Concurrency Control (Spring Break Recap)

R&G Chapter 17

(slides adapted from content by J.Gehrke, J.Shanmugasundaram, and/or C.Koch)

Transactions

What does it mean for a database operation to be correct?

Transactions

What does it mean for a database operation to be correct?

How does a database interact with its users?

Transactions

- Users group sequences of interactions with the DBMS into a <u>Transaction</u>.
  - *Atomicity*: From the user's perspective, transactions execute <u>fully</u> and <u>on their own</u>.
  - *Correctness*: Transactions preserve data consistency (as per Integrity Constraints).
- The DBMS interleaves DB operations while respecting the above guarantees.

Example: Schedule

Time	T1	T2
↓	`A=A+100`	
		`A=1.06*A`
	`B=B-100`	
		`B=1.06*B`

Example: Schedule

Time	T1	T2
↓	`A=A+100`	
		`A=1.06*A`
	`B=B-100`	
		`B=1.06*B`

OK!

Example: Schedule

Time	T1	T2
	`A=A+100`	
		`A=1.06*A`
		`B=1.06*B`
	`B=B-100`	

Example: Schedule

Time	T1	T2
↓	`A=A+100`	
		`A=1.06*A`
		`B=1.06*B`
	`B=B-100`	

Not OK!

Schedule Equivalence

- Two schedules are [X] Equivalent if:
  - **Equivalence**: The "visible effects" of both schedules are the same.
    - "visible effects" is hard to define/test for.
  - **Conflict Equivalence**: "Conflicting operations" are performed in the same order.
  - **View Equivalence**: Like conflict equivalence, but ignoring the order of 'invisible' operations
    - i.e., For 3+ writes, only the last one matters.

Schedule Serializability

- A schedule is Serial if there is **no** interleaving between transactions.
- A schedule is Serializable if it is equivalent to **any** Serial schedule for the same set of transactions.
  - Resp. Conflict/View Serializable if it is Conflict/View Equivalent.

Dependency Graphs

- One node per transaction
- One edge (from T_i to T_k) if T_k reads/writes an object most recently written by T_i.
- **Claim**: Conflict-equivalent schedules have identical dependency graphs.
  - A schedule is conflict serializable if and only if its dependency graph is acyclic.

2-Phase Locking

- To Recap:
  - Obtain reader/writer locks on objects.
  - Once an xact releases a lock, it can no longer acquire any new locks.
- **Claim**: 2-phase locking allows only conflict-serializable schedules.
  - If xact A modifies a value, no other xact can read/modify that value until A 'completes'.

Deadlocks

- Deadlock: A cycle of transactions waiting on each other's locks
  - Problem in 2PL; xact can't release a lock until it completes
- How do we handle deadlocks?
  - **Anticipate**: Prevent deadlocks before they happen.
  - **Detect**: Identify deadlock situations and abort one of the deadlocked xacts.

Deadlock Prevention

- 1) Track which xacts are waiting for which xacts (waits-for graph)
  - If a cycle exists, kill one of the xacts.
- 2) Prioritize transactions
  - a) Lower priority xacts die when they try to take a lock held by a higher priority xact.
  - b) Higher priority xacts kill any xact holding a lock that they need.

What do we lock?

- The entire database?
- A table in the database?
- Individual pages a table is stored on?
- Individual tuples in a table?

New Lock Modes

- 'Intent' to lock (a child)
  - Intent-to-Lock Shared (IS)
  - Intent-to-Lock Exclusive (IX)
- Actual lock (on the object)
  - Lock-Shared (S)
  - Lock-Exclusive (x)
- Lock Shared + Intent-to-Lock Exclusive (SIX)

New Lock Modes

Lock Mode(s) Currently Held By Other Xacts

Lock Mode Desired

	None	IS	IX	S	X
None	valid	valid	valid	valid	valid
IS	valid	valid	valid	valid	fail
IX	valid	valid	valid	fail	fail
S	valid	valid	fail	valid	fail
X	valid	fail	fail	fail	fail

Hierarchical Locks

- Lock Objects Top-Down
  - Before acquiring a lock on an object, an xact must have at least an intention lock on its parent!
- For example:
  - To acquire a S on an object, an xact must have an IS, IX on the object's parent (why not S, SIX, or X?)
  - To acquire an X (or SIX) on an object, an xact must have a SIX, or IX on the object's parent.

