DZone
A DEVADA MEDIA PROPERTY

THE 2018 DZONE GUIDE TO

Databases

RELATIONAL AND BEYOND

VOLUME V

BROUGHT TO YOU IN PARTNERSHIP WITH

Datical DELPHIX influxdata MariaDB redgate

Dear Reader,

Welcome to the *2018 DZone Guide to Databases: Relational and Beyond*! This year, we have seen AI further establish itself within the technology sector, powered by data science as its beating heart which, in turn, is enabled by an increasing capability to capture more data and analyze it faster than ever before.

With data at the forefront, the database space is evolving rapidly to accommodate the ever-growing data requirements of our age. At DZone.com, our Database Zone has also continued its growth with an 86% increase in year-over-year pageviews, with articles covering a spectrum of interests from relational databases to NoSQL, and the blended options that are increasingly available.

Reflecting the importance of databases within the era of modern computing, our Database Zone content regularly converges with many of the other topics we cover. This guide is no exception, as we examine the interface between databases and DevOps — the ability to deploy modern data technologies into production and reap the value from a myriad of optimizations available for scale and speed, as the data world continues its inexorable migration into the cloud.

Also, on the theme of convergence, we take a look at polyglot persistence, examining a practical approach to the hybrid model of allowing applications to communicate with a range of different database models in order to access data stored in the most appropriate formats.

As the uptake of big data continues to grow, so does the expectation that database platforms will provide the necessary capabilities and address a broad range of use cases and workloads. We cover this topic in an article about the complex data landscape.

In 2018, we have seen increasing recognition of the pitfalls of poor data practices, such as the Cambridge Analytica scandal, data breaches, and allegations of mass surveillance enabled by the collection and misuse of big data. Data professionals are the guardians of critical data assets, with responsibility for keeping that data securely stored, managed, and maintained. As their remit expands across a wider range of database management systems, new data types and emerging applications, what do DBAs and developers need from a database? An article in this guide looks beyond traditional requirements to those of the next generation.

Finally, turning to the next generation, we look at two database types that have shown considerable growth of late, in articles covering the use cases for time series databases and how to build enterprise performance into a graph database.

By Jo Stichbury
ZONE LEADER, DZONE

Table of Contents

DZone is...

BUSINESS & PRODUCT
MATT TORMOLLEN — CEO
MATT SCHMIDT — PRESIDENT
JESSE DAVIS — EVP, TECHNOLOGY
KELLET ATKINSON — MEDIA PRODUCT MANAGER

PRODUCTION
CHRIS SMITH — DIRECTOR OF PRODUCTION
ASHLEY SLATE — DESIGN DIRECTOR
ANDRE POWELL — SR. PRODUCTION COORD.
G. RYAN SPAIN — PRODUCTION COORD.
BILLY DAVIS — PRODUCTION COORD.
NAOMI KROMER — SR. CAMPAIGN SPECIALIST
JASON BUDDAY — CAMPAIGN SPECIALIST
MICHAELA LICARI — CAMPAIGN SPECIALIST

EDITORIAL
CAITLIN CANDELMO — DIR. OF CONTENT & COMMUNITY
MATT WERNER — PUBLICATIONS COORDINATOR
SARAH DAVIS — PUBLICATIONS ASSOCIATE
MICHAEL THARRINGTON — CONTENT & COMMUNITY MANAGER II
KARA PHELPS — CONTENT & COMMUNITY MANAGER
TOM SMITH — RESEARCH ANALYST
MIKE GATES — CONTENT TEAM LEAD
JORDAN BAKER — CONTENT COORDINATOR
ANNE MARIE GLEN — CONTENT COORDINATOR
ANDRE LEE-MOYE — CONTENT COORDINATOR
LAUREN FERRELL — CONTENT COORDINATOR
LINDSAY SMITH — CONTENT COORDINATOR

SALES
CHRIS BRUMFIELD — SALES MANAGER
FERAS ABDEL — SALES MANAGER
JIM DYER — SR. ACCOUNT EXECUTIVE
ANDREW BARKER — SR. ACCOUNT EXECUTIVE
BRETT SAYRE — ACCOUNT EXECUTIVE
ALEX CRAFTS — KEY ACCOUNT MANAGER
BRIAN ANDERSON — KEY ACCOUNT MANAGER
SEAN BUSWELL — SALES DEVELOPMENT REPRESENTATIVE

MARKETING
SUSAN WALL — CMO
AARON TULL — DIRECTOR OF MARKETING
SARAH HUNTINGTON — DIRECTOR OF MARKETING
WAYNETTE TUBBS — DIRECTOR OF MARKETING COMM.
KRISTEN PAGÀN — MARKETING SPECIALIST
COLIN BISH — MARKETING SPECIALIST

Executive Summary

KARA PHELPS - CONTENT AND COMMUNITY MANAGER, DEVADA

The pace of innovation in database technology is a bit slower than it is in other niches of the industry, perhaps because it's so foundational. Software developers often learn how to work with databases at the beginning of their careers, and databases are central to software's ability to "eat the world."

This year, we surveyed 463 software professionals about how they use database technology, and we noticed several trends. A sizeable percentage of professionals are considering adopting NoSQL database management systems. Still, the majority of those surveyed work with applications that are using relational persistent storage models in production and non-production environments. As for data storage and security, there's no clear crowd favorite when it comes to partitioning functionally, horizontally, or vertically. Let's dig into these results.

NOSQL IS A PRINCE CHARMING

DATA

29 percent of respondents plan to adopt a new database management system within the next year. When asked what DBMS they are considering, 30 percent said Cassandra, 28 percent said MongoDB, 25 percent said Neo4j, and 24 percent said Amazon DynamoDB — all of which are different types of NoSQL.

IMPLICATIONS

While most developers have no plans to replace their existing database management systems, NoSQL databases are appealing options for those looking for something different. Whether these developers are considering moving from relational databases or they're already onboard with NoSQL, they're convinced about NoSQL's benefits.

RECOMMENDATIONS

Proponents of NoSQL database management systems say that NoSQL offers more flexibility, better scalability, and improved control over availability. As real-time web apps with ever-changing analytics grow increasingly central to business operations, NoSQL databases are becoming the best choice for more developers. This doesn't mean that your SQL skills are falling out of favor anytime soon — SQL and NoSQL have different features with different ideal use cases.

NOT ONLY NOSQL

DATA

85 percent of respondents using a MongoDB (NoSQL) database use a relational persistent storage model in production; 88 percent in non-production environments. 91 percent of respondents using a Redis database use a relational persistent storage model in production; 88 percent in non-production environments.

