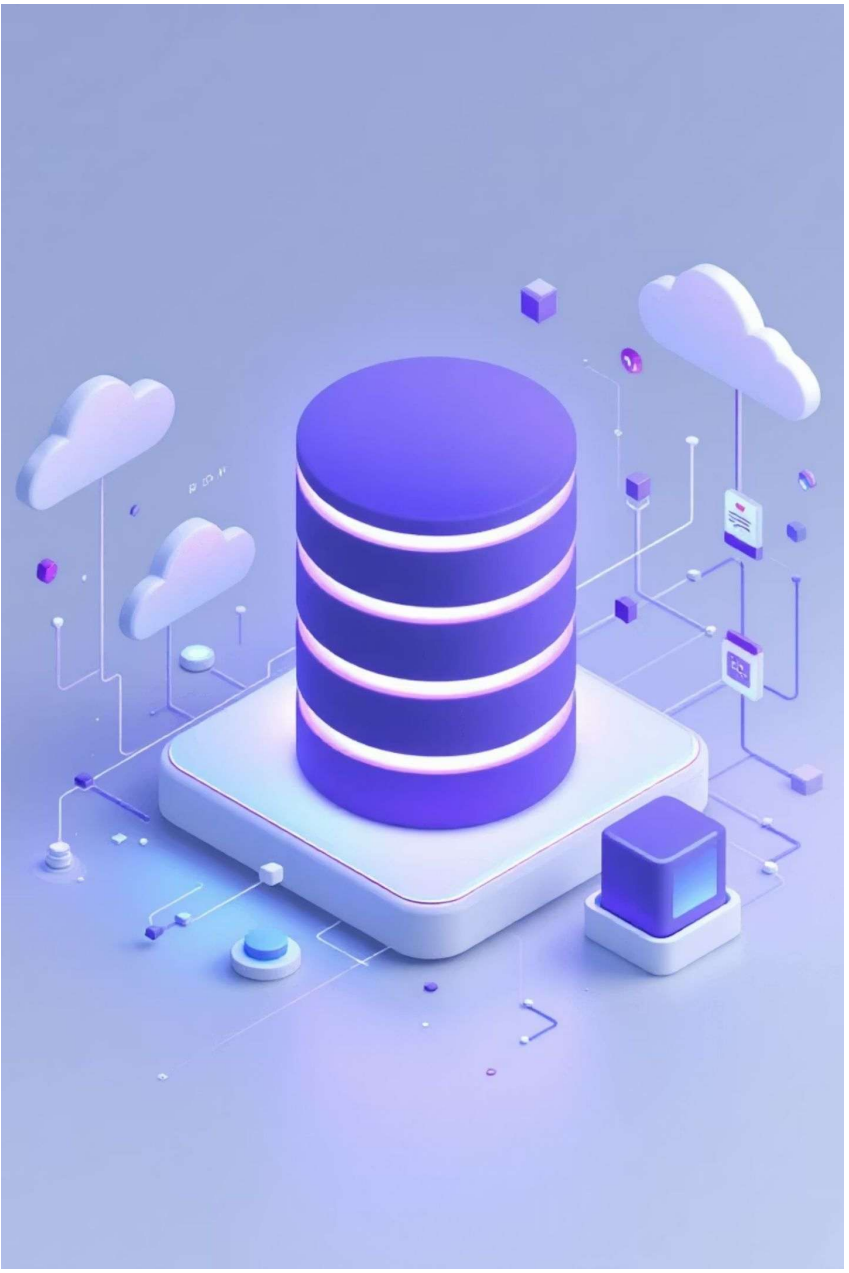


MERiT

NoSQL vs NewSQL: Understanding Modern Database Technologies

Welcome to our comprehensive exploration of NoSQL and NewSQL database technologies. This presentation will guide you through the fundamental concepts, use cases, and practical implementations of these modern database approaches that are transforming how we store and process data.

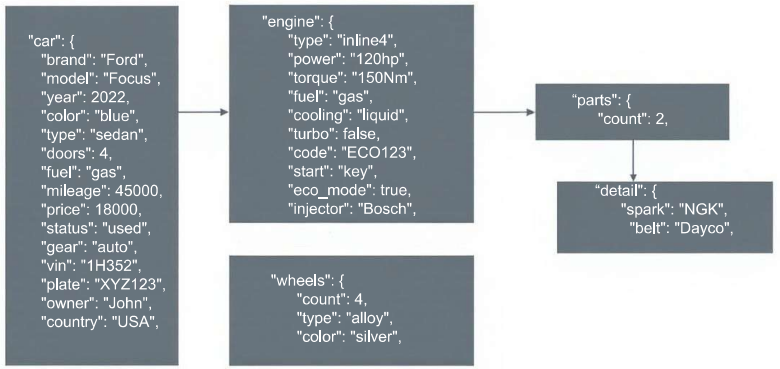
We'll examine how different database paradigms address various challenges in today's data-intensive applications, from flexible schemas to horizontal scalability and transactional consistency.

