CSCI 403 Database Management

Transaction Control
Concurrency Mechanisms

This Lecture
- Transaction Control
  - Concurrency problems
  - Transaction control
  - ACID guarantees
- Concurrency Mechanisms
  - 2-phase locking
  - Other mechanisms

Multiuser Database
- Problem: multiple users, all getting some time on the database
  - Users can contend for access to the same objects
  - Can lead to a variety of concurrency problems
- Model: Simple reads and writes of objects:
  - Blocks
  - Rows
  - Tables

Concurrency Problems
- Lost updates
- Dirty read
- Incorrect summary

Lost Updates
- First user reads some object A
- Second user reads same object (A)
- First user modifies A
- Second user modifies A based on their read of A
- First user’s modifications are lost

Example – airline reservation:
You book a seat, but later find out you “never” booked it.
### Dirty Read
- First user begins updating, updates object A
- Second user reads modified object A
- First user encounters an error, and undoes changes to A
- Second user is now working on incorrect version of A

Example: Flight is full (when it really isn’t).

### Incorrect Summary
- User 1 begins updating data
- User 2 runs a summary on same data – which will reflect partial changes

Example: a travel agent is changing a booking, which is implemented by adding passengers to one flight and then removing them from another. In between operations, the airline counts up total reservations, getting an incorrect total.

### Transaction Control
- Extend basic read/write model with transactions
- Operations in new model:
  - Begin transaction
  - End transaction
  - Commit
  - Rollback
  - Read
  - Write

### Atomic Transactions
- Goal of transaction control: atomicity (a transaction is treated as a single, indivisible operation)
  - After a successful commit, all changes (in transaction) must be permanent
  - After a rollback, all changes (in transaction) must be undone

### ACID
More generally, we are interested in enforcing these requirements:
- Atomicity (described above)
- Consistency – no inconsistent states before/after transaction (e.g., referential integrity is maintained)
- Isolation – the execution of a transaction should appear to be isolated from any other occurring on the database
- Durability – under no circumstance (e.g., server errors, loss of network connectivity) should committed changes be lost from the database

### Transactions in SQL
- No explicit begin/end transaction operations
  - First query starts a transaction
  - Transaction ends with commit or rollback (next query starts a new transaction)
- In psql, however, autocommit is on by default
  - Every query is immediately committed
  - To explicitly start a transaction, use keyword “BEGIN”

(Live demonstration)
Serializability

- Basic idea: schedule concurrent transactions such that the result is the same as if one followed the other serially.
- E.g.
  Not serial (lost update):
  T1: read(X)
  T2: read(X)
  T2: write(X)
  T1: write(X)

Not serial, but serializable:

T1: read(X)
T1: write(X)
T2: read(X)
T2: write(X)
T1: read(Y)
T1: write(Y)

Serializability

Managing Concurrency

Many different mechanisms:
- Locking (today’s focus)
- Timestamping
- Versioning/snapshots

Locks

Definition: a lock is a variable associated with an object describing the status of the object with respect to the operations that can be applied to the object.

Simple or binary locks:
- Two states: locked/unlocked
- A locked object is accessible only by the locker, until unlocked
- All others must wait
- This is generally considered too restrictive – e.g., what about multiple readers?

More Locks

Shared/Exclusive aka Read/Write locks:

Locks have three states:
- Read-locked (shared)
- Write-locked (exclusive)
- Unlocked
**Read Lock Rules**

When an object is read-locked:
- Locker may only perform reads.
- No other transaction can acquire a write-lock, must wait.
- Others may acquire read-locks and perform read-only operations.

**Write Lock Rules**

When an object is write-locked:
- Only locker may read OR write, all others must wait.

**Two-Phase Locking (2PL)**

- Require: all locking operations in a transaction occur before any unlocking operations (this defines 2PL)
- Guarantees serializability.
- Variations:
  - Conservative – all items locked before any reads or writes occur. Avoids deadlocks, but a bit restrictive.
  - Strict – write locks are released only after commit or rollback. Not deadlock-free.
  - Rigorous – like strict, but for all locks.

**Locking Disadvantages**

- Higher overhead
- Deadlocks
  - E.g.
    - T1: locks(X)
    - T1: requests lock(Y)
    - T2: locks(Y)
    - T2: requests lock(X)
- Various mechanisms for preventing or breaking (lots of tradeoffs)
- Starvation – “unfair” scheduling of locking

**Other Mechanisms**

- Timestamp ordering
- Multiversioning/snapshotting (Postgres uses a variant of this)
  - Each transaction sees a snapshot of data at time transaction begins
  - In case of conflict, transaction may have to be aborted/rolled back and restarted
  - Advantage over locking: reads don’t prevent writes, writes don’t prevent reads.

**Up Next**

- Programming against the database: Python, Java, and ORM