Briefing Document: Moving Beyond the Relational Model

Source: Excerpts from "03 - Moving Beyond the Relational Model.pdf" by Mark Fontenot, PhD, Northeastern University.

Date: October 26, 2023

Overview: This briefing document summarizes the key themes and important ideas presented in the provided excerpts. The document outlines the benefits and performance aspects of the relational model, delves into the concept of transaction processing and its ACID properties, and then discusses the limitations of relational databases in modern contexts, particularly concerning scalability and handling diverse data types. Finally, it introduces distributed systems and the fundamental CAP Theorem, highlighting the trade-offs involved in designing distributed data stores.

Main Themes and Important Ideas:

1. Benefits and Performance of the Relational Model:

* **Standardization and Familiarity:** The relational model boasts a "(Mostly) Standard Data Model and Query Language," making it widely understood and supported.
* **ACID Compliance:** A core strength is its support for ACID properties, ensuring reliable transaction processing:
    * **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" where "all data meets integrity constraints."
    * **Isolation:** Concurrent transactions "cannot affect each other." The document further elaborates on potential isolation issues like "Dirty Read," "Non-repeatable Read," and "Phantom Reads."
    * **Durability:** "Once a transaction is completed and committed successfully, its changes are permanent."
* **Efficient Data Handling (within its domain):** RDBMS employs various techniques to enhance efficiency, including:
    * Indexing
    * Control over storage (column vs. row oriented)
    * Query optimization
    * Caching/prefetching
    * Materialized views
    * Precompiled stored procedures
    * Data replication and partitioning

2. Transaction Processing and ACID Properties in Detail:

*   **Transaction Definition:** A "sequence of one or more of the CRUD operations performed as a single, logical unit of work." Transactions ensure either a full `COMMIT` or a complete `ROLLBACK` (or `ABORT`) in case of failure.
*   **Importance of Transactions:** They are crucial for:
    *   Data Integrity
    *   Error Recovery
    *   Concurrency Control
    *   Reliable Data Storage
    *   Simplified Error Handling
*   **Illustrative Example:** The provided SQL code for a `transfer` procedure demonstrates the implementation of a transaction with `START TRANSACTION`, `UPDATE`, `INSERT`, `ROLLBACK` (with a custom error message: "'Transaction rolled back: Insufficient funds'"), and `COMMIT`. This example highlights the atomicity and consistency aspects of ACID.

3. Limitations of the Relational Model in Modern Scenarios:

*   **Schema Evolution:** Relational databases can struggle with frequently changing data structures.
*   **Overkill for Some Applications:** Not all applications require the full rigor of ACID compliance.
*   **Expensive Joins:** Operations involving joining data across multiple tables can become performance bottlenecks.
*   **Handling Unstructured/Semi-structured Data:** Relational models are less naturally suited for data formats like JSON and XML.
*   **Horizontal Scaling Challenges:** Scaling relational databases across multiple servers (scaling out) has historically been complex.
*   **Performance Requirements:** Some applications demand lower latency and higher performance than traditional RDBMS can consistently offer.

4. The Shift Towards Distributed Systems and Storage:

*   **Scaling Up vs. Scaling Out:** Traditionally, vertical scaling (upgrading hardware) was favored due to its simplicity. However, limitations in cost and availability necessitate horizontal scaling (distributing data and workload across multiple machines).
*   **Distributed System Definition:** A distributed system is defined as "a collection of independent computers that appear to its users as one computer." - Andrew Tennenbaum.
*   **Characteristics of Distributed Systems:**
    *   Concurrent operation of computers.
    *   Independent computer failures.
    *   Absence of a shared global clock.
*   **Distributed Storage Approaches:**
    *   **Single Main Node:** (Implied as the traditional approach before full distribution).
    *   **Distributed Data Stores:** Data is spread across multiple nodes, often with replication for redundancy. "i.e. each block of data is available on N nodes."

    *   **Distributed Databases:** Can be relational (e.g., MySQL and PostgreSQL with replication/sharding, CockroachDB) or non-relational (NoSQL systems).
*   **Network Partitioning:** "Network partitioning is inevitable!" due to network or system failures. A robust distributed system must be "Partition Tolerant" and "can keep running even w/ network partition."

5. The CAP Theorem: Trade-offs in Distributed Data Stores:

*   **Statement:** "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 (CAP Definition):** "Every read receives the most recent write or error thrown." *Note: This differs from ACID consistency.*
    *   **Availability:** "Every request receives a (non-error) response - but no guarantee that the response contains the most recent write."
    *   **Partition Tolerance:** "The system can continue to operate despite arbitrary network issues."
*   **Database View of CAP:**
    *   **Consistency*:** "Every user of the DB has an identical view of the data at any given instant."
    *   **Availability:** "In the event of a failure, the database remains operational."
    *   **Partition Tolerance:** "The database can maintain operations in the event of the network's failing between two segments of the distributed system."
*   **CAP Trade-offs:**
    *   **Consistency + Availability:** May struggle with network partitions.
    *   **Consistency + Partition Tolerance:** May sacrifice availability (drop requests during partitions).
    *   **Availability + Partition Tolerance:** May sacrifice consistency (return stale data during partitions).
*   **Real-world Interpretation:** While often simplified as always having to give up one of the three, a more nuanced understanding is that in the presence of network faults and a requirement to serve all requests, consistency becomes impossible.

Quotes:

*   "(Mostly) Standard Data Model and Query Language"
*   "transaction is treated as an atomic unit - it is fully executed or no parts of it are executed" (Atomicity)
*   "a transaction takes a database from one consistent state to another consistent state" (Consistency)
*   "all data meets integrity constraints" (Consistent State)
*   "Two transactions T_1 and T_2 are being executed at the same time but cannot affect each other" (Isolation)
*   "a transaction T_1 is able to read a row that has been modified by another transaction T_2 that hasn't yet executed a COMMIT" (Dirty Read)

* "two queries in a single transaction T_1 execute a SELECT but get different values because another transaction T_2 has changed data and COMMITTED" (Non-repeatable Read)
* "when a transaction T_1 is running and another transaction T_2 adds or deletes rows from the set T_1 is using" (Phantom Reads)
* "'Transaction rolled back: Insufficient funds'" (Example ROLLBACK message)
* "'Transaction committed successfully'" (Example COMMIT message)
* "Once a transaction is completed and committed successfully, its changes are permanent." (Durability)
* "a collection of independent computers that appear to its users as one computer." -Andrew Tennenbaum (Definition of a distributed system)
* "Network partitioning is inevitable!"
* "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:" (CAP Theorem)
* "Every read receives the most recent write or error thrown" (CAP Consistency)
* "Every request receives a (non-error) response - but no guarantee that the response contains the most recent write" (Availability)
* "The system can continue to operate despite arbitrary network issues." (Partition Tolerance)

Conclusion:

The excerpts provide a clear transition from the well-established benefits and mechanisms of the relational model, particularly its ACID properties for reliable transaction processing, to the challenges it faces in modern, large-scale, and data-diverse applications. The introduction of distributed systems and the CAP Theorem highlights the fundamental trade-offs that architects and developers must consider when choosing and designing data storage solutions beyond the traditional relational database. The inevitability of network partitions necessitates a conscious decision regarding the prioritization of consistency and availability in distributed environments.