The Official Ten-Year Retrospective of NEWSQL
NEWSQL

adjective  
\ˈnü-sē-kwəl\ 

A category of relational DBMSs designed to support scalable workloads for operational applications.
A category of relational DBMSs designed to support scalable workloads for operational applications.
A category of relational DBMSs designed to support scalable workloads for operational applications.
A category of relational DBMSs designed to support scalable workloads for operational applications.
A category of relational DBMSs designed to support scalable workloads for operational applications.
Outline

How did the **NewSQL** trend start?

Is the **NewSQL** term still relevant?

What is the future of OLTP DBMSs?
Twenty-First Century of Database Systems
Early 2000s – The Internet Boom

It was now possible for a small organization to build an application that could be used by many concurrent users.

Database scalability challenges were no longer limited to major corporations.
Early 2000s – Legacy Systems

Single-node deployments using old "elephant" DBMSs. Viable open-source options did not exist.

Can only scale vertically. Expensive software + hardware.
Mid 2000s – Sharding Middleware

Combine multiple nodes into a logical database.
Route / rewrite queries to access data at specific nodes.

Development cost.
Limited functionality.
Late 2000s – NoSQL

Forgo DBMS-enforced protections to achieve high-availability and high-scalability. Non-relational data models without schemas. No transaction guarantees. Custom query APIs.
Late 2000s – NoSQL

Forgo DBMS-enforced primary key, high-availability and high-scale.

Non-relational data models.

No transaction guarantees.

Custom query APIs.
Late 2000s – NoSQL

Forgo DBMS-enforced protections to achieve high-availability and high-scalability.

Non-relational data models without schemas.

No transaction guarantees.

Custom query APIs.

*Application must handle eventually consistent data, lack of transactions, and joins.*
Forgo DBMS-enforced protect high-availability and high-scale.
Non-relational data models with No transaction guarantees.

Custom query APIs.
Application must handle even data, lack of transactions, and
Late 2000s – NoSQL

Forgo DBMS-enforced protection of high-availability and high-scalability

No transaction guarantees

Custom query APIs

Application must handle data, lack of transactions, and

Spam: Google's Globally-Distributed Database


Google, Inc.

Abstract

Spanning is Google's globally-distributed database for storing large quantities of data. As such, it can support transactions. We believe it was in part built to address this failing. Some authors have claimed that general two-phase commit is too expensive to support, because of the performance or availability problems that it brings [9, 10, 19]. We believe it is better to have application programmers deal with performance problems due to overuse of transactions as bottlenecks arise, rather than always coding around the lack of transactions. Running two-phase commit over Paxos mitigates the availability problems.

The application data model is layered on top of the directory-bucketed key-value mappings supported by the
The Rise of NewSQL Systems
Aslett Report (2011)

[Systems that] deliver the scalability and flexibility promised by NoSQL while retaining the support for SQL queries and/or ACID, or to improve performance for appropriate workloads.

Matt Aslett – 451 Group (April 4th, 2011)
https://www.451research.com/report-short?entityId=66963
Shared-nothing architecture.

High-performance, non-locking concurrency control.

ACID support for transactions.

SQL as the primary interface.
A class of modern relational DBMSs that provide the same scalable performance of NoSQL systems for OLTP workloads while still maintaining the ACID guarantees of a traditional DBMS.
Tradional SQL vs. NoSQL

- **Scalability**
  - **High** (Many Nodes)
  - **Low** (One Node)

- **Guarantees**
  - **Weak** (None/Limited)
  - **Strong** (ACID)

**NoSQL**

**NewSQL**

**Traditional**
Refrinement of NewSQL categories and properties.

New Architectures
Middleware
Database-as-a-Service

SIGMOD Rec Article (2016)

Andy Pavlo, Matt Aslett – SIGMOD Record (June 2016)
https://dl.acm.org/doi/10.1145/3003665.3003674
New Architectures

New codebase written from scratch without architectural baggage of legacy systems.

Almost all DBMSs use a shared-nothing architecture.

Our mistake was to not include shared-disk DBMSs.
New Architectures

New codebase written from scratch without architectural baggage of legacy systems.

Almost all DBMSs use a shared-nothing architecture.

Our mistake was to not include shared-disk DBMSs.
Middleware

Transparent data sharding and query redirecting over cluster of single-node DBMSs.

Usually support MySQL or PostgreSQL wire protocol.
Middleware

Transparent data sharding and query redirecting over cluster of single-node DBMSs.

