CHAPTER : Transactions

A transaction — a way for an application to group several reads and writes together into a logical unit

provides safety guarantees

ACID (Atomicity, Consistency, Isolation and Durability)

↑ the safety guarantees provided by transactions

- Atomicity
  — when a client make several writes, but a fault occurs in the midway (after some writes are done), the transaction is aborted and database undo any writes.

- Consistency
  — property of an application
  ↑
  invariants must always be true

* Isolation
  - concurrently executing transactions are isolated from each other
    [serializability]
      - each transaction can pretend that it is the only transaction running on the entire database

* Durability
  - promise that once a transaction has committed successfully, any data it has written will not be forgotten

Weak Isolation Levels

- protect against some concurrency issues

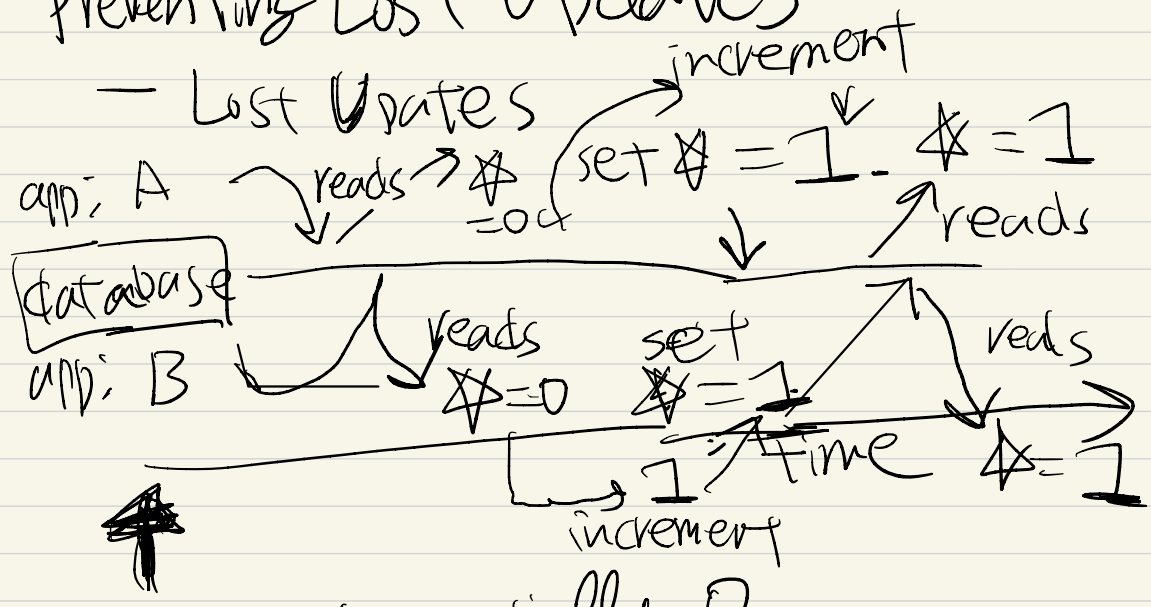
1. Read Commited
   - no dirty reads
     - can read uncommited data
   - no dirty writes
     - when concurrent writes are happened, the later commit overwrites an uncommited value.

Snapshot Isolation

- each transaction reads from a consistent snapshot of the database

Preventing Lost Updates

— Lost Updates



is int, initially 0

is 1 after the operations

should've been 2

Solutions

1. Atomic write operations

Ex: UPDATE table SET = + 1

Take an exclusive lock on the object

- Explicit locking
  - an application lock objects that are to be updated, then executes read-modify-write cycles.
- Automatically detecting lost update:
  - if transaction manager detects lost updates, it aborts the transaction
- Compare-and-set
  - allow an update to happen only if the value has not changed since you last read it
- Conflict resolution in replication
  - allow concurrent writes to create several conflicting versions of a value (siblings), and to use app code to resolve and merge versions