NoSQL Basics: Document Databases

Document-oriented databases like MongoDB represent a fundamental shift in data storage philosophy. Instead of the rigid table structures found in traditional relational databases, document databases store data in flexible, JSON-like documents.

Key characteristics include:

- Schema flexibility allowing each document to have different fields
- Native support for nested data structures
- Horizontal scaling through automatic sharding
- Query capabilities optimized for document retrieval



This approach significantly accelerates development cycles by eliminating complex migrations and allowing data models to evolve naturally alongside application requirements.

Document databases excel in scenarios where data structures may change frequently or where complex, hierarchical data needs to be stored as a single entity. Their flexible schema makes them particularly suitable for content management systems, user profiles, and product catalogs.

NoSQL Basics: Key-Value and Column-Store

Key-Value Stores

Redis exemplifies the simplicity and power of key-value databases. By focusing on a straightforward data model of key-value pairs stored in memory, Redis achieves extraordinary performance metrics:

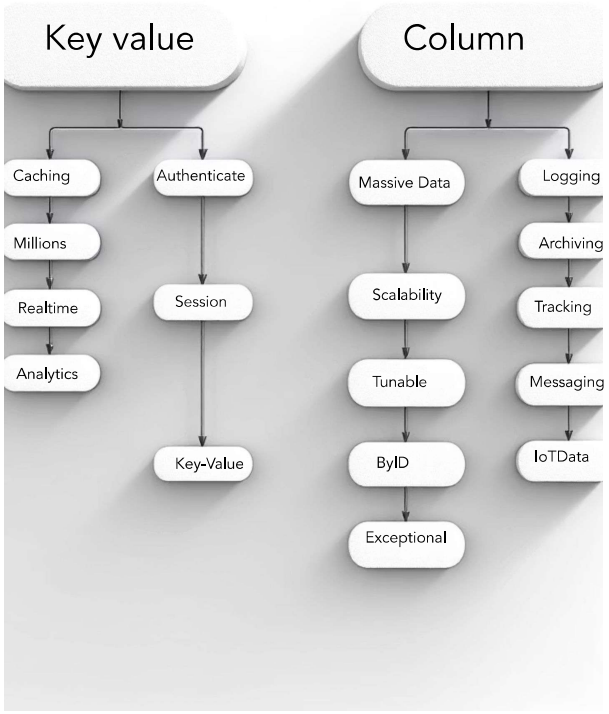
- Sub-millisecond response times
- Millions of operations per second
- Ideal for caching, session management, and real-time analytics

Column-Store Databases

Cassandra organizes data by columns rather than rows, creating a fundamentally different approach to data storage:

- Exceptional write throughput at massive scale
- Linear scalability across distributed clusters
- Tunable consistency models for different workloads
- Built for global distribution and fault tolerance

These specialized NoSQL variants address specific performance challenges. Key-value stores optimize for speed and simplicity, while column stores excel at distributing massive datasets across multiple nodes while maintaining availability.



Introduction to NewSQL

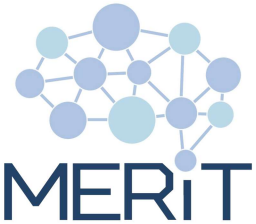
NewSQL represents an innovative approach that attempts to combine the best aspects of traditional relational databases with the scalability advantages of NoSQL systems. These databases maintain:

- Full SQL query support
- ACID transaction guarantees
- Horizontal scalability across distributed clusters
- Strong consistency models

Leading NewSQL implementations include CockroachDB and TiDB, both of which draw inspiration from Google's Spanner architecture. These systems use sophisticated consensus algorithms and distributed transaction protocols to maintain consistency across geographically distributed nodes.

The key innovation of NewSQL is its ability to parallelize SQL workloads across multiple machines while preserving the transactional semantics developers expect from traditional databases. This makes NewSQL particularly valuable for applications that require both scale and strong consistency, such as financial systems and e-commerce platforms.

Polyglot Persistence: Concept



Polyglot persistence represents a pragmatic approach to database selection that acknowledges no single database technology can optimally serve all data access patterns. This architectural pattern involves:

Strategic Database Selection

Choosing specific database technologies based on distinct data characteristics and access patterns rather than forcing all data into a single system.

Complementary Technologies

Combining relational databases for transactional integrity with NoSQL systems for flexibility and scale, creating a more resilient and performant overall system.

Domain-Driven Design

Aligning database choices with bounded contexts in the application domain, allowing each subsystem to use the most appropriate persistence mechanism.

By embracing polyglot persistence, organizations can leverage PostgreSQL's robust transaction handling alongside MongoDB's flexible document model, creating systems that are both reliable and adaptable to changing requirements.