IMPLICATIONS

Relational technology, which is widespread, highly developed, and well-understood, continues to dominate when it comes to persistence models — even when used alongside NoSQL databases. Polyglot persistence, or using the best persistent storage model suited to each task, means that developers don't have to compromise on functionality.

RECOMMENDATIONS

No matter what type of database you use for your applications, you're probably already familiar with relational models for data persistence. While they aren't the right tool for every job, it's still difficult to beat the relational persistent storage model in terms of consistency and data integrity. Also, be aware that if you are looking to adopt a NoSQL solution, it is not going to solve all of your problems.

A LITTLE BIT OF THIS PARTITIONING STRATEGY, A LITTLE BIT OF THAT

DATA

Among respondents currently developing web apps, 45 percent partition their database functionally, 44 percent horizontally, and 43 percent vertically. Among respondents currently developing enterprise business apps, 47 percent partition their database horizontally, 46 percent functionally, and 45 percent vertically. Among respondents currently developing native mobile apps, 49 percent partition their database functionally, 44 percent horizontally, and 33 percent vertically.

IMPLICATIONS

Each category adds up to more than 100 percent, which is no mistake — a significant number of developers combine two or more strategies in their partitioning schemes.

RECOMMENDATIONS

It's a common practice to use different types of partitioning in concert with each other at key points in a database. It can even help with performance and scalability, but the conflicting needs of each type can also raise issues until you find an appropriate balance for your system. The unique requirements and goals of your system will dictate the particular mix of strategies that your database needs.

Key Research Findings

BY JORDAN BAKER - CONTENT COORDINATOR, DEVADA

DEMOGRAPHICS

For DZone's *2018 Guide to Databases*, we asked our community various questions about the state of the database field. We received 582 responses, with a completion rate of 79%. Using the industry standard confidence interval of 95%, we have calculated the margin of error for this survey as 3.6%.

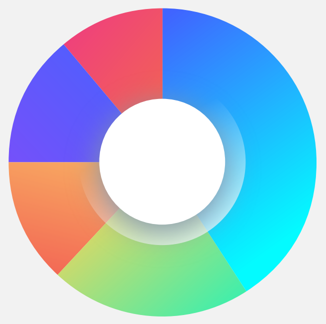
The demographics for the software professionals who took the survey are listed below.

- The average respondent has 18 years of professional experience in software.
- Respondents work for companies headquartered in three main geographic regions:
  - 41%: USA
  - 22%: Europe
  - 10%: South America
- Respondents live in four main geographic regions:
  - 30%: USA
  - 21%: Europe
  - 14%: South Central Asia
  - 11%: South America
- Respondents tend to work in enterprise-sized organizations:
  - 24% work for organizations sized 1,000-9,999
  - 22% work for organizations sized 100-999
  - 21% work for organizations sized 10,000+
- Respondents reported three main industry verticals:
  - 26% work for a software vendor
  - 14% work in financial/banking
  - 10% work in consulting
- The most prominent roles reported were:
  - 39%: Developer/engineer
  - 23%: Developer team lead
  - 17%: Architect
- Respondents typically work on one of three types of applications:
  - 85% develop web applications/services
  - 49% develop enterprise business apps
  - 26% developer native mobile apps
- The most popular programming language ecosystems reported were:
  - 84% Java
  - 72% JavaScript (client-side)
  - 40% Python
  - 35% Node.js (server-side)

SQL OR NOSQL? THAT IS THE QUESTION

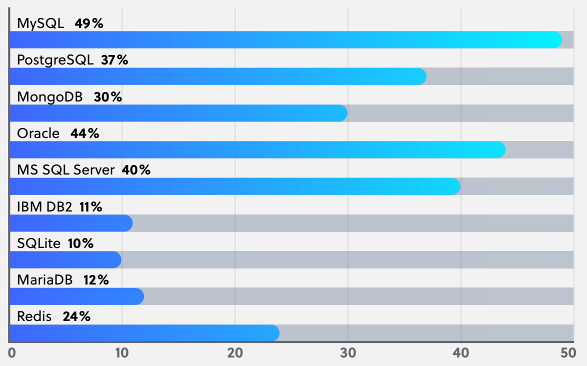
While we're having a bit of fun with the wordplay here, the

What percent of data that you work with is persistent versus ephemeral?



41	< 10% of data is ephemeral
21	11-25% of data is ephemeral
13	25-50% of data is ephemeral
14	>50% of data is ephemeral
11	Not sure

What database management systems do you use in production?



distinction between SQL and NoSQL databases in the world of software development is a very real one and will form the basis of many of our comparative analyses in this report. While SQL databases tended to have wider popularity among respondents, as we will see, NoSQL databases (specifically, MongoDB) proved more popular for certain development projects.

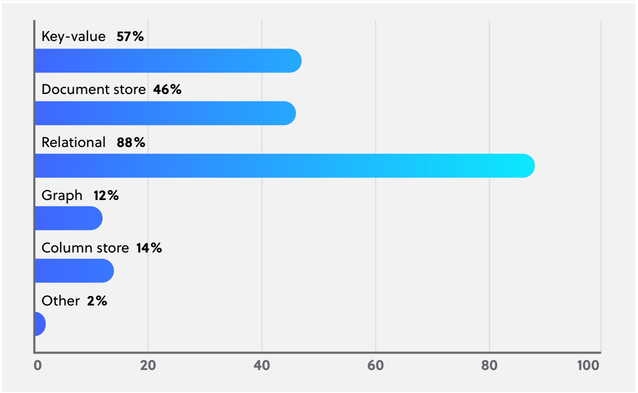
Among the general population of survey respondents, SQL databases dominated in both production and non-production environments. Among all respondents, MySQL proved the most popular database management system (DBMS). 49% told us they use MySQL in production and 54% reported using MySQL in non-production environments. Interestingly, Oracle proved a much more popular DBMS in production (44%) than non-production (30%) environments. PostgreSQL saw a similar, though statistically smaller, swing between environments. With 37% of respondents reporting to use PostgreSQL in production, it finished as the fourth most used DBMS in this environment. In non-production environments, however, PostgreSQL was the second most popular database management system (39%). NoSQL databases faired much worse. Only two NoSQL databases (MongoDB and Redis) had significant adoption rates, with 30% using MongoDB in production and 31% in non-production, and 24% using Redis in production and 18% in non-production. Despite the lower adoption rates for NoSQL, when we asked respondents which DBMSes they most enjoy, MongoDB (30%) finished in second behind MySQL (38%).

