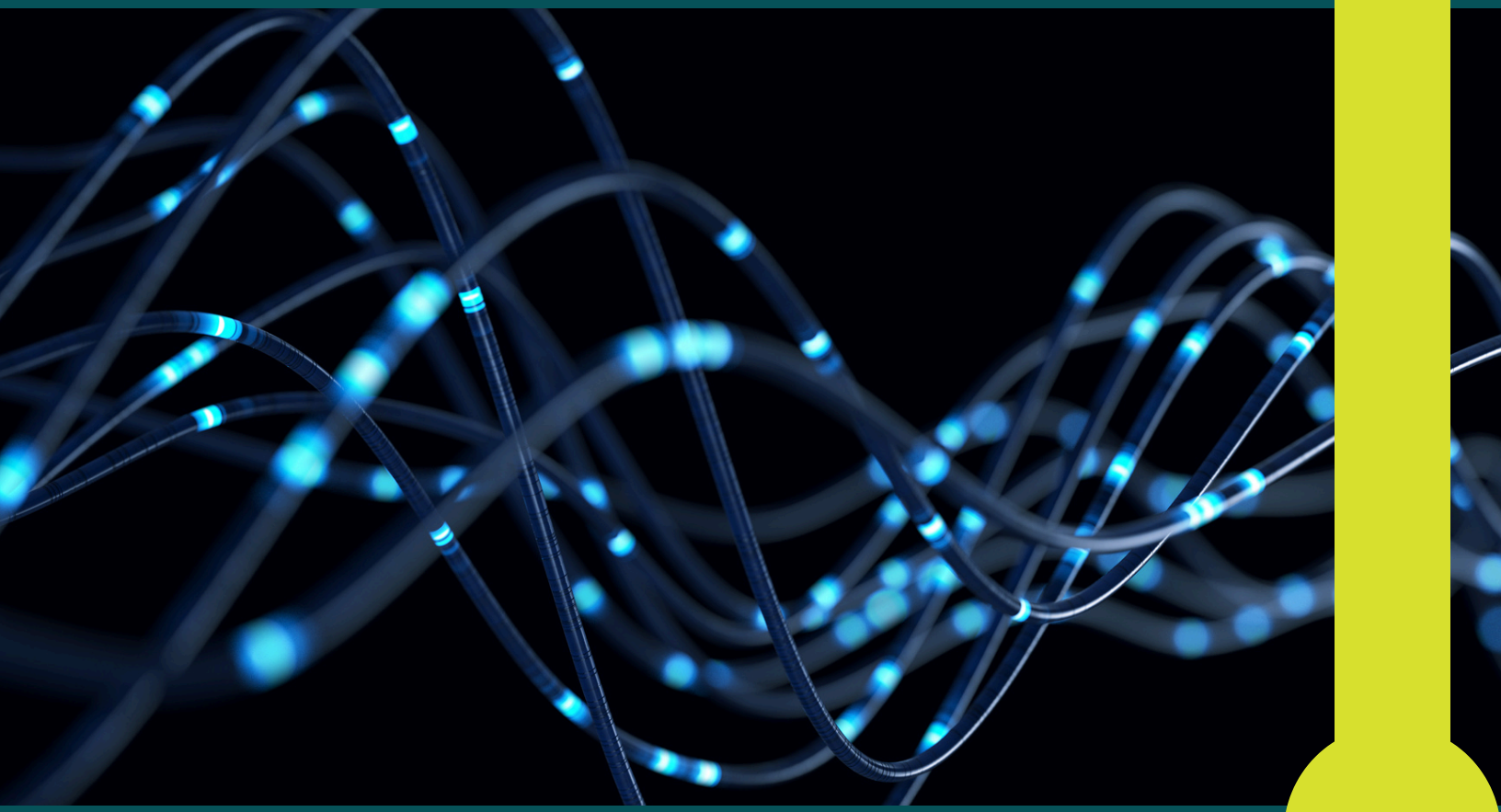
stelo



Understanding NoSQL

AND WHAT IT MEANS FOR DATA STREAMING

WHITE PAPER

Understanding NoSQL and What it Means for Data Streaming

SQL and NoSQL differ in a number of critical ways (e.g., relational vs. non-relational, schema, scalability, type of data). However, each serves a specific, worthwhile function. Here, we'll cover NoSQL database basics and how Stelo supports data mirroring from relational databases to NoSQL environments.

