# ALL YOU NEED IS NORMALIZATION

Good database design is the foundation for good products.

PREDICTABLITIY DOESN'T HAPPEN BY ACCIDENT

DATABASE DESIGN AND ADMINISTRATION

# A CASE STUDY IN BUILDING RELIABLE, SCALABLE, AND SECURE SQL SERVER PLATFORMS

**Author:** Jayanth Kurup
**Role:** Database & Data Platform Consultant

---

## EXECUTIVE SUMMARY

Modern enterprise systems depend heavily on databases that were often designed under assumptions that no longer hold true. As data volume, concurrency, and analytical demand increase, database platforms that once appeared stable begin to show signs of strain. Performance degradation becomes harder to predict, recovery processes reveal gaps, and security controls grow inconsistent as systems evolve organically.

This technical paper presents a real-world case study in database design and administration, focused on Microsoft SQL Server environments across on-premises and cloud deployments. It demonstrates how applying reliability engineering principles at the data platform layer leads to measurable business outcomes: predictable performance, reduced downtime, faster recovery, and stronger security posture.

Rather than relying on isolated “best practices,” this paper emphasizes system-level thinking. It shows how understanding query behaviour, data distribution, concurrency patterns, and operational workflows together enables sustainable improvements that scale with the business.

---

## 1. THE CUSTOMER PROBLEM SPACE

### 1.1 Performance Degradation as Systems Grow

Performance issues in mature transactional systems rarely appear suddenly. More commonly, they emerge gradually as datasets grow, access patterns shift, and concurrency increases. Queries that previously executed within acceptable timeframes begin to exhibit unpredictable latency. CPU usage rises even when application logic remains unchanged, and minor deployments can introduce regressions that are difficult to diagnose.

In most cases, these symptoms are not caused by poorly written queries. Instead, they arise from subtle drift within the system. Statistics no longer accurately represent data distribution, index structures become misaligned with real access patterns, and execution plans cached under earlier assumptions are reused in contexts where they are no longer optimal. Over time, the query

optimizer is forced to make decisions based on incomplete or misleading information, resulting in inefficient execution strategies despite logically sound SQL code.

---

**1.2 High Availability That Exists Only on Paper**

High availability and disaster recovery are often treated as architectural milestones rather than operational capabilities. Many organizations invest in technologies such as Availability Groups, failover clustering, or cloud-based backups, yet stop short of validating how these systems behave under real failure conditions.

Failover procedures may be documented but never executed end-to-end. Recovery time objectives (RTO) and recovery point objectives (RPO) are assumed rather than measured. Application dependencies on primary replicas remain implicit and untested. When failures do occur, downtime is frequently extended not because the technology failed, but because assumptions were never validated in practice.

Industry research consistently shows that recovery plans which are not rehearsed tend to fail during real incidents, highlighting the importance of treating HA/DR as an operational discipline rather than a configuration checkbox.

---

**1.3 Concurrency and Locking Failures in OLTP Systems**

As transactional workloads scale, concurrency behavior becomes a dominant factor in system stability. Locking and blocking issues often manifest under peak load conditions, where blocking chains escalate into timeouts and deadlocks emerge unpredictably. These failures are frequently misattributed to hardware limitations or generic “performance problems,” obscuring their true causes.

In practice, concurrency issues typically stem from architectural decisions such as inappropriate isolation levels, hotspot access patterns, or indexing strategies that do not reflect write-heavy workloads. While newer SQL Server versions introduce improvements in locking and optimistic concurrency, these features must be applied with a clear understanding of their trade-offs to avoid introducing new operational risks.

---

**1.4 Inconsistent Security and Governance**

Database security often evolves incrementally, shaped by urgent access requests rather than deliberate design. Permissions accumulate over time, sensitive data is only partially classified, and auditing is implemented reactively in response to incidents or compliance requirements.

This organic growth leads to elevated risk. Excessive privileges increase the impact of compromised credentials, and lack of visibility makes it difficult to answer basic questions about who accessed sensitive data and when. In regulated environments, these gaps translate directly into compliance exposure.

---

## 2. Consulting Approach: Reliability Engineering for SQL Server

The approach applied in this case study was grounded in reliability engineering rather than isolated optimization. Instead of addressing symptoms independently, the database platform was evaluated as a cohesive system with observable behavior under load, failure, and recovery scenarios.

This approach focused on four foundational principles:

1. Platform correctness and observability
2. Performance driven by data behavior rather than assumptions
3. Operationally validated high availability
4. Security designed intentionally rather than accumulated over time

By treating the database as a long-lived system rather than a static configuration, it became possible to introduce targeted improvements that increased stability without adding unnecessary complexity.