Given the above information, it's not surprising that 50% of respondents reported working mostly with SQL and some NoSQL, and 33% said they work only with SQL. But do these distinctions hold up for all developers, despite the type of applications being developed? To answer this question, we compared the data reported in the Demographics section regarding the types of software respondents develop to the data about the most popular database management systems. Let's first concentrate on the result for production environments. For respondents who develop web applications, MySQL (52%), Oracle (41%), and MS SQL Server (39%) constituted the top three choices of DBMS in production. For enterprise business application developers, Oracle proved the most popular at 54%, followed by MongoDB (50%) and MySQL (47%). Among respondents developing native mobile apps, 69% told us they use MySQL, 45% reported using MongoDB, and another 45% said they use MS SQL Server; Oracle came in a close fourth with 43% of respondents. For production environments, it seems that non-relational databases, particularly MongoDB, have garnered far more popularity among enterprise business and native mobile application developers.

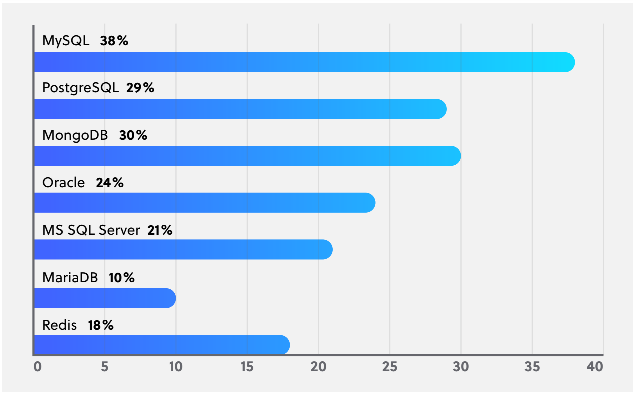
Interestingly, if we look at how these same types of developers (web, enterprise business, and native mobile) use these same database management systems in non-production environments, the popularity of SQL and NoSQL systems changes somewhat. Web app developers working in non-production environments reported MySQL (59%), PostgreSQL (41%), and MongoDB (34%) as their most used DBMS. Enterprise business app devs reported much the same, with 55% percent using MySQL, 40% using PostgreSQL, and 38% using Oracle (MongoDB finished in fifth among enterprise business app devs in non-production environments, with 35%). NoSQL databases took on a more important role among native mobile app developers, however, with 47% of mobile devs telling us they use MongoDB. In fact, MongoDB proved the second most popular option to use with native mobile apps in non-production, trailing only MySQL (70%).

By and large, it appears that most developers work with SQL databases. However, among those developers who told us they

Which persistent storage models does your application use?



What database management systems do you most enjoy working with?



are planning to adopt a new DBMS this year, the top four options under consideration were all NoSQL databases: Cassandra (30%), MongoDB (28%), Neo4j (25%), and Amazon DynamoDB (24%).

DATA STORAGE AND DATABASE PARTITIONING

In this year's survey, a vast majority (76%) of respondents reported that cloud technology has made working with databases easier. Despite this significant margin, 47% told us that they work with data located on-premise, 32% work with data in the cloud, and 19% house their data in a hybrid cloud solution. Comparing this data to our 2017 database survey, we find that the percentage of respondents using on-premise data storage remained static. Those respondents who use cloud data storage solutions grew 7% year-over-year, from 25% in 2017 to 32% in 2018. Correspondingly, the percentage of those who use told us cloud storage options make databases easier to work with increased by 21% (from 55% in 2017 to the 76% reported above). Interestingly, despite this growing faith in the cloud among developers and database professionals, 53% of respondents reported that 0% of their databases run in containers.

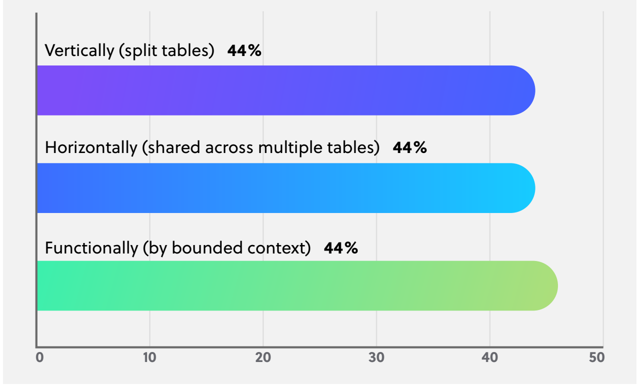
Despite the environment in which they keep their databases, respondents reported three main methods for partitioning their databases: functionally (by bounded context); vertically (split tables); and horizontally (shared across multiple machines). Functionally partitioned databases proved the most popular among respondents with a 46% adoption rate. Vertically and horizontally partitioned databases both claimed a 44% adoption rate. When we asked survey-takers about these same partitions in 2017, 37% told us they functionally partition databases (a 9% year-over-year increase) and 49% told us they vertically partition databases (a 5% year-over-year decrease). This year-over-year metric for horizontal partitions proved statically insignificant, however, going from 41% in 2017 to 44% in 2018, which falls within the margin of error for this survey.

Let's again turn to our list of popular SQL and NoSQL databases as a means of comparison. By comparing our data on popular databases to the data on ways to partition databases in production, we find that NoSQL databases do not fit the mold described in the previous paragraph. Of MongoDB users, 39% partition their databases horizontally, 34% partition functionally, and 31% partition vertically. Of Redis users, 32% partition horizontally, 30% partition functionally, and 24% partition vertically. Thus, it appears that NoSQL databases prove more popular for horizontal partitioning, which is interesting as horizontal partitioning came in as the least popular partitioning method in this year's survey.

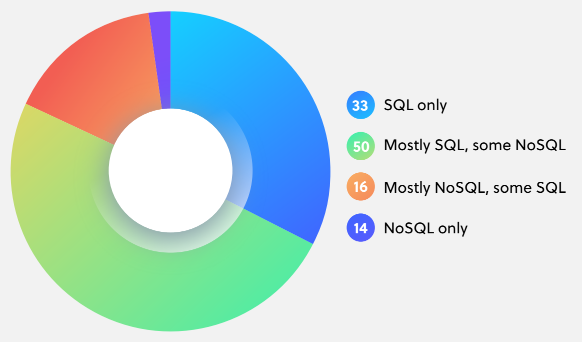
SQL databases used in production environments, however, tend to align well with the general partition popularity rankings reported above. For functionally partitioned databases, 50% of respondents use MySQL, 44% use MS SQL Server, and 40% use PostgreSQL. For vertically partitioned databases, 56% of respondents reported using MySQL, 48% using Oracle, and 48% using PostgreSQL. And for horizontally partitioned DBs, 48% use MySQL, 44% use Oracle, and 37% use MS SQL Server.

