Final project

• Reminder: topic + literature review due Friday via email!
  • Min 1 page, max 5 pages, without references
  • Follow format linked on website

• Project proposal due on October 4
  • 1 - 2 pages (without references)
  • Should have problem statement, the technical approach you will take to solve the problem, as well as a plan to evaluate your approach compared to prior work
Transactions, ACID, Concurrency Control

Slides adapted from here, here
Transactions

• A transaction is the execution of a sequence of one or more operations on a database to perform some higher-level function.

• A transaction may carry out many operations on the data retrieved from the database.

• Example: move $100 from Alice’s bank account to Bob’s bank account:
  • Check whether Alice has $100
  • Deduct $100 from Alice’s account
  • Add $100 to Bob’s bank account
Defining transaction correctness

• **ACID**
  
  • **Atomicity:** all actions in a transaction happen, or none happen
  
  • **Consistency:** if each transaction is consistent, and the database initializes in a consistent state, then it will also end up in a consistent state
  
  • **Isolation:** execution of a transaction is isolated from that of other transactions
  
  • **Durability:** if a transaction commits, then its effects persist in spite of failures
Atomicity

- Two possible outcomes of executing a transaction
  - Transaction commits \(\rightarrow\) all of its effects are reflected in the database
  - Transaction aborts \(\rightarrow\) none of its effects are reflected in the database

- Approaches for atomicity
  - Logging: logs all actions of a transaction so that it can undo aborted transactions
  - Shadow paging: DBMS makes copies of data pages and transaction makes changes to these copies; pages made visible once transaction commits
Consistency

• The data representation is logically correct

• Database consistency
  • DB accurately models the real world and follows integrity constraints (e.g., the age of a person cannot be negative)
  • Transactions in the future see the effects of past committed transactions

• Transaction consistency
  • A transaction should only change the database state in allowed ways such that the DB stays consistent after a committed transaction
  • Ensuring transaction consistency is the application’s responsibility
Isolation

• DBMS provides transactions with the illusion that they are running alone in the system

• Easy for user to reason about correctness

• How to achieve this?
  • Serialize all transactions by processing one at a time
  • Interleave transactions efficiently and correctly
Serializability

• An interleaving is correct if it is equivalent to some serial execution

• Serializable schedule: a schedule that does not interleave the actions of different transactions

• Equivalent schedules: given schedules $S_1$, $S_2$, and database state $D$, the effect of executing the $S_1$ on $D$ is identical to the effect of executing $S_2$ on $D$
Conflicting operations

- Two operations conflict if
  - They are by different transactions, and
  - They are on the same object and at least one of them is a write
- Read-Write, Write-Read, Write-Write conflicts
- DBMS support conflict serializability:
  - Two schedules are conflict equivalent iff
    - They involve the same operations of the same transactions
    - Every pair of conflicting operations is ordered in the same way in both schedules
  - A schedule is conflict serializable if it is conflict equivalent to some serial schedule
Concurrency control

- Mechanism for ensuring isolation
- DBMS uses concurrency control protocol to decide the proper interleaving of operations from multiple transactions
- Two categories
  - Pessimistic: don’t let the problems arise in the first place
  - Optimistic: assume conflicts are rare, deal with them after they happen
Basic timestamp ordering

- Transactions read and write objects without locks

- Every object is tagged with the timestamp of the last transaction that successfully did a read/write

  - $TS_w(X) = \text{write timestamp on } X$
  - $TS_r(X) = \text{read timestamp on } X$

- Check timestamp for every operation; if a transaction tries to access an object with a future timestamp, then abort and retry
Timestamp ordering

• Reads:

  • If $TS(T_i) < TS_w(X) \rightarrow$ abort $T_i$
    and restart with a new timestamp

  • Else: allow $T_i$ to read $X$, update
    $TS_r(X) = \max(TS_r(X), TS(T_i))$, make a local copy of $X$ to ensure
    repeatable reads for $T_i$
Timestamp ordering

• Writes:

  • If $TS(T_i) < TS_r(X)$ or $TS(T_i) < TS_w(X) \rightarrow$ abort $T_i$ and restart with a new timestamp

  • Else: allow $T_i$ to write to $X$, update $TS_w(X)$, make a local copy of $X$ to ensure repeatable reads for $T_i$
Weak isolation?

- Serializability is useful but enforcing it may be too expensive
- Anomalies:
  - Dirty read: reading uncommitted data
  - Unrepeatable reads: redoing a read results in a different result
  - Phantom reads: insertions or deletions result in different results for the same range query
- Isolation levels
  - SERIALIZABLE (strongest)
  - REPEATABLE READS: phantoms may happen
  - READ-COMMITTED: phantoms and unrepeatable reads may happen
  - READ-UNCOMMITTED: all anomalies can happen
Durability

- All of the changes of committed transactions should be made persistent after a crash or a restart

- Techniques:
  - Logging, checkpointing
  - Shadow paging
Today’s reading: Obladi