Usually support MySQL or PostgreSQL wire protocol.
Database-as-a-Service

Distributed architecture designed specifically for cloud-native deployment.

Most of them use MySQL for single-node storage.
Why Not MySQL Engines?

The acquisition of MySQL AB by Sun Microsystems in January 2008 appeared to signal that open source databases were on the brink of opening up a new battleground against the proprietary database giants. In announcing the deal, Sun hoped to reinvigorate MySQL to challenge the established vendor, and development resources required for MySQL to challenge the established vendors in supporting mission-critical, high-performance applications on Web-based software. Needless to say, reality was somewhat different as Sun faced wider problems of its own and eventually succeeded to take over by Oracle (Nabbing ORCL) in April 2010, in doing so taking ownership of the leading commercial open source database to the database heavyweight.

We had previously argued that MySQL was very much the crown jewel of the open source database world thanks to its focus on Web applications, its lightweight architecture, and its ease of use. However, the reality was that MySQL was not a viable alternative to Oracle or other enterprise database vendors. It was not the right tool for the job, and even in the early days of Facebook and Twitter, where MySQL was originally designed, it was not the right choice.

In recent years, the database market has shifted away from open source databases with lightweight architectures targeted at Web applications. Not only have the likes of MariaDB and PostgreSQL emerged to provide alternative, open-source solutions for MySQL, but there are also a growing number of products available under the umbrella of NoSQL, which offers a more flexible and scalable approach to handling data. As a result, MySQL’s market share has declined, and the company has faced increasing competition from other database vendors.

In conclusion, while MySQL was once a beacon for open source advocates and a shining example of the power of collaboration and community, it is clear that the company and the technology it represented have fallen behind the times. As the database landscape continues to evolve, it will be interesting to see what the future holds for MySQL and its legacy.
Why Not MySQL Engines?

InnoDB is an excellent OLTP engine for single-node MySQL instances.

Nobody has built a long-term successful business replacing it.
What Went WRONG?
What Went Wrong?

Almost every NewSQL company from the last decade has closed, sold for scraps, or pivoted to other markets.
Selling an OLTP DBMS is Hard

Existing Application:
- People are risk adverse in replacing an OLTP DBMS even if it is slow or expensive.

Greenfield Application:
- The engineering start-up "cost" of a relational DBMS is higher than shoving JSON into a NoSQL DBMS.
Existing DBMSs Are Really Good

The two most popular open-source DBMSs got even better in the last decade.

Most new applications don't have any data, so a single node DBMS is good enough.
Cloud Disruption

Most NewSQL companies started with selling on-prem and missed shift to cloud.

Difficult to compete with major cloud vendors on cost and technology.
Lack of Open-Source

Few of the NewSQL DBMSs were open-source.

This may have inhibited their adoption, especially with developers building new applications.
The Aftermath
Distributed SQL

New vendors are promising the same benefits of earlier NewSQL systems and seeing better adoption.

Many of their core concepts are similar to earlier systems.
"Not Only SQL"

The vanguard NoSQL systems that touted their lack of SQL, joins, and transactions now include these features.
Remnants of NewSQL

As of 2021, the term NewSQL seems to be only used by Chinese start-up database companies.
Remnants of NewSQL

As of 2021, the term NewSQL seems to be only used by

RadonDB is a new generation of distributed relational database based on MySQL, we call it MyNewSQL. It was designed to create a database that capable to satisfy the requirement of large-scale transaction workload with high availability and reliability. RadonDB is architected to two independent cluster layers: SQL Layer and Transaction Layer, and the following guide show the detail of the inner-workings:
What Comes NEXT?
Random Prognostications
an evening with Andy Pavlo
MySQL

MariaDB

• Fracturing will hurt perception & adoption.
  – See CouchDB vs. Couchbase.
  – This may be intentional.
**MySQL**

- Fracturing will hurt post adoption.
  - See CouchDB vs. Couch.
  - This may be intentional.

**mongoDB**

- Will become first choice in new start-ups & Web apps.
  - Flush with cash, with more on the way.
  - MySQL-like growing pains.
The Next 10 Years

It will be difficult to supplant existing OLTP DBMSs unless there is another major hardware transition.

It will also be difficult to upend major cloud vendors on pricing unless...
The Next 10 Years

You still need humans to design, configure, and optimize logical/physical aspects of a database.

Humans are expensive.

Automation is the future.
Conclusion

NewSQL is dead.

From an academic view, the NewSQL movement was a success.

From a business view, it was a failure for those that embraced the NewSQL mantle.
END
@ANDY_PAVLO