Beyond Relational Model

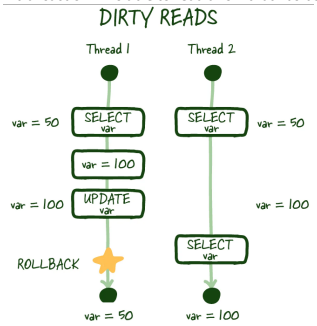
Monday, January 27, 2025 9:40 AM

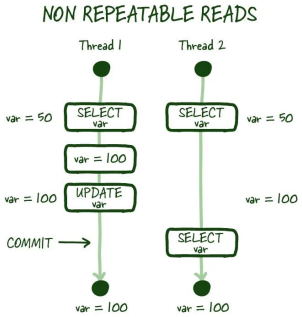
Benefits of the Relational Model
- (Mostly) Standardized Data Model and Query Language
- ACID Compliance
  - Atomicity, Consistency, Isolation, Durability
- Works well with highly structured data
- Can handle large amounts of data
- Well understood, lots of tooling, lots of experience

Many way that a RDBMS increases efficiency:
- Indexing
- Directly controlling storage
- Column oriented storage vs row oriented storage
- Query optimization
- Caching/prefetching
- Materializing views
- Precompiled stored procedures
- Data replication and partitioning

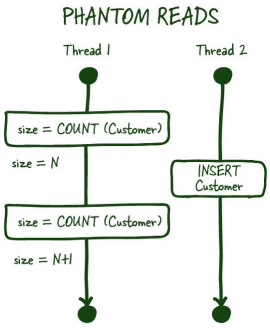
Transaction – A sequence of one or more of the CRUD operations performed as a single, logical unit of work
- Either the entire sequence succeeds (COMMIT)
- OR the entire sequence fails (ROLLBACK or ABORT)
- Help ensure...
  - Data integrity
  - Error recovery
  - Concurrency control
  - Reliable data storage
  - Simplified error handling

ACID Properties
- **Atomicity**
  - Transaction is treated as an atomic unit - it is fully executed or no parts of it are executed
- **Consistency**
  - a transaction takes a database from one consistent state to another consistent state
  - consistent state - all data meets integrity constraints
- **Isolation**
  - 2 transactions T1 and T2 are being executed at the same time but cannot affect each other
  - If both T1 and T2 are reading the data – no problem
  - If T1 is reading the same data that T2 may be writing, can result in:
    - Dirty Read
    - Non-repeatable Read
    - Phantom Reads
- **Durability**
  - Once a transaction is completed and committed, its changes are permanent
  - Even in event of system failure, committed transactions are preserved
    - One of the ways this is handled is with logging

- **Dirty Read**
  - A transaction T1 is able to read a row that has been modified by another transaction T2 that hasn't yet executed a COMMIT
  - 

- **Non-repeatable Read**
  - 2 queries in a single transaction T1 execute a SELECT but get different values because another transaction T2 has changed data and COMMITTED
  - 

- **Phantom Reads**
  - when a transaction T1 is running and another transaction T2 adds or deletes rows from the set T1 is using



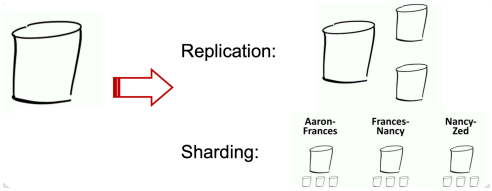
Flaws with Relational Model

- Sometimes Schemas evolve over time
- Not all apps may need the full strength of ACID compliance
- Joins can be expensive
- Alot of data is semi-structured or unstructured (JSON, XML, etc)
- Horizontal scaling presents challenges
- Some apps need something more performant (real time, low latency systems)

Scalability – Up or Out?

- ***Conventional Wisdom:*** Scale vertically (up, with bigger more powerful systems) until demands of high availability make it necessary to scale out to some kind of distributed computing model
  - **Distributed system** is "a collection of independent computers that appear to its users as one computer
  - Characteristics:
    - Computers operate concurrently
    - Computers fail independently
    - No shared global clock
- ***But why?*** Scaling up is easier - no need to really modify your architecture. But there are practical and financial limits
- ***However:*** There are modern systems that make horizontal scaling less problematic.

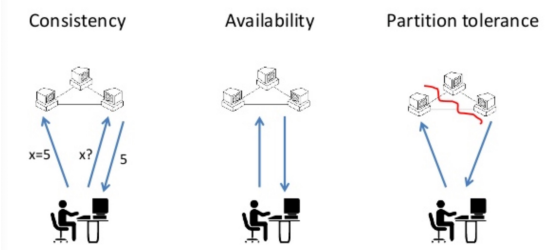
Distributed Data Sources



Replication – making multiple copies
Sharding – splitting on some attribute

- Data is stored on a >1 node, typically replicated
  - So each block of data is available on N nodes
- Can be relational or non-relational
  - MySQL and PostgreSQL support replication and sharding
  - CockroachDB - new player on the scene
  - Many NoSQL systems support one or both models
- Remember: Network partitioning is inevitable!
  - network failures, system failures
  - Overall system needs to be Partition Tolerant
    - System can keep running even w/ network partition

The CAP Theorem

- The CAP Theorem states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:
  - Consistency - Every read receives the most recent write or error thrown
  - Availability - Every request receives a (non-error) response - but no guarantee that the response contains the most recent write
    - Can I get to it all the time?
  - Partition Tolerance - The system can continue to operate despite arbitrary network issues.
- 

If you cannot limit the number of faults, requests can be directed to any server, and you insist on serving every request, then you cannot possibly be consistent.

- **Consistency + Availability:** System always responds with the latest data and every request gets a response, but may not be able to deal with network issues

issues

- **Consistency + Partition Tolerance:** If system responds with data from a distributed store, it is always the latest, else data request is dropped.
- **Availability + Partition Tolerance:** System always sends response based on distributed store, but may not be the absolute latest data.