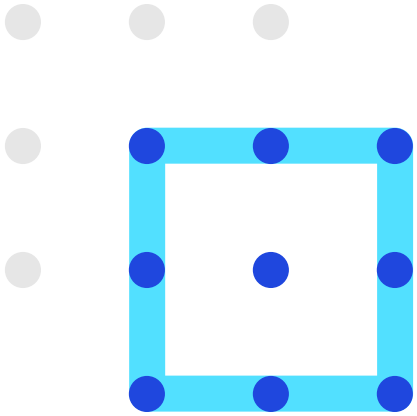
WHITEPAPER

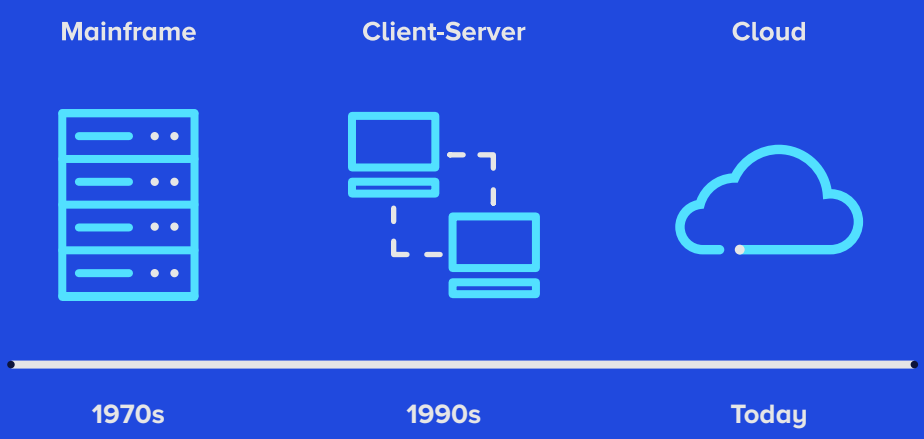


Modernizing with NoSQL from Relational Data Platforms

The Keys to Success in Designing Your Applications for NoSQL

DATASTAX

When internet pioneers like Google, Facebook, Yahoo, and Amazon began designing the infrastructure they needed to run their global businesses, they hit a major obstacle in their data layer. Legacy database software did not support the requirements of the (then) newly evolved applications that had markedly different requirements than the previous application eras of centralized and semi-centralized systems.



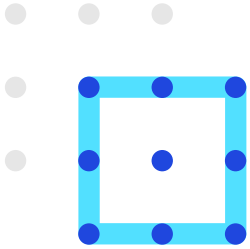
Because the new radically distributed and massive scale of multi-home internet applications could not be serviced with traditional databases, the innovators did what innovators typically do—they built their own. These databases are today known as NoSQL platforms.

Today, every business—no matter the size—that wants to successfully compete in the digital marketplace has to satisfy today's increasing data demands and smartly decide between using a traditional relational database management system (RDBMS) or NoSQL software for each application. Further, they need to know when and how to migrate from traditional databases to modern databases when the underlying application's requirements call for it.

How do you determine when to modernize your systems from legacy database software to data platforms built from the ground up to support digital, mobile, and cloud applications that run everywhere? Even though most IT analysts and investors say that NoSQL will dominate new software stack spending for years to come, a polyglot environment will be the norm for some time, so a coexistence mindset and strategy is necessary.

01

Application Litmus Tests for Database Modernization



Below are key litmus tests that you can use to ascertain whether to use legacy RDBMS or NoSQL for new applications and when to migrate existing legacy systems to NoSQL.

Location Dependent or Independent?

The first and most obvious criteria for deciding between relational and NoSQL involves the "reach" of the application. Will it serve just a centralized location and set of users or is it a widely distributed system that caters to users everywhere? The former is location *dependent* while the latter is location *independent*.

