
## Why NoSQL (MongoDB) Over SQL (RDBMS) for Core Application Functionality

Expansive Object Model: MongoDB allows for objects to have base properties, and for nesting of object properties, with the capacity to predicate indexes about nesting and object properties.

Flexible Object Property Indexing: MongoDB allows for indexing any property of any object at any level of the hierarchy; provides significant performance improvements on data reads (further improved with GraphQL design).

Availability: Single master node model, with subordinate node backups. MongoDB transition period for subordinate to elevate to master: approximately 10 seconds. As consistent as AWS with RDBMS.

Write Scalability: Only restricted by number of servers leveraged in the cluster; allows greater flexibility with how scaling can be determined (i.e. concurrent users, budget, hours of operation, etcetera).

Query Language Support: Structured as JSON fragments, supports SQL, does not support complex joins (remedied with GraphQL design). MongoDB supports dynamic querying with instantiated views.

Performant Benchmark: Scalability can be efficiently and cheaply leveraged both horizontally (more server computation) *and* vertically (more RAM). Demonstrably lower latency with higher throughput with NoSQL over SQL. MongoDB also uses internal memory for storing working information, providing quicker access and load times. NoSQL is more optimized for computation and querying, whereas SQL is more optimized for storage.

Ease of Use: No separate programming language (i.e. SQL) required to learn for writing a MongoDB based application, code for interaction is written in native application programming language via any of the provided drivers. Less challenging to set up, configure, run and support in comparison to RDBMS.

Schema: MongoDB allows schema-less, dynamic data modeling, which affords support of fluid data dynamics. Instead of being restricted to table-based data structures only, applications using NoSQL formats may also leverage Key-Values, Wide-Column Containers, Document Collections and Graph Structuring.

Aggregation: MongoDB provides a native aggregation framework for Extract, Transform, Load (ETL) pipelines for data transformations.

Philosophy: MongoDB premised on CAP (Consistency, Availability and Partition) versus RDBMS premised on ACID (Atomicity, Consistency, Isolation and Durability). Important priority for application functionality as current business and technological concerns align on Availability.

Reporting: Proper data design, with a GraphQL implementation, allows for performant OLAP (Online Analytical Processing) and OLTP (Online Transaction Processing) operations.

## Summary

The NoSQL decision for application core functionality is predicated upon NoSQL being projected to be most performant in anticipated application benchmarks and cheaper to scale with an improved user experience.

## Situations Recommended to Retain Entity Data in SQL Formats

- Billing Engine driven data (reasoning: permanency of structure and byte usage management).
- Entity data which any *analytical marketing* is predicated upon (reasoning: business consideration).
- Entity data that is definitively scoped for the entirety of a product's lifecycle (reasoning: permanency of structure).

- Scenarios where computational power needs dictate the business case; such as algorithm processing, or CRON jobs for computations (reasoning: cheaper to scale *only* vertically with SQL).
- No anticipation of application changes, growth or feature development (reasoning: permanency of structure).
- ACID compliance, in lieu of CAP theorem, is a business requirement for data structures (reasoning: business consideration).