What is a NoSQL Database?

The term "NoSQL" or "non-SQL," short for "not only SQL database," mainly applies to databases for large-scale web and cloud applications. While many were established in the early 2000s, NoSQL database technology is still evolving today.

As a quick refresher, SQL, or Structured Query Language, is a well-known language for accessing databases and creating commands to manipulate them. Understanding what makes NoSQL unique starts with what it's *not* rather than what it *is*. NoSQL databases store information in a format other than relational tables.

The NoSQL approach to database management rose in popularity alongside a growing variety of data models. At the time, it became more important to support clustering and replication, also known as mirroring, for more models including key-value, document, columnar, graph, etc.

---

*Understanding what makes NoSQL unique starts with what it's **not** rather than what it **is**.*

In other words, NoSQL databases have lots of different models for storing the same kinds of information. They're designed such that data can be accessed even if some information isn't available or needed. NoSQL databases are sometimes referred to as "elastic" storage repositories because you can add capacity without impacting the rest of the database. They're loosely coupled and distributed.

Finally, many NoSQL database systems are open source. Typically, vendors mix and match elements from a variety of NoSQL database types to support different business needs.

Functional Differences Between NoSQL and SQL Databases

Beyond their most obvious difference, whether they're relational or non-relational, SQL and NoSQL differ in a number of ways. Here, we'll discuss their differences in terms of structure, transaction processing requirements, scalability, and support.

Data Models / Formats

From a structural perspective, NoSQL databases don't rely on rows and tables alone. They can leverage many different formats. SQL databases are more tabular in nature, so data structuring is a requirement.

Successful Transaction Indicators

Relational database management systems (RDBMS) that leverage SQL need to exhibit ACID to demonstrate that transactions are processed successfully. "ACID" stands for atomicity, consistency, isolation, and durability. Ensuring these qualities is said to ensure the validity of each transaction.

- **Atomicity:** Each transaction is treated as a single unit that succeeds or fails; if any statement fails, the entire transaction fails. This quality prevents partial data updates.
- **Consistency:** Data moved to a database must be valid according to all defined rules such that the database remains consistent.
- **Isolation:** Concurrent transactions should leave the database in the same state as they would if the transactions were performed in sequential order to minimize or remove the effects of an incomplete transaction.
- **Durability:** Once a transaction has been committed, it will remain that way, even if the system fails.

Most NoSQL databases are built around standards, like SQL-92, that enable reliable transactions and ACID properties. Relational databases need to process transactions in a sequential fashion to achieve ACID properties, but that's avoidable with NoSQL because it's distributed and scalable. NoSQL integrates the CAP theorem, also known as Brewer's Theorem, alongside relevant guidelines from ACID. "CAP" stands for consistency, availability, and partition tolerance:

- **Consistency:** All clients see the same data at the same time, no matter which node they connect to. This is accomplished via mirroring.
- **Availability:** Working nodes always return a valid response, so every client making a request for data will hear back, even if nodes are down.
- **Partition Tolerance:** A cluster continues to work regardless of communications breakdowns between nodes, otherwise known as partitions.

Scalability

Scalability is key when it comes to futureproofing data management strategy because it accommodates changes in data type and volume. NoSQL databases scale better horizontally. In other words, additional servers or nodes make it possible to increase the load. SQL databases scale better vertically. Load increases are accommodated by migrating data to a larger server or adding CPU, RAM, etc.

Scalability is key when it comes to futureproofing data management strategy because it accommodates changes in data type and volume.

Support

NoSQL has seen a quick rise in adoption, but it's a smaller user community relative to SQL. Many SQL languages are proprietary, but they have the benefit of longevity. There are many experts and examples readily available for those looking for information. Those using NoSQL benefit from open-source systems, but they don't have the same user history to draw on.

NoSQL Background

We can't talk about NoSQL without the contextual history of SQL. SQL was popularized in the 1970s because it became the default language for RDBMS. At the time, data storage was more expensive, so reducing data duplication was an attractive quality. SQL is still widely used for relational databases where data is stored in linked rows and tables. This design lends itself to being simple and reliable for planned, complex ad hoc SQL queries.