Location independence means that your data can live anywhere and be *read* and written everywhere as well. It provides for uniform customer response times because your data is where your customers are, which means they can access their data just as fast from any location. This is especially important since many new application architectures are multi-homed, including most retail mobile applications and those that are IoT in nature.

If your application needs location independence, then a NoSQL platform that sports a distributed architecture will be the right choice.

High or Continuous Availability?

Some business applications can tolerate unscheduled downtime, but those that operate the business are increasingly expected to stay online 100% of the time. And that is no easy task for an application's data layer.

Studies done by the Uptime Institute humble even the overly confident IT professional who thinks their systems won't go down—especially those who trust the cloud to save them. Even with all the advances in technology, Uptime's studies show that ***outages are actually increasing*** and cloud providers are now the ***second*** most commonly cited reason for IT service failure (No. 1 is still on-premises data center issues).

While relational technology can provide *high* availability, it cannot—because of its underlying architecture—promise *continuous* availability. If an application needs the latter, it needs a NoSQL platform with a masterless architecture.

Moderate or High Data / User Activity?

Almost every application has a unique data access pattern. Some systems have a trickle of data coming in and going out, others have little entering the system but a lot of reads on that data, while still others might have enormous amounts of data coming in from countless locations (e.g. IoT apps for sensors with time-series data) and similar I/O counts on data reads.

Relational databases are adequate for small to moderate data interactions, but very high write/read activity almost always necessitate a NoSQL solution that is not chained to a legacy master/slave architecture.

Moderate or High Data Volumes?

Not to be confused with the activity criteria, the data volume question concerns itself with two things: data scale and uniform performance.

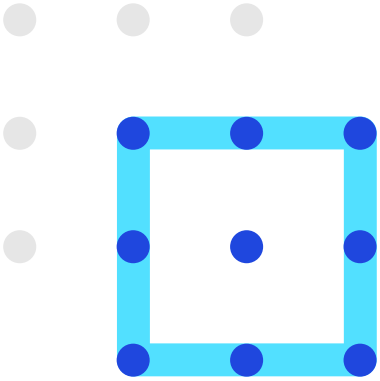
While good at handling moderate data volumes, the limitations of operational relational databases in the area of data scale are well documented. In fact, many companies still make their living by archiving RDBMS data into separate databases and linking them together to avoid performance slowdowns caused by too much data.

Today, many digital applications have requirements to keep nearly all collected data online with continuous availability for end-user interactions, which is a deathblow for uniform performance SLAs if relational databases are used. To handle high data scale and deliver uniform response times regardless of data volumes, a scale-out NoSQL platform is needed that scales for *both* writes and reads.

Fixed or Flexible Schema?

Microservices architectures are the *soup du jour* today where developing modern applications are concerned.

Microservices greatly benefit from distributed NoSQL databases that can provide a per-service isolation of resources and multi-region data replication. Microservices stitch together components that roll up to the big picture application in a way that allows development teams to more easily work in parallel and thus get things to market more quickly. Key to the success of microservices is strong flexibility at the data layer, which accommodates the concept of polyglot persistence with respect to database schemas in a more modern way than previous implementations.



Relational databases handle fixed, traditional schemas for applications very well. However, with skin-only support for data models outside the relational schema, they can struggle to support modern microservices applications that require schema flexibility, which are offered by some NoSQL providers.

A flexible, multi-model schema approach represents the next phase of maturity for the database industry and supports multiple data models against a single, integrated backend, that:

- Supports multiple data models (e.g., tabular, JSON, graph) at a logical layer for ease of development for application developers
- Ensures all models are exposed via cohesive mechanisms thereby avoiding cognitive context switching for developers
- Provides a unified persistence layer that delivers geo-redundant, always-on characteristics and a common framework for operational aspects such as security, provisioning, disaster recovery, etc.
- Empowers a variety of use cases across OLTP and OLAP workloads for lines of businesses within an enterprise to innovate with agility
- Delivers best in class TCO efficiency for the long haul to enable wider adoption within centralized IT teams of an organization

Traditional or Contextual Transactions?