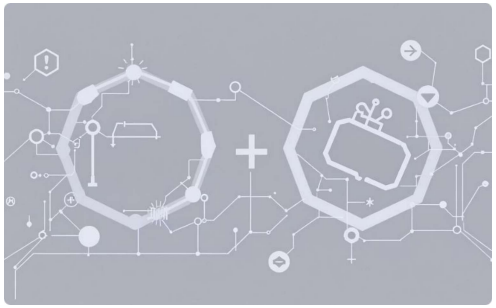
Implementation Examples

PostgreSQL + MongoDB

This combination leverages PostgreSQL's robust transactional capabilities for financial data while utilizing MongoDB's flexible schema for content that requires frequent structural changes. Many e-commerce platforms adopt this approach, storing order and payment information in PostgreSQL while keeping product catalogs and user profiles in MongoDB.

Cassandra + CockroachDB

This pairing addresses applications requiring both massive write throughput and transactional integrity. Cassandra handles high-volume sensor data or logs, while CockroachDB manages critical relational data that requires global consistency across regions.



Redis + TiDB

Redis provides ultra-fast caching and real-time features while TiDB offers MySQL-compatible distributed SQL capabilities. This combination is particularly effective for applications requiring both lightning-fast responses and scalable transaction processing.

Use Cases and Comparisons

Selecting the appropriate database technology requires careful analysis of your application's requirements. Each database paradigm excels in specific scenarios:

Relational Databases (SQL)

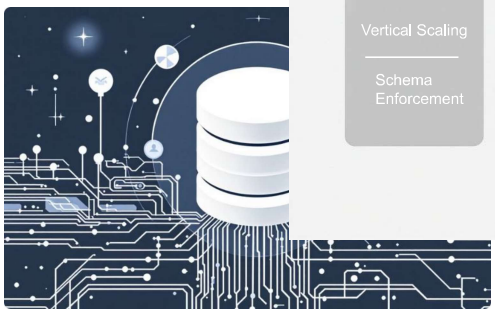
- Financial systems requiring strict ACID compliance
- Applications with complex reporting needs
- Systems with well-defined, stable data structures
- Scenarios requiring complex joins and transactions

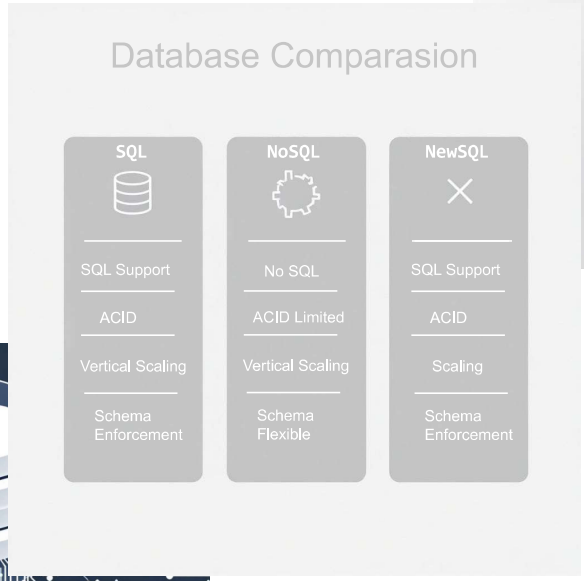
NoSQL Databases

- Applications with evolving or unpredictable schemas
- Systems requiring massive horizontal scaling
- Use cases with simple query patterns but high throughput
- Scenarios prioritizing availability over consistency

NewSQL Databases

- Applications requiring both SQL familiarity and horizontal scaling
- Global systems that need strong consistency across regions
- Modernization projects migrating from traditional RDBMS
- High-value transactional workloads that must scale elastically





The decision matrix should consider factors including data structure complexity, consistency requirements, scaling needs, and your team's technical expertise. Many modern architectures benefit from combining multiple database types to address different aspects of their data management challenges.

Conclusion and Next Steps

Throughout this presentation, we've explored the fundamentals of NoSQL and NewSQL database technologies and introduced the concept of polyglot persistence. These approaches represent complementary tools in the modern data architect's toolkit, each addressing specific challenges in today's complex data landscape.

Practical Experimentation

Set up local development environments using Docker to experiment with PostgreSQL and MongoDB side by side, exploring their strengths and limitations firsthand.

Benchmark Testing

Conduct performance testing with realistic workloads to understand how different database technologies respond under various conditions and load patterns.

Integration Patterns

Explore design patterns for effectively combining multiple database technologies within a single application architecture, focusing on data consistency and synchronization challenges.

Our next session will include hands-on lab exercises where you'll deploy both PostgreSQL and MongoDB containers, create schemas, insert data, and compare query capabilities across these fundamentally different database paradigms.