Designing Data Intensive Applications

CHAPTER – 7

Transactions

Youtube channel – MSDeep Singh

Subscribe for more...

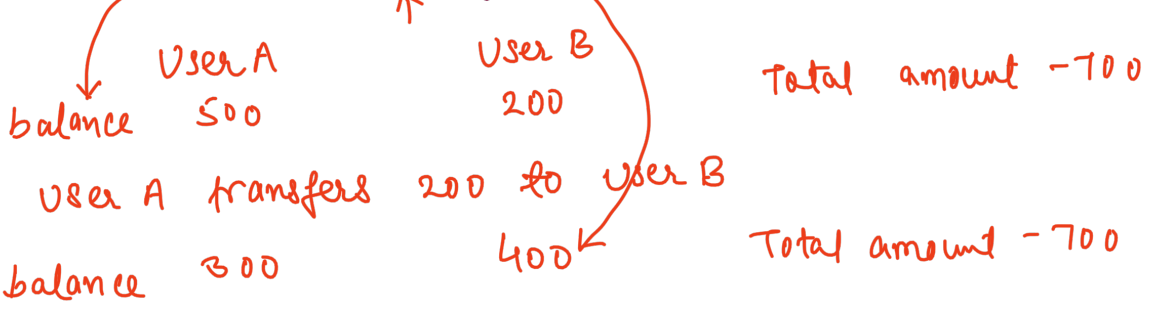
Transaction → group multiple reads and writes into single logical unit.
- executed as single operation
- either all succeeds or all fails.

ACID → safety gurantees provided by transactions.

① Atomicity → atomic operation
- Can't be broken into smaller parts.
- Successful if all parts of transaction are complete.
- for any fault, it will fail.

② Consistency → correctness of database.

Example – Transaction amount between two accounts should be balanced before and after the transaction.



- consistency is more of application property. Atomicity, Isolation and durability are database properties.

application's responsibility to achieve Consistency utilizing A.I.D from ACID.

③ Isolation → Concurrent transactions occur independently in DB without compromising Consistency.

each transaction pretend that it is only transaction being performed

the result will be same if these transactions are carried out sequentially.

④ Durability → Once transaction is committed, it will be intact even in case of System failures.

ex - usage of WAL in case of B Trees.

Replication and Durability → No technique is 100% risk free.

↳ we can just take measures to reduce it

Single object writes

↳ you're writing a 20KB JSON to DB, what if network issue after 10KB is written?

↳ Storage engines in-general provide atomicity & isolation at single object level.

atomicity → log for crash recovery - B tree example

isolation → Lock on object.

Multi-Object Transactions

use-cases

① In RDBMS, row in one table has foreign key inference to row in another table.

② multi field updates in document.

③ secondary indexes

Retrying aborted Transaction

① operation completed on server but client didn't get response due to network error.

② If error is due to overload, retry might not help.
↳ use exponential backoff & jitter.

③ Retry for only temporary errors, not permanent.

Weak Isolation Levels

↳ A reads data and B modifies data at some time.
↳ A and B try to write data to same record concurrently.

Transaction Isolation → System works in manner as everything is working in sequence.
↳ Serializable isolation.

↳ As strong isolation support has performance cost, some databases tend to use weak isolation levels.

① Read Committed

↳ On Read, you'll see data that is committed. (no dirty reads)
↳ On write, you'll only overwrite data that is committed. (no dirty writes)

use locks { To avoid this issue, transaction B can wait till A is committed or aborted.

→ ① use locks → can lead to more wait time.
→ ② Hold both old and new value for which update is going on.
↳ return old value till new one is committed.

Disadvantage → Read committed can cause anomaly on read
↳ Read skew
↳ Timing anomaly

Example

Account A	Account B
$500	$500
↳ transfer $100	

For time being → $400 if read from accounts

$500 ↳ New value not yet committed

Total amount → $900 ↳ this issue could occur in - time on Read.

Total amount of Sandeep has in two accounts - $1000

Solution to above

② Snapshot Isolation → Each transaction reads from consistent snapshot of database.

extremely beneficial for long running read only queries
↑ backups / analytics

How to implement?

↳ write locks to prevent dirty writes.

↳ no locks required on reads.

↳ maintains several versions of objects (unlike 2 in read committed)
↑ Multi version Concurrency Control (MVCC)

Indexes with snapshot isolation

↳ Index point to all versions → index query can filter out object versions not visible to current transaction.

↳ use append only / copy on write B-Tree pages.

↳ don't overwrite pages, create new copy of modified page.

↳ write transaction creates new B-Tree root
↑ old root is consistent snapshot.

Lost Update Problem

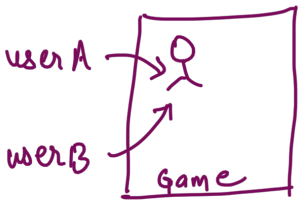
① read from db
② modify value
③ write back

} two transactions perform concurrently
↑ one modification could be lost

Solutions

@ Atomic write operations → atomic update operations.
↑ by acquiring lock

(b) Explicit locking → handle locking mechanism in application.
↑ not provided by DB
atomic operations not sufficient
not possible to handle via DB query
ex multiplayer game.



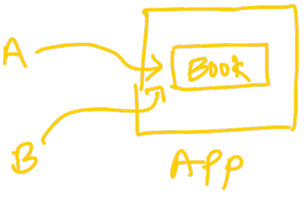
Both users trying to perform action at same piece.

(c) Detect lost updates → allow to execute in parallel and fail if lost update is detected.
can be retried

(d) Compare and set → On update, check the current value is same as what it was read earlier.

Write Skew

Example - Book meeting room.
↳ meeting room should not be booked twice.
↳ Two users tried to book at same time.
↳ check availability → snapshot isolation
↳ room is available, so booked for both?
→ use Serializable Isolation
→ lock multiple rows that are accessed for transaction.



Phantom → write in one transaction change result of search query in another transaction.

Serializability

Serializable Isolation → transactions may execute in parallel but they have same effect as executed sequentially.
↑ no concurrency

(1) Actual Serial Execution → remove concurrency.
↳ no overhead of locking.

② Encapsulate transactions in stored procedures

↳ transactions should be short.

Book Flight → look from flight → select → book seat → payment

Single transaction

T1 T2 T3 T4

To support serial execution, transactions should be short.

to avoid Lag due to locks

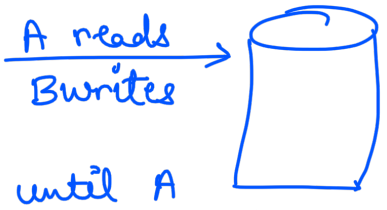
↳ application should submit entire transaction code to DB ahead of time.

↑ stored procedure.

no I/O wait
no concurrency overhead.

③ Two Phase Locking (2PL)

↳ Several transactions are allowed to write to an object unless no-one is modifying it.



B should wait until A commits/aborts

B wait until A commits/aborts

↑ there might be situation of deadlock if lock is not released. → transaction is aborted by DB in this case.

See you again in next chapter...

Subscribe more more such content.

YouTube - MsDeep Singh

Happy Learning ☺