However, some SQL and relational database requirements (e.g., rigid schema) make them less suitable for applications where flexibility and speed are paramount concerns (e.g., websites, data distribution). NoSQL databases emerged to address these needs.

Many popular NoSQL databases have been developed by companies (e.g., Amazon, Google), but there are examples of systems, like BerkeleyDB, that were developed in the academy before being sold and commercially released.

Originally, "NoSQL" was a more literal term; SQL wasn't used as the application programming interface (API) to access data. However, SQL has proven so useful that many NoSQL databases have added support for it. When NoSQL and SQL elements are mixed together, the combination is commonly referred to as a multimodel database.

Once a literal term, NoSQL now often includes SQL, resulting in what's called a multimodel database.

Common Types of NoSQL Databases

The most common types of NoSQL database systems are:

Document Databases

With semi-structured data and a document format, document databases are useful for content management and data management for mobile applications (e.g., blog platforms, web analytics, and e-commerce). They allow developers to create and update their programs without referencing any sort of master schema. The popularity of this type of NoSQL database grew alongside JavaScript and JSON, which are both favored by application developers.

Examples include: CouchDB, MongoDB

Graph Databases

In graph databases, data is organized as nodes and edges to better represent how data relates. Nodes are similar to rows in relational databases, and edges represent the connections between nodes. Unlike relational models that rely on strict schema, graph databases can evolve over time. These databases are useful for social media platforms and reservation systems.

Examples include: IBM Db2 Graph, AllegroGraph, Neo4j

Key-Value Databases

Known for simplicity and scalability, key-value databases pair a unique key with an associated value. This action is useful for session management (e.g., shopping cart details, multiplayer gaming).

Examples include: Aerospike, Amazon DynamoDB, Redis

Wide-Column Databases

Like relational database tables, wide-column databases use tables, columns, and rows; however, naming and formatting can change between rows in a single table. This is possible because each column is stored separately on a disk, which is optimal when querying data by columns rather than rows. This type of database is useful for recommendation engines and catalogs.

Examples include: Amazon Simple DB, Apache Cassandra, Hypertable

How Stelo Supports Data Streaming

Historically, it was easier to move data from one relational database to another because the task only required managing different formats of SQL. Open database connectivity (ODBC) was designed as an interface for relational databases that follow an SQL model with ACID properties and individual, row-level operations. While that's useful for data mirroring functions, it's not always appropriate for NoSQL environments where it's critical to be able to ingest large amounts of data quickly and reliably, but not necessarily by row or transaction.

With the advent of NoSQL, data still needs to move around, but the task requires different delivery methods.

With the advent of NoSQL, data still needs to move around, but the task requires different delivery methods. More and more, data is structured in relational databases, and it needs to be transformed for high performance streaming.

Stelo manages both data mirroring and data streaming with technology that's designed to complement your current and future data management infrastructure. Most organizations still rely on relational databases to power business operations and you need a mirroring tool like Stelo Data Replicator that can aggregate operational data into your NoSQL environment of choice to take advantage of state-of-the-art analytics.

To make this process quick and easy, our approach involves decoupling the delivery of data from the relational framework to make it more self-defining. That way, we're able to stream source data or mirror it into a NoSQL environment.

Stelo manages both data mirroring and data streaming with technology that's designed to complement your current and future data management infrastructure.

Stelo also accommodates fine-tuned data streaming management. Depending on the demands facing your business, you can schedule data streaming in batches to minimize compute resource utilization, and by extension cost, or stream continuously to accommodate more data at a faster rate.

UNDERSTANDING NOSQL AND WHAT IT MEANS FOR DATA STREAMING

With Stelo, data streaming doesn't require additional components. Stelo connects to NoSQL environments such as Databricks, Google BigQuery, and Confluent, with the same hassle-free, no-coding approach as our ODBC support. We built both streaming and mirroring capabilities directly into our data replicator because we believe design for scalability is one of the best ways to futureproof data management systems.

For more information on Stelo Data Replicator and NoSQL mirroring and streaming, visit our data streaming solutions page or contact us directly.