Many digital applications and their database interactions have evolved past the traditional transactions offered in legacy relational engines.

A typical credit card transaction is a good example. The authorization process used by the credit card vendor contains many different contexts in order to avoid a fraudulent event. Not only does it contain the standard database transactional characteristics, but it also involves search and/or graph operations that review historical purchase activity, followed by analytics that are run on that data, which ensures it's in line with the current purchase, and then it goes through the approval process and returns its response back to the application and user. One transaction, done in split-second fashion, so the user doesn't grow impatient and move on to something else.

There are various industry terms used for contextual transactions—hybrid transactional analytical processing (HTAP), translytical processing, and so on. Whichever name is used, the idea is that your typical ACID transactions of legacy databases are long gone.

Instead, a data architecture that forgoes the typical separation of database workloads is required today, which is what some NoSQL platforms offer.

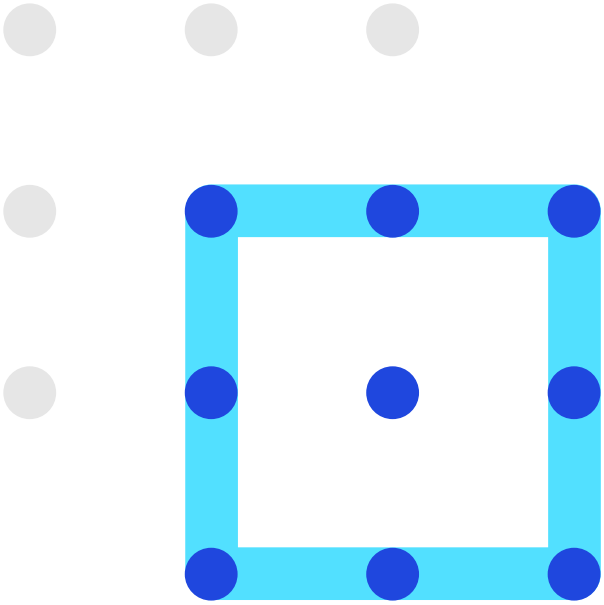
Cloud Compatible or Native?

All RDBMS vendors offer a version of their database in the cloud, however, that doesn't make them a cloud database. There is no cloud pixie dust that automagically takes legacy relational databases or certain NoSQL databases and transforms them into a native cloud database that exploits all the benefits that cloud offers.

Modern data architectures will almost certainly use cloud in some way. In their "The Future of the DBMS Market Is Cloud" report, Gartner states, "Database management system deployments and innovations are increasingly cloud-first or cloud-only. Data and analytics leaders selecting DBMS solutions must accept that cloud DBMS is the future and must plan for shifting spending, staffing and development accordingly."

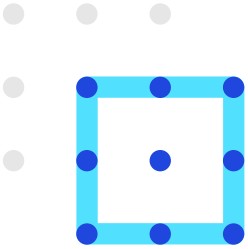
So how does a native cloud database behave versus those that simply are compatible with their on-premises counterparts in the cloud? A short list of the characteristics include:

- **Transparent elasticity** – being able to easily expand and contract resources given usage and resource demands.
- **Unbounded scalability** – one TB or PB, one thousand users or ten million, the cloud database elasticity should include seamless scalability (and make no mistake, the two are not synonymous).
- **Built-in redundancy** – the database should be able to fully exploit different regions, availability zones, and (yes!) different clouds to guarantee zero downtime and no loss of data access.
- **Simplified data distribution** – same as location independence described above.
- **Autonomous manageability** – the typical administrator tasks of backup, provisioning, tuning, etc., should be handled in a hands-free way.
- **Uniform security** – data protection should be applied in a consistent manner across workloads, data models, other clouds, and on-premises participants in the deployments.



02

Transition and Migration Keys to Success



One of the great things about some NoSQL platforms is that they are very forgiving where provisioning, deployment, and configuration mistakes are concerned. However, there are two essential keys to success that must be followed if you are to be successful with transitioning or migrating from relational to NoSQL.