DBMSes used in non-production environments adhered to these same patterns. We have visualized the percentages related to database partitioning in non-production environments in the below graphs.

How do you partition your database?



Do you spend more time using SQL or NoSQL databases?



Evolving the Database for DevOps

BY KELLYN POT'VIN-GORMAN
DATA PLATFORM TECHNOLOGY PROFESSIONAL, **MICROSOFT**

QUICK VIEW

01. The role of the DBA is not going away, but evolving.

02. DevOps, although essential for most businesses to get ahead, must be done with the data being a central point.

03. Databases will serve the industry best by being less platform-specific and being better at serving up data quickly and transparently.

I'm writing this article with feelings of "conflicted passion." As a database administrator with two decades of experience, devoted to investments in platform expertise, I've honed my skills. I was known for building database systems, focused on making the most with the technology of a given database platform. While working at Oracle, I utilized strong skills in partitioning, PL/SQL code and functions, and other Oracle-specific features. A similar scenario developed when I worked with Microsoft SQL Server, MySQL, Informix, and others. I used the features that were built by the database vendor to make the most of the database platform. This solution created a double-edged sword. The business received the most from the platform they chose, but the cost was a single-platform solution for coding, release, and skillset. The solution often had challenges bridging to environments on other platforms, often requiring IT departments to have policies on what they would and would not support. Due to this cost, the DevOps engineer in me knows that we must change for DevOps to encompass the database more successfully, resulting in the conflict I feel.

Automation is key to information technology meeting increasing customer demands. The amount of online content on DevOps and agile processes tells us the importance of this evolution, but a majority of the focus in at the application tier and not the database platform applications rely on to provide much of its functionality. As my career moved from DBA to DevOps engineer, my focus changed from database platform-centric solutions to locating ways to automate and enforcing vanilla approaches to problem-solving.

DBA HEAL THYSELF

One of the goals of automation is to remove human intervention, which, in turn, removes human error and human bias, the former being a large attraction for database administrators (DBAs) to move toward DevOps. Automation of everything already present shouldn't be the finish line of DevOps, though. A continual enhancement and improvement to decrease the length of the development cycle and perfecting steps within the cycle should be an ongoing priority.

Many traditional DBAs are hesitant to embrace DevOps in for fear of their roles being made obsolete. For systems to evolve, human initiative, introspection, and inspiration are required. By removing tedious, repetitious tasks, we're able to free up DBAs to focus on these valuable endeavors. When DBAs realize the latter goal and recognize how their role can evolve as part of it, they have less hesitation and can become an important and pivotal part of the DevOps initiative.

Let's first remove the misconception of the single database platform environment. Consider the actual percentage of businesses with a single database platform. If we ignore policy and take the data from the environment as any indicator, the numbers will show that less than 10% are truly single platform. No matter the policies set by the IT department to drive a single platform decision, a unique vendor requirement or business-critical system on a secondary platform is very likely to exist. If you speak with any DBA, it's a common challenge of at least one or more outlier environments they're left to support, even if their training and experience haven't prepared them for it. With this myth dispersed, we can now assume that most technical environments have more than one database platform in a given technical environment.

Moving beyond this first myth, we might think we're closer to taking on the overwhelming hurdle of culture and processes part of any DevOps project, but the fact that we commonly leave the review of database changes to the end of the workflow demonstrates we aren't. It's a sign of poor communication between teams. And when we talk of the communication break between DBAs and Development, it's almost synonymous with IT. It's also a clear indicator that the cultural divide hasn't been addressed as part of the DevOps shift. Many may claim that it's due to philosophical or control differences between the teams, but at some point, this divide must be met and addressed as the blocker it is.

There is a roadblock factor that we experience repeatedly around this challenge. DevOps teams tighten timelines and rarely have the window to wait for a DBA to review changes, yet they also may feel hesitation to bring the DBA in earlier when concerns could add time at any point in the schedule. It's essential that a process be designed where developers, operations, and DBAs agree how changes to the database are handled and steps are defined to involve the DBA from the beginning. The earlier that DBAs are brought in, the fewer issues that will arise, creating a higher rate of success as time proceeds. Anxiety gives way to anticipated collaboration between each of the teams — a primary goal of all DevOps teams.

CHANGE IS BAD — NO, CHANGE IS GOOD

DBAs aren't fans of change, as consistency is often synonymous with stability. A DBA is often recognized as competent by environment stability, resulting in a high uptime for access to database environments. This being the case, DBAs can be viewed as a roadblock in their hesitation around change — and the development cycle is all about change. With development cycles part of continuous delivery, and with continuous integration putting significant pressure on stability, a DBA can begin to pressure toward slowdown of the development cycle. Even having all changes tracked in version control doesn't mean that outages won't occur, and outages should be expected at some point, even with mature DevOps practices. The most important goal is to have the DBA part of the process to eliminate as much potential of an outage or impact as possible. And if one does occur, lessons should be learned to remove the potential for any repeat.

One of the biggest ways to eliminate impacts from a development cycle is to ensure there are proper development and testing environments that are duplicates of production. Using virtualization to provide as many development and testing environments — as many as needed for multiple development cycles simultaneously in process — should be a priority of any DevOps group. Notice that I assigned responsibility for this to the DevOps group instead of the DBA group. The reason for this distinction is that we've held the DBA responsible for this for too long. As part of DevOps, and as teams are redistributed towards multi-role teams (developer, DBA, administrator, etc.) on each, it distributes the responsibility and the ownership for the resources across the business, too. It creates a buy-in for all involved to be part of the solution, using newer technical enhancements like virtualization, snapshots, and security measures to provide what's needed to succeed.

Many traditional DBAs are hesitant to embrace DevOps in for fear of their roles being made obsolete.

These teams may consider containerizing environments, as well. There are numerous types of container technology outside of just Docker. A popular solution, Kubernetes, can be used to create "pods" that will allow identification of what tiers belong to each other and the ability to package these separate tiers as one. It will conserve resources vs. using virtual machines and save money in the end. Containerizing can be very beneficial at the database level, too. There are options to snapshot and use virtualized databases as part of containers — even in pods of multiple database platforms, applications, and structured/unstructured flat files.