---

## 3. High Availability: From Architecture to Validation

### 3.1 Designing for Real Failure Scenarios

High availability design began by identifying realistic failure modes rather than abstract architectural goals. These included node failures, storage disruptions, network partitions, and planned maintenance events. Availability Groups were selected due to their balance of resilience, scalability, and support for read workloads, but design decisions explicitly considered quorum behavior, commit modes, listener dependencies, and application connection semantics.

---

### 3.2 Lab-Validated Implementation

A defining characteristic of this engagement was the insistence on validation before production rollout. The implementation process included clean installation, baseline configuration, explicit enabling of Always On features, controlled creation of Availability Groups, and repeated manual failover exercises.

Failovers were observed not only at the database level, but also from the perspective of application behavior and monitoring systems. This process surfaced issues such as connection strings lacking multi-subnet support, jobs implicitly tied to primary replicas, and alerting systems that generated misleading signals during transitions. Addressing these issues early significantly reduced real outage risk.

---

## 4. Performance Tuning Based on Data Reality

### 4.1 Moving Beyond Query-Level Fixes

Rather than starting with query rewrites or hints, performance analysis began with understanding the data itself. This included examining distribution skew, correlation between predicates, and data growth patterns over time. The analysis revealed that several performance regressions were driven by inaccurate cardinality estimation caused by skewed data and outdated statistics.

---

### 4.2 Statistics, Plan Stability, and Predictable Performance

Stability was improved by ensuring statistics were maintained with appropriate sampling, identifying queries sensitive to parameter sniffing, and applying plan-level interventions only where justified. Instead of disabling optimizations globally, decisions were made on a case-by-case basis using measurable evidence.

This selective approach aligns with documented research showing that targeted intervention produces more sustainable performance improvements than broad configuration changes.

---

### 4.3 Concurrency Modernization

Locking and blocking issues were addressed by reviewing isolation levels in the context of workload behavior, identifying contention hotspots, and aligning index design with write concurrency. Where newer engine features provided benefits, guidance was given on operational implications and monitoring strategies to ensure regressions could be detected early.

---

## 5. Handling Time Correctly: A Practical Example

Time handling errors often produce subtle but costly business issues. In this case, reporting discrepancies were traced to inconsistent handling of local time, UTC, and daylight saving transitions. The solution standardized the use of datetimeoffset for persisted timestamps and enforced explicit time zone conversions using AT TIME ZONE.

This eliminated an entire class of errors that had previously required manual reconciliation during reporting cycles and improved trust in time-based analytics.

---

## 6. Security as an Architectural Concern

Security improvements focused on implementing least privilege in practice rather than theory. Access requirements were mapped to roles, implicit permissions were removed, and role-based access patterns were introduced. Data classification was used as a design tool to identify sensitive columns and guide auditing efforts toward meaningful events.

This approach reduced risk while simplifying ongoing administration and compliance reporting.

---

## 7. Business Outcomes

The technical improvements delivered measurable business value. Incident frequency decreased as performance became more predictable and failure scenarios were validated rather than assumed. Recovery times improved due to clearer operational procedures, and performance scaled more smoothly as data volumes increased.

Equally important, the database platform shifted from a reactive liability to a dependable foundation capable of supporting future analytics and AI initiatives.

---

## 8. Conclusion

Reliable databases are not the result of isolated best practices. They emerge from disciplined design, continuous validation, and a deep understanding of system behavior under real workloads. This case study demonstrates how applying reliability engineering principles to database design and administration transforms SQL Server platforms into resilient, scalable systems that support long-term business growth.

---

**References**


1. Gray, J., & Reuter, A. (1993). *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann.
2. Microsoft. *SQL Server Query Processing Architecture Guide*. https://learn.microsoft.com/sql/relational-databases/query-processing-architecture-guide
3. Microsoft. *Statistics and Cardinality Estimation*. https://learn.microsoft.com/sql/relational-databases/statistics/statistics
4. Microsoft. *Always On Availability Groups Overview*. https://learn.microsoft.com/sql/database-engine/availability-groups/windows/overview-of-always-on-availability-groups-sql-server
5. Microsoft. *Locking and Row Versioning Guide*. https://learn.microsoft.com/sql/relational-databases/sql-server-transaction-locking-and-row-versioning-guide
6. NIST. *SP 800-53 Rev. 5 – Security and Privacy Controls*. https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final
7. ISO/IEC 27001:2022. *Information Security Management Systems*.