Key Number One - Rethinking the Data Model

If you want to almost certainly fail at a NoSQL deployment, design your data model as you would in an RDBMS.

While the underlying schema of Apache Cassandra™ looks nearly identical to relational tables, the input for them is not identifying entities, relationships, and attributes, but instead understanding your data access patterns and queries that will need to be satisfied in the supported application. This may seem like a subtle difference, but it's a big factor in whether your database will perform and scale as you hope.

Fortunately, making this mindset shift is not hard and there are plenty of free online courses at DataStax Academy that take you step-by-step through the data model design process.

Key Number Two - Smart Hardware Design

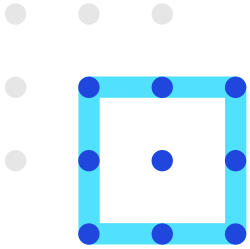
A better approach is to smartly scale out where neither compute nor storage bottlenecks exist.

There are two ways to doom a NoSQL deployment from a hardware standpoint. First, you can be penny-wise and pound-foolish and use the least capable / fewest sets of hardware. Unless you have a small and static database, you will hit a scale and performance wall that cannot be overcome.

The second route to failure involves using a scale-up approach and few pieces of very large hardware. Unless your application is compute/CPU-bound, then you will almost certainly become a victim of storage congestion and contention. A better approach is to smartly scale out where neither compute nor storage bottlenecks exist.

03

Relational or NoSQL Requirements Summary

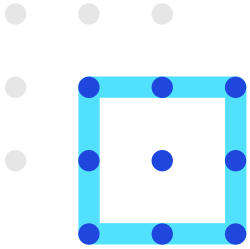


Below is an at-a-glance summary that can be used to help decide between RDBMS and NoSQL technology:

Use an RDBMS when you need	Use NoSQL when you need
Location dependent application	Location independent application
High application availability	Continuous application availability
Low to moderate data/user activity	Low to high data/user activity
Low to moderate online data volumes	Low to very high online data volumes
Fixed-only database schema	Flexible database schema
Traditional-only transactions	Contextual transactions
Cloud compatible database functionality	Cloud native database functionality

04

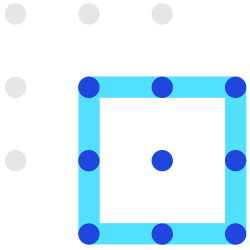
How DataStax Compares



DataStax data management platforms make it easy to build, deploy, and scale applications that exploit the full value of ever-growing data—wherever and however they are deployed. Built on the No. 1 open source database noted for scale, constant uptime, and elegant data distribution—Apache Cassandra—DataStax solutions provide best-in-class support for the requirements that signal the need for a modern / NoSQL data platform:

Requirement	Support?	How?
Location independence	Yes	Masterless architecture and gold standard in replication support make putting data anywhere and synchronizing it everywhere easy
Continuous availability	Yes	Masterless architecture provides complete redundancy in data and compute resources
High data/user activity	Yes	Linear scale and uniform response times via scale-out deployment
High online data volumes	Yes	Linear scale capability via scale-out deployment
Flexible schema	Yes	Full multi-model support for tabular, documents, key-value, and graph
Contextual transactions	Yes	Integrated analytics, search, and in-memory engines for full contextual support
Cloud native	Yes	Foundation built and proven by internet pioneers for cloud native experience

For More Information



Learn more about how DataStax handles today's modern application requirements by visiting our website and resources page.

DataStax Enterprise (DSE) was natively built to deploy modern applications in hybrid cloud environments and to consume time series and sensor-based information faster than any other database. Based on Cassandra, DSE provides a contextual, always-on, real-time data management platform which can grow to unlimited scale. The platform is complemented by DSE Search, which provides real-time indexing, DSE Analytics, which delivers streaming and batch processing, and DSE Graph, which enables users to derive powerful insights from graph data.

Learn More

datastax.com/resources

DATASTAX