With this change to the housing of database sources together, a secondary data source for central analytics might not be required as often as it's proposed. SQL may have similar "flavors" between database platforms, but statements can result in different outcomes depending on the database platform. By using your databases as only data stores and by eliminating much of the proprietary code, forcing it outside the database layer, there will be less impact of changes between database engines. This will leave tedious and simple management at this point in technology, such as index management, query optimization, and statistics collection at the database tier to be automated by the database engine. The database is no more than a data store at this point, and this is where the DBA in me has a conflict. I must move beyond my pride in my database platform skills or loyalty to any product. The DevOps engineer in me must embrace solutions that work at a higher level than any unique platform.

In the end, we will hopefully see interfaces (both command line and graphical) that possess the ability to interact with a database platform, no matter the vendor.

This evolution will make use of object-relational mapping (ORM) frameworks, along with other tools to simplify the demands. I will make a choice to standardize how I deal with our database tiers and move more outside of the database so that we can create a set of unified processes when automating database tier work.

I will also move away from tightly coupled architecture environments with a centric database. Tightly coupled architecture can create more complexity than most DevOps endeavors are designed to manage. One way to resolve this is to build microservices on top of databases. By granting each microservice its own database — specifically, a virtualized copy of a database — DevOps is able to move as fast as any code and it's siloed to the point of fewer collisions with other development cycles, only requiring a final merge as the last step in the development cycle before deployment.

As a DBA, I will now need to (again) move past my traditional role to be centric to a specific database platform and simply treat the database as a storage layer. The database layer is there to serve up data to fulfill and analyze; nothing more. Yes, most database technologies have the technologies and features to be used for more than just to store data, but by doing so, it creates a complexity layer and lock-in to the database platform that also creates challenges later in the development lifecycle. By using microservices, you decouple the database from this challenge. The microservice will serve as a conduit for other microservices to deliver data vs. direct query of the database, limiting the demands on platform centric code or SQL.

In the end, we will hopefully see interfaces (both command line and graphical) that possess the ability to interact with a database platform, no matter the vendor. We already see this with Scala-based Spark. Developers can code in C#, Java, SQL, or Python, and yet, the output is the same at the analytical level. Database management must evolve in a similar way. With this need fulfilled, the conflict ends and the passion emerges for database management that is transparent to the database administrator, leaving cycles open to focus on DevOps automation, data quality, and code performance.

KELLYN POT'VIN-GORMAN is a member of the Oak Table Network and an Idera ACE and Oracle ACE Director alumnus. She is the newest Technical Solution Professional in Power BI with AI in the EdTech group at Microsoft. Kellyn is known for her extensive work with multi-database platforms, DevOps, cloud migrations, virtualization, visualizations, scripting, environment optimization tuning, automation, and architecture design. Kellyn has spoken at numerous technical conferences for Oracle, big data, DevOps, testing, and SQL Server. Her blog (dbakevlar.com) and social media activity under her handle @DBAKevlar are well respected for her insight and content.

Why Time Series matters for metrics, real-time, and sensor data

Download the e-book

"MySQL is not intended for time series data... I can testify it is like pounding nails with a screwdriver. It's definitely not what you want to do in any relational database."

John Burk, Senior Software Developer

influxdata®

The Modern Engine for Metrics and Events

Time Series: The Next Revolution of Technologies

Time series databases are the fastest growing database category, according to independent research, and store measurements tracked over time. This could be metrics and events from servers, applications, networks, sensors, or even trades in a market. The difference with time series data from regular data is that it is time-stamped, and analytics on this data is focused on "change over time." It comes in two forms: regular and irregular. Regular (metrics) are measurements gathered at regular intervals (e.g. CPU usage every millisecond). Irregular (events) are driven by some trigger, for example a sensor that gets triggered if the temperature exceeds some threshold.

Time is a fundamental constituent of any platform built to enable autonomy. Every data point has a time stamp so the system can understand when something was measured or when an event occurred. The platform must be able interpret and analyze data in real-time in order to take action while it's still meaningful and be designed for control type functions since having visibility into a situation is only useful if you are able to control what happens next.

Time Series Platforms are becoming a critical architectural component in every environment

THE TIME SERIES WORKLOAD

Time series data is very different from any other workload: it requires millions of writes per second, the ability to do real-time queries on huge datasets, time-based functions that help measure change over time, optimal disk compression with these large data sets, and the need to keep high value data accessible for real-time access, while storing older data with potentially different time precision available for historical analysis.

InfluxData is the leading time series platform and comes out of the box ready to use. Learn more.

WRITTEN BY MARK HERRING
INFLUXDATA, CMO

PARTNER SPOTLIGHT

InfluxData

The modern engine for metrics and events

influxdata®

CATEGORY	RELEASE SCHEDULE	OPEN SOURCE?
Time Series Database	Quarterly Release cycles	Yes

CASE STUDY

Coupa Software needed to create a custom DevOps Monitoring solution for their leading spend management cloud platform. With InfluxData they moved from pure data collection to predictive analytics and achieved a consistent track record of delivering close to 100% uptime SLA across 13 major product releases and 5 major product module offerings, as well as solving their data accessibility, aggregation, and retention challenges. Operational metrics are collected via Telegraf, stored in InfluxDB, and analyzed by Kapacitor. They use Grafana for visualization and have created a custom alerting framework. This has become the foundation to the path of building a system that is self-healing and can provide predictive analytics key to accurate forecasting.

STRENGTHS

- Fastest time to awesome
- Developer happiness
- Ease of scale-out and deployment

NOTABLE USERS

- Wayfair
- CERN
- Coupa
- Mulesoft
- Nordstrom

WEBSITE influxdata.com **TWITTER** @InfluxDB **BLOG** influxdata.com/blog

Four Use Cases Driving Adoption of Time Series Platforms Across Industries

BY NAVDEEP SIDHU
HEAD OF PRODUCT MARKETING, **INFLUXDATA**

QUICK VIEW

01. Why is time series data becoming so important?

02. Learn about the use cases that are creating a huge demand for time series data platforms.

03. What is the future of time series data platforms in the context of future AI/ML-driven apps?

Time series databases are growing fast, and as per DB Engines, they are the fastest growing database category, ahead of even the Hadoop and NoSQL data stores. This rapid growth of time series data platforms is due to several recent technology trends that are getting traction.

One of the main reasons is that modern application architectures are spitting out a huge volume of metrics compared to legacy application architectures. The newer applications today are being powered by microservices and instrumented by DevOps toolchains. This underlying architecture generates a lot of metrics and events data, which requires special handling. On top of that, these newer apps are being hosted on cloud platforms using VMs, Docker, and others that use orchestration tools like Kubernetes.

