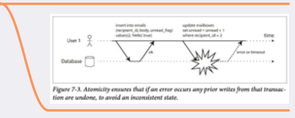
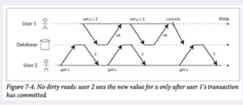
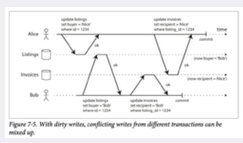
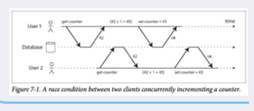
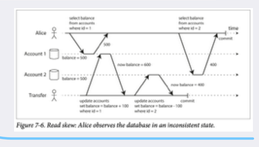
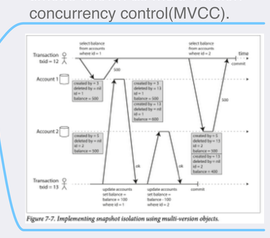
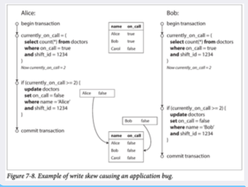
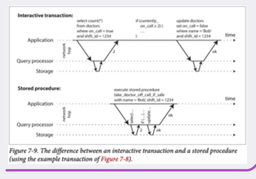
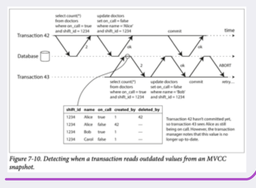
- Transactions
  - The meaning of ACID: vague
    - Atomicity
      - In general, atomic refers to something that cannot be broken down into smaller parts.
      - If a transaction was aborted, the application can be sure that it didn't change anything, so it can safely be retried.
    - Consistency
      - Consistency refers to an application-specific notion of the database being in a "good state".
      - Consistency is a property of the application.Thus, the letter C doesn't really belong in ACID.
    - Isolation
      - Isolation means that concurrently executing transactions are isolated from each other: they cannot step into each other's toes.
      - If one transaction makes several writes, then another transaction should see either all or none of those writes, but not some subset.
    - Durability
      - Durability is the promise that once a transaction has committed successfully, any data it has written will not be forgotten.
  - Slippery of Concept:large-scale system would have to abandon transaction in order to maintain good performance and high availability; Is an essential requirement for "serious applications".
  - Single-Object and Multi-Object Operations
    - Some problem
      - 
      - 
    - Single-object writes
      - Read-modify-write and Compare-and-set
      - They can prevent lost updates when several clients try to write to the same object concurrently.
    - The need for multi-object transactions
      - Many distributed datastore have abandoned multi-object transactions because they are difficult to implement across partitions .
    - Handling errors and aborts
      - If the database is in danger of violating its guarantee of atomicity, isolation, or durability, it would rather abandon the transaction entirely than allow it to remain half-finished.
  - Weak Isolation Levels
    - Read Committed
      - Two guarantees
        - When reading from the database, you will only see data that has been committed(no dirty reads).
        - When writing to the database, you will only overwrite data that has been committed(no dirty writes).
      - No dirty reads
        - 
        - A transaction has written some data to the database, but the transaction has not ye committed or aborted.Another transaction see that uncommitted data.
      - No dirty writes
        - 
        - The earlier write is part of a transaction that has not yet committed, so the later write overwrites an uncommitted value.
      - Does not prevent the race condition between two counter increments.
        - 
      - Implementing read committed
        - Prevent dirty writes by using row-level locks.
        - Prevent dirty reads:while the transaction is ongoing ,any other transactions that read the object are simply given the old value.Only when the new value is committed do transactions switch over to reading the new value.
    - Snapshot Isolation and Repeatable Read
      - Nonrepeatable read or read skew.
        - 
        - Not a lasting problem and some situations cannot tolerate such temporary inconsistency.
          - Backups
          - Analytic queries and integrity checks.
      - Snapshot isolation:each transaction reads from a consistent snapshot of the database.
      - Implementing snapshot isolation
        - The database must potentially keep several different committed versions of an object, because various in-progress transactions may need to see the state of the database at different points in time.Is known as multi-version concurrency control(MVCC).
        - 
      - Visibility rule for observing a consistent snapshot
      - Indexes and snapshot isolation
        - How indexes work? Index simply point to all versions of an object and require an index query to filter out any object versions that are not visible to the current transaction. When garbage collection removes old object versions that are no longer visible to any transaction, the corresponding index entries can also be removed.
      - Repeatable read and naming confusion
        - Many databases that implement it call it by different names.DB2 uses "repeatable read" to refer to serializability.
    - Preventing Lost Updates
      - Occur if an application reads some value from the database, modifies it, and writes back the modified value.If two transactions do this concurrently, one of the modifications can be lost, because the second write does not include the first modification.
      - solutions
        - Atomic write operations
        - Explicit locking
        - Automatically detecting lost updates
        - Compare and set
        - Conflict resolution and replication(eg: last write wins LWW)
    - Write Skew an Phantoms
      - 
  - Serializability:It guarantees that even though transactions may execute in parallel, the end result is the same as if they had executed one at a time, serially, without any concurrency.
  - Two-Phase Locking(2PL) :For around 30 years, there was only one widely used algorithm for serializability in database:two-phase locking(2PL).
    - Three techniques
      - Literally executing transaction in a serial order.
      - Two-phase locking, which for several decades was the only viable option.
      - Optimistic concurrency control techniques such as snapshot isolation.
    - Actual Serial Execution: Remove the concurrency entirely.
      - Two developments caused this rethink.
        - RAM became cheap enough.
        - Database designers realized that OLTP are usually short and only make a small number of reads and writes.
      - Encapsulating transactions in stored procedures.
        - 
        - Pros and cons of stored procedures
          - cons
            - Each database vendor has its own language for stored procedures.
            - Code running in a database is difficult to manage.
            - A database is often much more performance-sensitive than an application server.
          - pros
      - Partitioning
        - Vastly slower than single-partition.
        - Whether transactions can be single-partition depends very much on the structure of the data used by the application.
    - Two-Phase Locking(2PL) :For around 30 years, there was only one widely used algorithm for serializability in database:two-phase locking(2PL).
      - In 2PL, writers don't just block other writers; they also block readers and vice versa.Snapshot isolation has the mantra readers never block writers, and writers never block readers.
      - Implementation of two-phase locking.
        - Read shared and write exclusive
      - Performance of two-phase locking
        - Dead lock occur more frequently under 2PL.
        - Have quite unstable latencies and can be very slow at hight percentiles.
      - Predicate locks
        - The key idea here is that a predicate lock applies even to objects that do not ye exist in the database, but which might be added in the future.
      - Index-range locks
        - Predicate locks do not perform well : if there are many locks by active transactions, checking for matching locks becomes time-consuming. Most databases with 2PL actually implement index-rangelocking.
    - Serializable Snapshot Isolation(SSI):first described in 2008 ,it has the possibility of being fast enough to become the new default in the future.
      - On the top of snapshot isolation,SSI adds an algorithm for detecting serialization conflicts among writes and determining which transaction to abort.
      - Pessimistic versus optimistic concurrency control
      - Decisions based on an outdated premise
        - Detecting reads of a stale MVCC object version.
        - Detecting writes that affect prior reads
      - Detecting stale MVCC reads
        - 
      - Performance of serializable snapshot isolation.
        - Compared to two-phase locking, the big advantage of serializable snapshot isolation is that one transaction doesn't need to block waiting for locks held by another transaction.