Moving Beyond the Relational Model

1. Benefits of the Relational Model

The relational model has been the foundation of most database systems due to several advantages:

- **Standardized Data Model & Query Language**: SQL is widely used and understood.
- **ACID Compliance**: Ensures reliability and consistency in transactions.
- **Highly Structured Data Handling**: Works best when data conforms to a well-defined schema.
- **Scalability & Efficiency**: Optimized to handle large datasets.
- **Mature Ecosystem**: Many tools, libraries, and experienced professionals available.

2. Relational Database Performance

Relational Database Management Systems (RDBMS) improve efficiency through various mechanisms:

- **Indexing**: Improves query performance by reducing the number of rows scanned.
- **Storage Optimization**:
    - **Row-oriented storage** (good for transactional workloads).
    - **Column-oriented storage** (better for analytical queries).
- **Query Optimization**: Query execution plans are optimized to minimize computation.
- **Caching & Prefetching**: Stores frequently accessed data in memory.
- **Materialized Views**: Precomputed query results that speed up retrieval.
- **Precompiled Stored Procedures**: Reduces parsing and execution time.
- **Data Replication & Partitioning**: Enhances fault tolerance and scalability.

3. Transaction Processing

A **transaction** is a group of operations performed as a single unit. It follows:

- **COMMIT**: If all operations succeed.
- **ROLLBACK (ABORT)**: If any operation fails, undo all changes.

Why Transactions Are Important

- **Data Integrity**: Prevents partial updates.
- **Error Recovery**: Ensures changes are reversible.
- **Concurrency Control**: Prevents conflicts between multiple transactions.
- **Reliable Data Storage**: Maintains consistency.
- **Simplified Error Handling**: Avoids manual correction of inconsistencies.

4. ACID Properties

Ensures reliability in relational databases:

1. **Atomicity**: A transaction is all-or-nothing.
2. **Consistency**: A transaction transforms the database from one valid state to another.
3. **Isolation**: Concurrent transactions do not interfere with each other.
4. **Durability**: Once committed, changes are permanent.

Isolation Issues

- **Dirty Read**: Reading uncommitted data from another transaction.
- **Non-repeatable Read**: A repeated query within the same transaction yields different results.
- **Phantom Reads**: Changes in rows appearing/disappearing due to other transactions.

Example: Bank Transfer

- A stored procedure ensures funds are only transferred if sufficient balance exists.
- If insufficient, the transaction is **rolled back**.
- Ensures **atomicity** (either full transfer or none) and **consistency** (balances remain valid).

5. Limitations of Relational Databases

Despite their strengths, relational databases are not always the best solution:

- **Schema Evolution**: Changing data structures can be difficult.
- **Joins Can Be Expensive**: Large datasets may cause performance issues.
- **Handling Semi-Structured/Unstructured Data**: JSON, XML, and other formats are not natively supported.
- **Scalability Challenges**: Scaling horizontally is difficult.

6. Scalability: Scaling Up vs. Scaling Out

Scaling Up (Vertical Scaling)

- **Adding more power (CPU, RAM, SSDs) to a single server.**
- Easier to implement but has limits in hardware and cost.

Scaling Out (Horizontal Scaling)

- **Adding more machines to distribute the workload.**
- More complex but provides higher availability and performance.

7. Distributed Systems

A **distributed system** consists of multiple computers that function as a single system.
Characteristics:

- **Concurrent Operation**: Nodes work together.
- **Independent Failure**: One failure doesn't bring down the system.
- **No Shared Global Clock**: Synchronization must be managed.

Distributed Storage Models

- **Single Main Node**: Centralized but supports replication.
- **Distributed Data Stores**: Data is split across multiple nodes.
    - MySQL, PostgreSQL support replication/sharding.
    - NoSQL databases are inherently distributed.

Key Challenge: Network Partitioning

- **Network failures are inevitable**, so systems must handle partial failures.

8. CAP Theorem

A fundamental principle in distributed systems:

Three Guarantees (Only Two Can Be Achieved at a Time)

1. **Consistency (C)**: Every read receives the latest write or an error.
2. **Availability (A)**: Every request receives a response (though it may not be the latest data).
3. **Partition Tolerance (P)**: The system continues working despite network failures.

Trade-offs in Distributed Databases

Guarantee	What It Means
C + A	Latest data always returned, but failures cause downtime.
C + P	Ensures latest data, but some requests may fail.
A + P	System always responds, but data may be outdated.

Reality of CAP Theorem

- No system is 100% **consistent** and **available** in the presence of network partitions.
- Instead, databases pick a balance based on needs:
  - **CP Systems (Consistency + Partition Tolerance)**: Ensure accurate data but may have downtime (e.g., Zookeeper).
  - **AP Systems (Availability + Partition Tolerance)**: Prioritize uptime but may return stale data (e.g., DynamoDB).
  - **CA Systems (Consistency + Availability)**: Possible only in non-distributed setups.