CSC 350

CONCURRENCY

SCENARIO

1. Sue goes to Best Buy to purchase a VR headset.
2. She swipes her debit card.
    A. The banking db checks her balance.
    B. The banking db deducts the amount from her account.
    C. The banking db credits the money to Best Buy's account.
    D. The banking db sends the OK back to Best Buy.
3. Best Buy says "Card Approved".
4. Sue goes home happy.

SCENARIO

A. The banking db checks Sue's balance.

```
sBalance = read(SueAccount)
```

B. The banking db deducts the amount from Sue's account.

```
sBalance -= amount
write(sueAccount, sBalance)
```

C. The banking db credits the money to Best Buy's account.

```
bBalance = read(BBAccount)
bBalance += amount
write(BBAccount, bBalance)
```

SCENARIO

T1 - Sue Purchase

sBalance = read(SueAccount)

sBalance -= amount

write(sueAccount, sBalance)

bBalance = read(BBAccount)

bBalance += amount

write(BBAccount, bBalance)

T2 - Best Buy Replenish

bBalance = read(BBAccount)

bBalance -= orderAmount

write(BBAccount, bBalance)

LOCK TYPES

- Shared Locks - Compatible with other shared locks.
- Exclusive Locks - Incompatible with any other lock type.

If a transaction has a shared lock on resource A, other transactions can acquire shared locks on that resource, but not exclusive locks.

If a transaction has an exclusive lock on resource A, no other transaction may acquire any type of lock on resource A.

SCENARIO

T1 - Sue Purchase

```
lock(SueAccount)
    sBalance = read(SueAccount)
    sBalance -= amount
x-lock(SueAccount)
    write(sueAccount, sBalance)
unlock(SueAccount)
lock(BBAccount)
    bBalance = read(BBAccount)
    bBalance += amount
x-lock(BBAccount)
    write(BBAccount, bBalance)
unlock(BBAccount)
```

T2 - Best Buy Replenish

```
lock(BBAccount)
    bBalance = read(BBAccount)
    bBalance -= orderAmount
x-lock(BBAccount)
    write(BBAccount, bBalance)
unlock(BBAccount)
```

SCENARIO

T1	T2
x-lock(A)	x-lock(B)
x-lock(B)	x-lock(A)
unlock(B)	unlock(A)
unlock(A)	unlock(B)

DEADLOCKS

- Deadlocks occur when transaction A is holding onto a lock needed by transaction B, which is holding a lock needed by transaction A.
- Deadlocks are a “necessary evil” of concurrency, that can’t be completely removed, but we can employ strategies to minimize their effects.

TWO-PHASE LOCKING

- Growing phase - A transaction may obtain locks, but may not release any lock.
- Shrinking phase - A transaction may release locks, but may not obtain any new locks.
- Strict two-phase locking - All exclusive-mode locks taken by a transaction are held until that transaction commits.
- Rigorous two-phase locking - All locks are held until the transaction commits.

LOCK CONVERSIONS

- Upgrade - Convert a shared lock to an exclusive lock.
- Downgrade - Convert an exclusive lock to a shared lock.

COMMON LOCKING PROTOCOL

When a transaction issues a read operation, the system issues a lock instruction followed by the read instruction.

When a transaction issues a write operation, the system checks to see whether the transaction already holds a shared lock on the resource. If it does, then the system issues an upgrade instruction, followed by the write instruction. Otherwise, the system issues an x-lock instruction, followed by the write instruction.

All locks obtained by a transaction are unlocked after that transaction commits or aborts.

WHAT EXACTLY IS BEING LOCKED?

Each database engine has different granularities of locks:

Postgres: https://www.postgresql.org/docs/current/static/explicit-locking.html

https://nordeus.com/blog/engineering/postgresql-locking-revealed/

MySQL: https://dev.mysql.com/doc/refman/5.7/en/innodb-locking.html

MongoDB: https://docs.mongodb.com/manual/faq/concurrency/