WEEK 10

TRANSACTION – LOCKING

STUDENT OBJECTIVES

- Upon completion of this video, you should be able to:
  - List the 5 levels of lock granularity
  - Differentiate between Binary Locks, 2 Phase Locking and Shared Locks
  - Give an example of a situation that would cause Deadlock
  - List 3 methods for controlling Deadlock
  - Give an example of a situation that would cause Livelock

LOCKING

- guarantees the current transaction **EXCLUSIVE** use of the data. If T2 is using a piece of data (such as a balance), then T1 cannot use it until T2 has committed (the lock is released)
- **Lock Granularity:**
  - Database Level: SLOW (okay for batch not for real time), locks T2 out even if it wants to use a different table than T1!
  - Table Level: Also slow, not used much
  - Page Level: The DBMS locks a disk page (= 1 disk block could be 4 K, 8K, 16K) **QUESTION: How many tuples or tables would be locked when a page lock is on?** It depend on the size of the table
  - Row Level:
  - Field Level: **QUESTION: Very flexible but not good, why?**

have a bunch of lock, it is hard to manage.

LOCK TYPES

- **Binary Locks:** an object is either Locked (1) or Unlocked(0): Ok but restrictive because what if 2 transactions just want to read from the object but not update it?
- **Shared Locks** (see below)
- **2 Phase Locking** (see below)

problem is that the efficiency is low.

EXAMPLE OF BINARY LOCK:

Time	Transaction	Step	Stored Value
1	T1	Lock Item (What is an item? Consider Lock Granularity)	
2	T1	Read Bal	35
3	T1	Bal = 35+100	
4	T1	Write Bal	135
5	T1	Unlock Item ← now other thread could get access.	
6	T2	Lock Item	
7	T2	Read Bal	135
8	T2	Bal = 135-30	
9	T2	Write Bal	105
10	T2	Unlock Item	

- **Shared/Exclusive Locks:** 3 Types of locks:
  - ***no lock*** (unlocked)
  - ***shared lock*** (the transaction only wants to read the data)
  - ***exclusive lock*** (the transaction needs to update the data)
- 2 or more Read transactions can be safely executed so *shared* locks allow several transactions to act at the same time, however an *exclusive* lock is granted only if the object has no other locks on it and until the exclusive lock is released no other locks are granted on that object

PROBLEMS:

- Transaction schedule may not be **serializable**
- the schedule may produce **deadlocks**

Time	Transaction	Step	Stored Value
1	T1	Share lock y	30
2	T1	Read y	
3	T1	Unlock y	
4	T2	Share lock x	20
5	T2	Read x	
6	T2	Unlock x	
7	T2	Exclusive lock y	30
8	T2	Read y	
9	T2	y= x + y	50
10	T2	Write y	50 y
11	T2	Unlock y	
12	T1	Exclusive lock x	20
13	T1	Read x	
14	T1	x = x + y	50
15	T1	Write x	50 x
16	T1	Unlock x	

As a result of X has 50 and Y has 50 BUT if it was T1, then T2, X would have 50, Y would have 80 or T2, then T1, X would have 70, Y would have 50, thus this was not a serializable schedule!

2-PHASE LOCKING

- ensures ***serializability*** but does not prevent ***Deadlock***
- **Phase 1:** The transaction acquires all required locks without unlocking any data (growing phase), once all locks have been established the transaction is at the locked point
- **Phase 2:** Transaction can release locks but not get new locks (shrinking phase) have to wait
- **Rules:**
  - 1. No 2 transactions can have conflicting locks.
  - 2. No unlock can precede a lock in the same transaction,

THIS SIMPLE PROTOCOL GUARANTEES THAT IF A SET OF TRANSACTIONS RUNS WITH 2-PHASE LOCKING, THEY WILL ALL BE GUARANTEED TO BE SERIALIZABLE.

DEADLOCK

- **DEADLOCK:** 2 or more transactions are waiting eternally for each other to release a lock (called **deadly embrace**).

EXAMPLE OF DEADLOCK:

Situation: T1 needs to lock x and then lock y, while at the same time T2 need to lock y and then x

they're unable to unlock.

Time	Transaction	Reply	Lock Status	
			Data x	Data y
0			Unlocked	Unlocked
1	T1: Lock(x)	OK	Locked	Unlocked
2	T2: Lock(y)	OK	Locked	Locked
3	T1: Lock(y)	Wait	Locked	Locked
4	T2: Lock(x)	Wait	Locked	Locked
5	T1: Lock(y)	Wait	Locked	Locked
6	T2: Lock(x)	Wait	Locked	Locked
7	T1: Lock(y)	Wait	Locked	Locked
8	T2: Lock(x)	Wait	Locked	Locked
…	…	…	…	…

CONTROLLING DEADLOCK

- **Deadlock prevention:** Transaction aborts if there is a possibility of deadlock occurring. If the transaction aborts, it must be rollback and all locks it has are released
- **Deadlock detection:** DBMS occasionally checks for deadlock, if there is deadlock, it randomly picks one of the transactions to kill (I.e. rollback) and the other continues
- **Deadlock avoidance:** a transaction must obtain all it's locks before it can begin so deadlock will never occur.

- Choice of which method to use depends on your situation:
  - If deadlock isn't likely use deadlock detection
  - If deadlock is likely to happen use deadlock prevention
  - If system response time is not a high priority use deadlock avoidance

LIVELOCK

- A transaction never gets its lock
- Consider a disk server that chooses transactions to grant a lock to on the basis of minimal disk distance:
- Example:
  - T1: Lock (A) Okay
  - T2: Lock (A) No (cylinder =200)
  - T3: Lock (A) No
  - T1: Release (A)
  - Grant lock to T3 ← T_3 is closer on disk.
  - T4: Lock (A) No
  - T3: Release (A)
  - T4: Lock (A) Okay
  - Processing continues but T2 may never be given the lock to A.

starvation.

LIVELOCK SOLUTIONS

- OPTION 1: Always grant to the T which has waited longest→FCFS (first come, first serve)
- OPTION 2: Give transactions priority and the longer a transaction has to wait, the higher the priority it gets.

MYSQL LOCKING

- Example:

LOCK TABLE items;

START TRANSACTION;

INSERT INTO items (name, label) VALUES ('B123', 'bike');

UNLOCK TABLES;