1. Benefits of the Relational Model

- Standard data model and query language
- ACID compliance (Atomicity, Consistency, Isolation, Durability)
- Works well with highly structured data
- Can handle large volumes of data
- Well-supported with tools and experience

2. Relational Database Performance Enhancements

- Indexing
- Storage control (column vs. row orientation)
- Query optimization
- Caching & prefetching
- Materialized views
- Precompiled stored procedures
- Data replication & partitioning

3. Transaction Processing

- **Transaction**: A sequence of CRUD operations treated as one logical unit
    - Entire sequence **commits** or **rolls back**
- Ensures:
    - Data integrity
    - Error recovery
    - Concurrency control
    - Reliable data storage
    - Simplified error handling

4. ACID Properties

- **Atomicity**: Transaction fully completes or doesn't execute at all
- **Consistency**: Transitions DB from one valid state to another
- **Isolation**: Transactions don’t interfere with each other
    - Issues:
        - Dirty Reads: Read uncommitted data
        - Non-repeatable Reads: Same query returns different results
        - Phantom Reads: Row set changes during transaction
- **Durability**: Once committed, changes are permanent—even after failure

5. SQL Transaction Example: Transfer Money

- Transfers money between two accounts with rollback if funds are insufficient
- Uses:
    - `START TRANSACTION`
    - `UPDATE` accounts
    - Conditional check
    - `ROLLBACK` or `COMMIT`
    - Inserts into a `transactions` table

6. Why Move Beyond Relational?

- Schema evolution over time
- Not all applications require full ACID
- Expensive joins
- Increasing semi-structured/unstructured data (e.g., JSON, XML)
- Difficult horizontal scaling
- Need for low latency / real-time systems

7. Scalability: Vertical vs Horizontal

- **Scale Up (Vertical)**:
    - Bigger servers, simpler
    - Eventually hits cost/performance limits
- **Scale Out (Horizontal)**:
    - Distributed computing models
    - Modern tools make this easier

8. Distributed Systems

- “A collection of independent computers that appear as one” *(Andrew Tennenbaum)*
- Key properties:
    - Concurrent operation
    - Independent failures
    - No global clock

9. Distributed Data Stores

- Data stored on multiple nodes (replicated or partitioned)
- Examples:
    - MySQL/PostgreSQL (support replication/sharding)
    - CockroachDB
    - Many NoSQL databases
- Systems must be **Partition Tolerant**
    - Must operate despite network/system failures

10. CAP Theorem

- Cannot achieve all three simultaneously:
    1. **Consistency**: Latest write always read
    2. **Availability**: Every request gets a response
    3. **Partition Tolerance**: Operates despite network issues

11. CAP Theorem Trade-offs

- **Consistency + Availability**: Not partition-tolerant
- **Consistency + Partition Tolerance**: May sacrifice availability
- **Availability + Partition Tolerance**: May sacrifice consistency

Note: CAP's "Consistency" ≠ ACID's "Consistency"

12. CAP Theorem Real-World Takeaway

- You must trade off between:
    - Consistency
    - Availability
    - Partition Tolerance
- You can't have all three under fault conditions