Solution

Concurrency Control - 2PL protocol

Strict Two-Phase Locking:

- Shared locks for reads
- Mutually exclusive locks for writes



Expanding Phase:

- Seeks permission for the locks it requires.
- Add Locks

Shrinking Phase:

- Holds all the locks until the commit point and releases all the locks at a time.

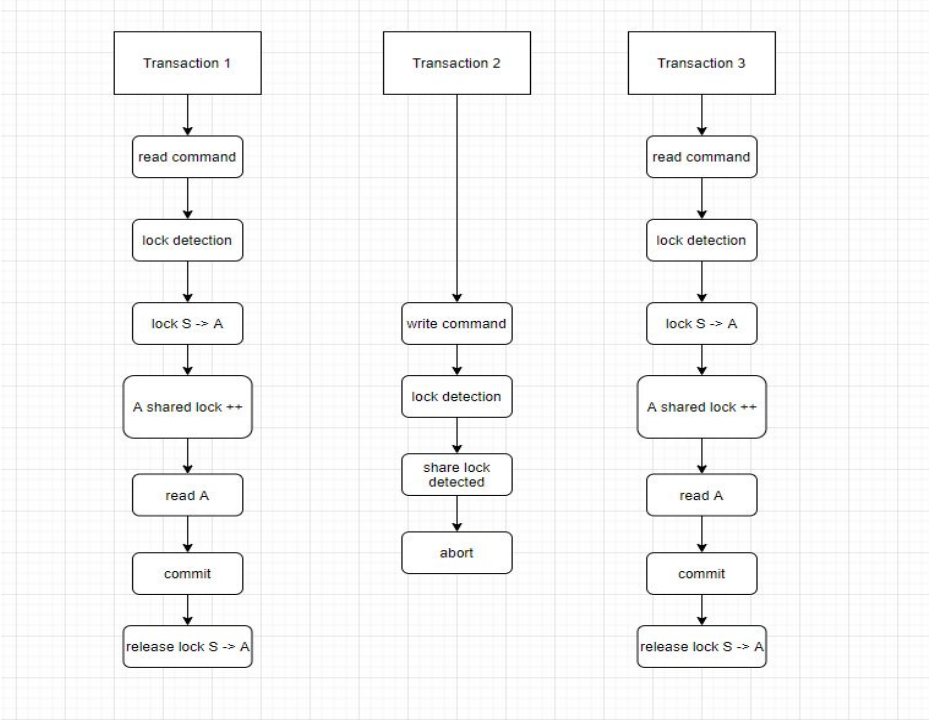
Our Design & Solution

Concurrency Control - 2PL protocol

Lock Manager:

1. Add mutex/shared lock(lock each record by transaction)
2. Release mutex/shared lock
3. Detect if any lock exists

Lock: Record Level

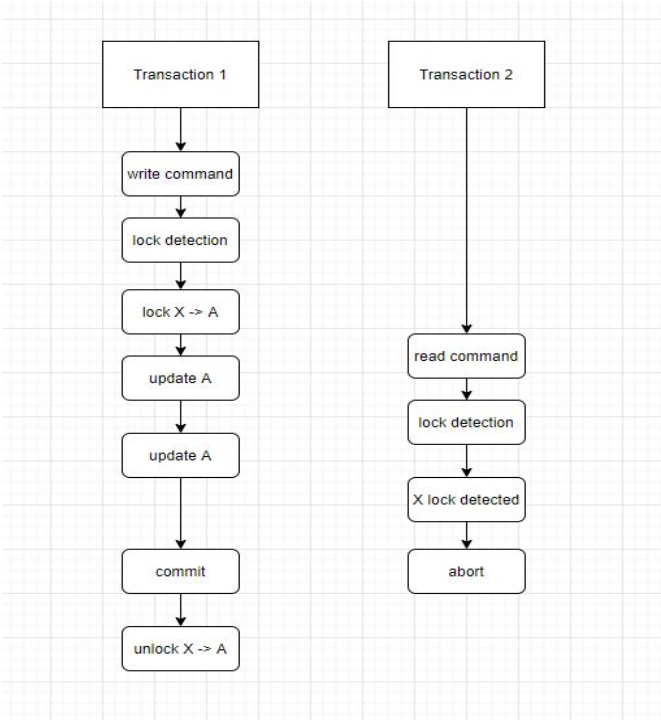


Our Design & Solution

Concurrency Control - 2PL protocol

Lock Manager:

1. Add mutex/shared lock(lock each record by transaction)
2. Release mutex/shared lock
3. Detect if any lock exists



Why use locks?

Concurrency issues:

- Modification Lost

  An update be replaced by another update in another thread

- Read dirty data

  Current transaction can read data that another transaction has not yet committed

- Non-repeatable read

  Multiple read a data in a transaction, but this data has been modified in another transaction

- Phantom read (future work)

  One transaction reading a range of data, but another transaction insert a new data into this range.

Data Structure Protection
Internal Lock: Latch

Page Range/Page/RID

- Using threading.RLock class
- Prevents same Rids being used in more than one page
- Latch will not block the transactions in different page ranges

Bufferpool eviction

- Using threading.RLock class
- Prevents same page range being evicted by several threads at the same time
- Eviction will be banned if current page range is currently accessed by any transaction

Thank you