These factors have led to a deluge of metrics and events which now require storage, queries, and analytics. After trying to use different data stores, such as traditional RDBMS or Hadoop/NoSQL, users finally realized that such use cases require purpose-built storage and processing and finally adopted time series data platforms.

Let's take a deeper look into four use cases behind this tremendous growth in time series platforms.

DEVOPS

DevOps is not a new concept, but continues to gain traction and has become the biggest enabler of continuous delivery. Companies of all sizes are now adopting DevOps for faster delivery of their IT applications. The combination of app-dev, QA, and Ops teams makes natural sense, and every industry is adopting DevOps wholeheartedly.

Every organization uses a different set of tools for DevOps, based on what kind of technology they use, but it is fairly common for these toolchains to be comprised of several dozen technologies for deployment automation. Tools such as Chef, Puppet, Jira, and Git are used to instrument deployments using virtual machines and orchestration technologies like Docker, Kubernetes, and Mesosphere. The resulting orchestration ends up having dozens of moving parts and requires advanced monitoring to make sure that all the components are performant and meeting SLAs.

The good news is that all the tools and technologies used in a DevOps toolchain emit time series data, which can then be used to monitor, track, and analyze deployment speed and build performance. However, this requires a time series data platform which can ingest these metrics at a very fast rate, enable queries across the data set, and perform analytics in time to detect and fix any deployment or build issues. By using a purpose-built platform for metrics and events, DevOps toolchains can be monitored, tweaked, and optimized for multiple deployments per day.

IoT

Sensors and devices are enabling the instrumentation of the physical world. Every industry from manufacturing and automotive to healthcare and IT is racing towards implementing IoT systems. This move is resulting in these sensors and devices being embedded in everything from lightbulbs to solar panels, which means petabytes of data are being generated every second. This data is all time series data, which needs much more than simple storage. In fact, IoT has three unique requirements from data platforms.

1. MONITORING AND TRACKING

Data generated by sensors and devices provide information that can be translated into meaningful insights, such as battery performance, turbine production, shipment delivery status, and other information, which is usually mission-critical to business and improves productivity. Being able to quickly ingest and process this large volume of time series IoT data is the first step in any successful IoT project.

2. ANALYTICS

The next step is generating analytics from IoT data. Historical data from sensors is used to gain insights that can be applied to the current situation to create a major competitive advantage. Predictive maintenance, optimized traffic routing, improved churn management, and enhanced water conservation are all possible with IoT analytics.

3. ACTION AND CONTROL

Last but not least, IoT data can be woven into business processes to trigger intelligent actions. With the speed and velocity of events being generated by sensors, businesses want to act on this data in real time with no human intervention. For example, actions such as automatically shutting down a pump in case of a leak or changing a wind turbine's direction with wind speed all create an immediate business advantage. Time series platforms with built-in action frameworks enable such operations.

MICROSERVICES

Microservices are incredibly helpful when building scalable and reliable architectures. Everything related to how IT services are delivered and consumed is undergoing tremendous change. Monolithic architectures are being replaced by microservices-driven apps, and cloud-based infrastructures are being tied together to deliver microservices-based architectures, which are delivering high-performing scalable apps 24/7.

To implement a microservices-style architecture, there are a few fundamental requirements.

SCALABLE WEB ARCHITECTURE

One of the first requirements of such an architecture is to ensure a high level of scalability, so these microservices are often deployed on the cloud so that they can scale on-demand. The cloud and container architectures powering these microservices require sub-second monitoring to trigger more capacity as needed, and that is usually achieved by analyzing the time series metrics generated by the stack.

CROSS-DOMAIN VISIBILITY

A microservices-powered application can be built by different teams spanning multiple IT domains. One team could be responsible for "customer account creation" and another team could be responsible for a microservice for "order creation." When new customers complain that it is taking a long time to place an order, the problem could be on either side. By using time series data generated by the microservices stack, performance metrics such as response times can be measured across multiple microservice domains.

DISTRIBUTED ARCHITECTURE

A distributed architecture is required either for technical reasons, such as scalability and reducing latency, or reasons such as location, jurisdiction, or disaster management needs. The distributed nature makes a lot of sense but creates a monitoring challenge. Different versions of the same microservice could be deployed in different areas — how do you track the performance against benchmarks and SLAs at a sub-second level? Again, time series data platforms come to the rescue and use metrics and events to track performance.

REAL-TIME ANALYTICS

Due to reasons including the ones outlined above, the volume of metrics and events generated by modern apps is beyond what any human is able to realistically interpret and act on. How do we learn what's going on and take the best action possible? Machine learning with real-time analytics is crucial in finding the "signal from the noise." Whether the organization needs real-time analytics to buy and sell equities, perform predictive maintenance on a machine before it fails, or adjust prices based on customer behavior, processing, analyzing, and acting on time series data in real time is the problem to solve.

OBSERVE

Observing systems in real time is critical as complex systems, by nature, behave and fail in very unpredictable ways. Observing different metrics, such as boiler temperature, order volume, unique visitors, and even infrastructure metrics such as system stats during the holiday season, are all part of understanding application and user behavior.

LEARN

The learning process is all about teaching the system what data means and how to make decisions automatically. AI/ML is so powerful in terms of what it can do, but it needs tons of data to learn. Time series data platforms bring metrics and events from mission-critical apps and infrastructure so that the algorithms can learn what's going on.