T1

T2

```
DELETE FROM Officers
WHERE rank = 1
  AND age =
   (SELECT MAX(age)
    FROM Officers WHERE rank=1)
LIMIT 1;
```

Time

T1 T2

```
DELETE FROM Officers
WHERE rank = 1
  AND age = (71)
    (SELECT MAX(age)
     FROM Officers WHERE rank=1)
LIMIT 1;
```

Time

T1

```
DELETE FROM Officers
WHERE rank = 1
  AND age = (71)
   (SELECT MAX(age)
    FROM Officers WHERE rank=1)
LIMIT 1;
```

T2

```
INSERT INTO Officers(rank,age)
               VALUES (1, 96);
```

Time

T1

```
DELETE FROM Officers
WHERE rank = 1
  AND age = (71)
   (SELECT MAX(age)
    FROM Officers WHERE rank=1)
LIMIT 1;
```

T2

```
INSERT INTO Officers(rank,age)
               VALUES (1, 96);
DELETE FROM Officers
WHERE rank = 2
  AND age = (80)
   (SELECT MAX(age)
    FROM Officers WHERE rank=2)
LIMIT 1;
```

Time

T1 | T2

Time

```
DELETE FROM Officers
WHERE rank = 1
   AND age = (71)
    (SELECT MAX(age)
     FROM Officers WHERE rank=1)
LIMIT 1;
```

```
INSERT INTO Officers(rank,age)
               VALUES (1, 96);
DELETE FROM Officers
WHERE rank = 2
   AND age = (80)
    (SELECT MAX(age)
     FROM Officers WHERE rank=2)
LIMIT 1;
```

```
SELECT MAX(age)
FROM Officers(rank,age)    (63)
WHERE rank = 2
```

T1 T2

Time

T1:

```
DELETE FROM Officers
WHERE rank = 1
   AND age = (71)
     (SELECT MAX(age)
      FROM Officers WHERE rank=1)
LIMIT 1;
```

T2:

```
INSERT INTO Officers(rank,age)
               VALUES (1, 96);
DELETE FROM Officers
WHERE rank = 2
   AND age = (80)
     (SELECT MAX(age)
      FROM Officers WHERE rank=2)
LIMIT 1;
```

T1:

```
SELECT MAX(age)
FROM Officers(rank,age)   (63)
WHERE rank = 2
```

[INCONSISTENT]

Dealing With Insertions

- Query semantics can be violated by an intermediate insertion if we only lock tuples.
  - Solution 1: Lock entire table (expensive!)
  - Solution 2: Lock a **predicate**.

    (e.g., `rank = 1`)
  - Solution 3: Lock **index page(s)**.

    (equivalent to a range/predicate lock)

Index Locking

- If there is an index on `rank`, T1 locks the index page(s) with `rank = 1`.
- If no such entries exist, lock the page where an entry would go.
- Index locking is a special case of Predicate Locking that is much more efficient to implement.

Simple Tree Locking Algorithm

- **Scan:** Start at the root and descend.
  - Repeatedly S lock node, then unlock parent.
- **Update**: Start at the root and descend
  - Repeatedly X lock node.
  - If all children of a node are locked and safe, release the parent lock
  - Safe node: A node that will not propagate changes.

Simple Tree Locking Algorithm

- **Scan:** Start at the root and descend.
    - Repeatedly S lock node, then unlock parent.
- **Update**: Start at the root and descend
    - Repeatedly X lock node.
    - If all children of a node are locked and safe, release the parent lock
    - Safe node: A node that will not propagate changes.