AUTOMATE

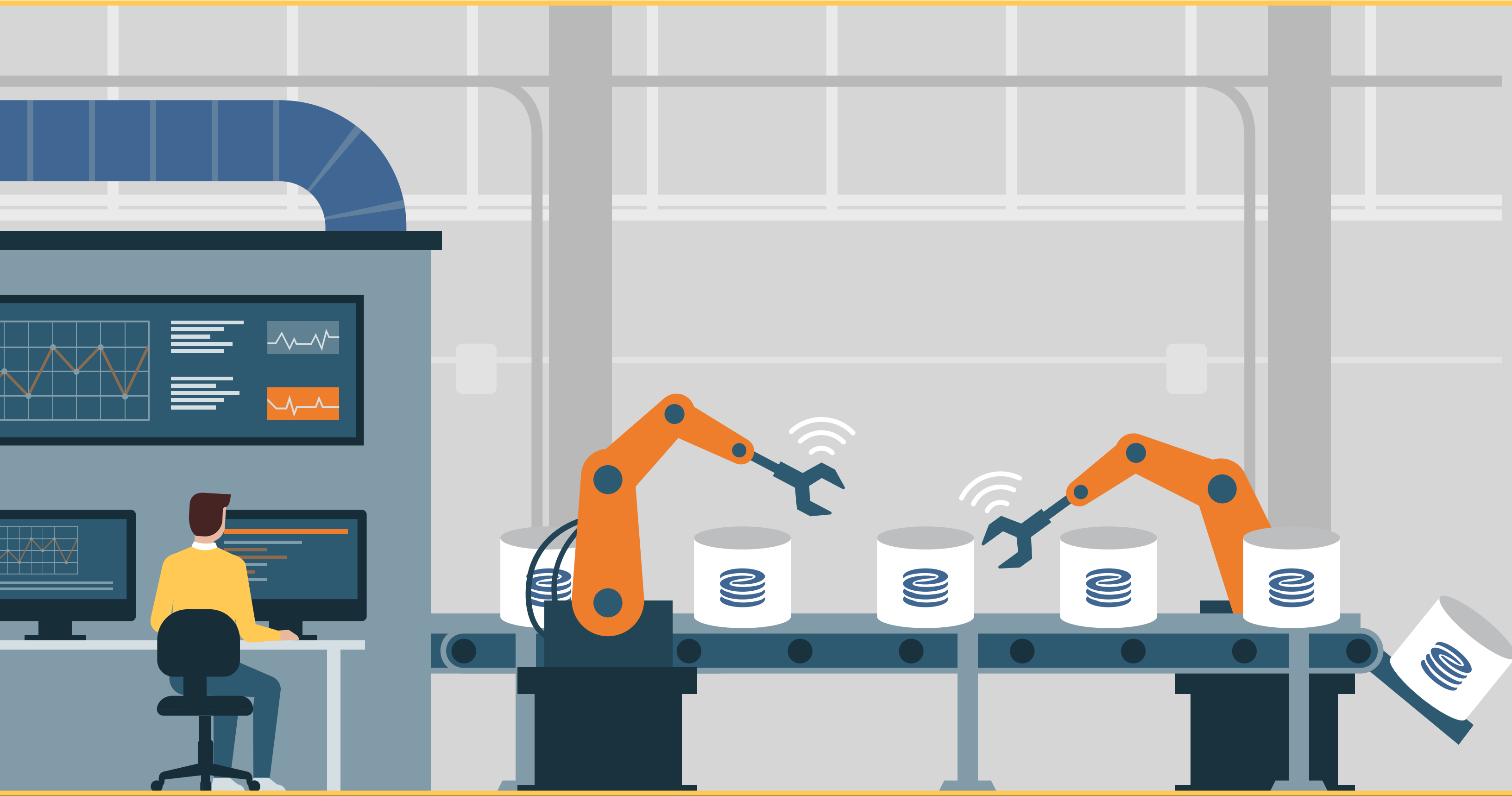
After the analytics are generated, the next step is to automate the decision. The decision could simply be predicting when to request more resources for the app or a decision to give a customer a special discount. Such decisions could be made by the app itself or by another AI/ML app. In some cases, it is simply alerting another app to take action.

IN SUMMARY

IT apps are undergoing a sea change. Across industries, there is a realization that organizations should turn to AI/ML-driven smart apps, which can drive automation with the help of IoT and advanced instrumentation. Such app architectures are powered by microservices and built using DevOps toolchains, with the ultimate goal of rapid upgrades to provide an excellent customer experience. As the mechanisms to deliver these apps have evolved, so has the need to monitor and instrument them. Time series data platforms that can handle a large number of metrics and events will continue to grow with the requirements of AI, predictive analytics, and newer app architectures.

NAVDEEP SIDHU is Head of Product Marketing at InfluxData which provides the leading time series platform to instrument, observe, learn, and automate any system, application and business process across a variety of use cases. Prior to joining InfluxData, he was VP of Product Marketing at Software AG where he focused on delivering platforms for building and integrating enterprise applications. Earlier in his career, he spent over 7 years with Deloitte Consulting building and integrating large scale mission-critical applications followed by 3 years at Sterling Commerce where he launched B2B Gateway and Partner Community Management applications.

Datical



DevOps Meets Database

Datical is the #1 database release automation solution that:

- Enables database code to be treated just like application code
- Provides broad database platform support
- Delivers safe, secure and scalable automation

... with prebuilt plug-ins to popular DevOps tools.

See Datical in action:
www.datical.com/demo

Database Release Automation

Software is reshaping every industry – from healthcare, to finance, to retail, and beyond. The pace of change is accelerating, and companies must master their application release process in order to survive in the digital era. In the software economy, innovation and speed is the new standard.

As companies have adopted automation to help increase the pace of software delivery, a new constraint has emerged: data. The database has long been a unique, separate discipline, not part of the DevOps tool chain. In fact, most companies — even those are using the latest DevOps automation tools — are still managing and deploying database changes manually, anchoring development teams.

That's why we developed Datical. Our database release automation solution is designed to break the application release bottleneck created by today's manual database deployment process. Datical automation offers a great opportunity to improve productivity and performance, allowing development, testing, and DBA staff to focus on more important projects and initiatives. Database release automation also helps to eliminate otherwise unavoidable incidents of human error, while increasing data security and application performance and reliability. Put simply, database release automation from Datical helps speed the delivery of better performing applications into production faster, safer, and with more reliability.

Datical is strategically guiding some of the world's most admired companies through their database deployment modernizations efforts and delivers business innovation through database automation. Learn more here: How to Get Started with Database Release Automation in 4 Easy Steps.

WRITTEN BY BEN GELLER
VICE PRESIDENT MARKETING, DATICAL

PARTNER SPOTLIGHT

Database Release Automation

Datical

Our applications are very data-driven, so it's very important database code changes consistently move along the software delivery path with application code changes.

CATEGORY	RELEASE SCHEDULE	OPEN SOURCE?
Continuous Delivery and Release	Continuous	No

CASE STUDY

The team at Colonial Life, part of the Unum Group, found itself in the middle of a critical effort to improve software delivery. They quickly realized the process to deploy database changes could not keep pace with the rest of the application delivery process. Aligning database changes with application code changes was a manual task that added considerable friction to software releases. The team at Colonial Life knew they had to automate their database deployment process to realize the software delivery improvements sought by the company. Colonial Life now uses Datical to automate database releases. As a result, they have been able to increase the amount and pace of software releases they deliver from one every 17 weeks to one every 7 weeks.

STRENGTHS

- Treat database code just like application code. Create database Continuous Delivery by unifying application and database changes.
- Capture DBA expertise in programmatic rules that can be enforced during the application build and release process.
- Simulate the impact of database changes before they are deployed.
- Automatically track and report the status of every database deployment across the enterprise.