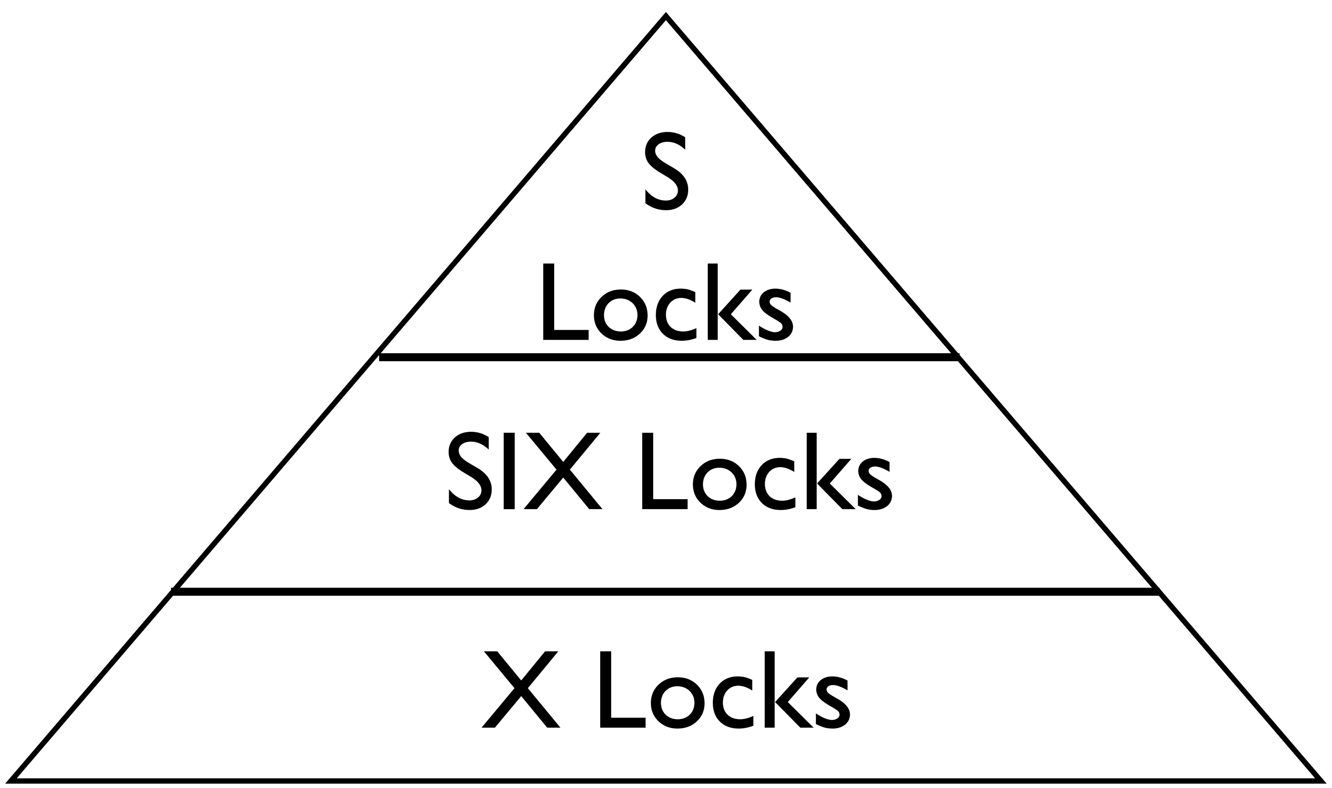
When is a node safe for single inserts? single deletes?

Better Tree Locking Algorithm (Bayer Schkolnick)

- **Scan:** As before
- **Update**: Set locks as if for search (using S)
  - Acquire X lock on the leaf.
  - If leaf is not <u>safe</u>, release <u>all</u> locks and restart using the simple algorithm.
- Gambles that only the leaf node will be modified.
  - S locks set on first pass are wasted otherwise.
  - In practice, better than the simple algorithm.

Hybrid Algorithm

- The likelihood that we need an X lock decreases as we move up the tree!



Aborting Transactions

- If a transaction T_i is aborted, all of its actions have to be undone!
  - What if T_j reads a value modified by T_i?
  - How can we prevent this from happening?
- The DBMS maintains a log of every write, which it uses to undo aborted xacts.
  - The log also assists in crash recovery!

The Log

- The log records:
  - T_i writes: Both old and new values of object.
    - Log record must make it to disk before the changed data page!
  - T_i commits/aborts: A record of the event.
- Log records are chained (stored as a linked list) for each xact, so it's easy to undo an xact.