NOTABLE USERS

- NBC Universal
- Nike
- The Standard
- Sony
- Anthem
- TD Bank
- Travelers
- UPS

WEBSITE datical.com **TWITTER** @Datical **BLOG** datical.com/blog

Making Your Data Intelligent

BY MAX DE MARZI
GRAPH DATABASE EXPERT

QUICK VIEW

01. Using graph databases, your data knows exactly what it's connected to and how.

02. The cost of a graph query is determined by how many relationships are traversed to find the answer, not the size or complexity of the data.

03. The shape of the data can be as important as, if not more important than, the values of the data.

Back in 1965, Ted Nelson coined the term "hypertext." In his vision, documents would have Paths going in both directions connecting them all together. On the web that Tim Berners-Lee built, we only got half that vision since what we know as links today only travel in one direction. It wasn't until the search engines started mapping these links into a graph that the web started to make sense.

A row of data stored in a relational database has an even worse story. In order to understand how it is connected, you must tell it exactly which tables to join and how to join them. It doesn't even understand the concept of links — those are reserved for the auxiliary join tables. The data stored in a relational database require your SQL expertise to delivery any value. But what happens when we move that data into a graph database? What happens when your rows become nodes and your join tables become real relationships connecting them together? All of a sudden, your data knows exactly what they are connected to and how they are connected.

Your data becomes intelligent, so your queries don't have to. You can now simply ask, "Are these two things connected in some way?" and they will tell you so. What about indirectly connected? Three, four, five jumps away? They know. It's the same data you've had all along, but it can now start to reveal its secrets. These are the kinds of questions you may have never even thought to ask, but you will. While some enterprises are still working on data lakes with varying levels of success, leading organizations are looking over the horizon at the next thing: the knowledge base.

Yes, knowledge bases. Just like Bumblebee from the original Transformers, somebody came back in a DeLorean from 1985 to make them cool again. But this time, they are bigger, they are more powerful, and they may yet survive the AI winter that killed their earlier incarnations. Only this time, it's not just about building expert systems. It is about understanding the fabric and structure of enterprise data — understanding the entire business, end to end, and turning that into knowledge. What are the ways these things over here connect to those things over there? For example, given the over 1,000 press releases from today alone, which ones should you read because they are more likely to have an effect on your investment portfolio? Not simply because the press release mentioned one of your investments, but because their customers, partners, suppliers, competitors, subsidiaries, etc. were mentioned two and three levels away?

How would it know? Is it a fortune teller or an oracle? No; there is no magic in computer science. So, how exactly do graph databases make data intelligent, then? The trick is in how the data is stored. The following description is a little abstracted from the underlying mechanics, but the principles are the same: You have an array of nodes, and you have an array of relationships. The relationships point to the node they came from and the node they go to, just like a join table would in a relational database — except they don't use foreign keys that have to be found in indexes. Instead, they use pointers.

The nodes each have sets of relationships grouped by type and direction (incoming or outgoing). Let's take the example of a social network. Node 1 is a user. It has 150 "friends" relationships, 40 "likes" relationships, 80 "posts" relationships, etc. We want to get the emails of their 150 friends. Starting at node 1, we go to the set holding the 150 outgoing friend relationships, jump to each one by following a pointer, then from each of these, follow a pointer to the node on the other side of the relationship. Then, for each one, we grab their email address.

It's called "index-free adjacency" but it really means that traversals are done following pointers instead of looking up keys. This friends query will take the same amount of time regardless of whether there are one million or one billion users in the database. The sets of relationships grouped by type and direction also help us ignore connections we don't want. Instead of a "posts" relationship type, imagine one per day… POSTS_2018_12_31, POSTS_2019_01_01, etc. If we want to see the things our friends posted on any particular day, we can just follow that day's relationship type — which means our query scales regardless of how many posts our friends have over a long period of time because we are only looking at one day's worth of data in our query.

You can play the same trick in relational databases by using partitioning or having one join table between users and posts per day. But that's like teaching an old dog new tricks; with graph databases, you get this for free. Another thing to note is that following the 150 "friends" relationship is (almost) the same as following a single "parent" relationship 150 times in 150 different nodes. The cost of a query is determined by how many relationships are traversed to find the answer, regardless of what or where those relationships take us and regardless of the overall size or complexity of the data.

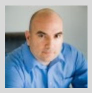
In many of today's enterprises, data is kept in silos, even though customer data is connected to products, process, supply chain, manufacturing, staff, legislation, and all the underlying systems that support the entire organization (and, in some cases, partners, suppliers, and competitor data, too). How would the effect of new legislation for your suppliers affect your business? It's hard to know as long as the data remains separate. Newer master data management projects are using graph databases because they are perfectly suited to handle this kind of project.

As data grows and structure changes, new relationship types and new types of objects can be added on the fly. A key concept in graph databases is that schema is optional or fluid. Nodes of the same type may have different properties and relationship; they don't store null values. While in relational databases, you end up with wide tables filled with nulls, the graph databases keep only what exists.

Speaking of existence, when building recommendation systems, we look for relationships that don't exist but should. You should buy this product or watch this movie. We know this because people who have a similar behavior to you have bought that product or watched that movie. In fraud detection systems, we look for relationships that should not exist. Your insurance claim processor just so happens to be the cousin of the husband of the person making a dubious claim. Recommendations and fraud are two sides of the same coin, both addressed very well by a database that makes relationships first class citizens.

It's not only relationships, though. Having data in a graph makes pulling graph metrics a snap. How many nodes are connected to this node? How many of those are connected to each other? What is the sum of the weight of those relationships? What is the value of the strongest relationship? Which is the most central by betweenness, closeness, harmonics, etc.? What are these metrics two hops or three hops out? What are people doing with these metrics? They are using them to feed machine learning models where the structure of the subgraph of a node may be just as important if not more so than its properties.

This is key and bears repeating: The shape of the data can be just as important as the values in the data. We may not know how old a user is, and they may have friends of all ages, but if many of those aged 35-40 are all friends with each other, chances are that the user is in that same range. The financial transfers may seem normal, but you realize that the recipients of those transfers are all connected somehow through deep and tangled corporate ownership links. Graph databases will allow you to clearly see these relationships that were hiding in the data all along. As your data becomes intelligent, so do you.



MAX DE MARZI is a graph database expert. His graph power level is over 9,000, but he is terrible at just about everything else. Over the past six years, he has written over 140 blog posts on graph use cases on his blog at maxdemarzi.com where he waxes poetically and obsessively over these collections of points and lines. If you have a question about graphs, don't be shy… connect.

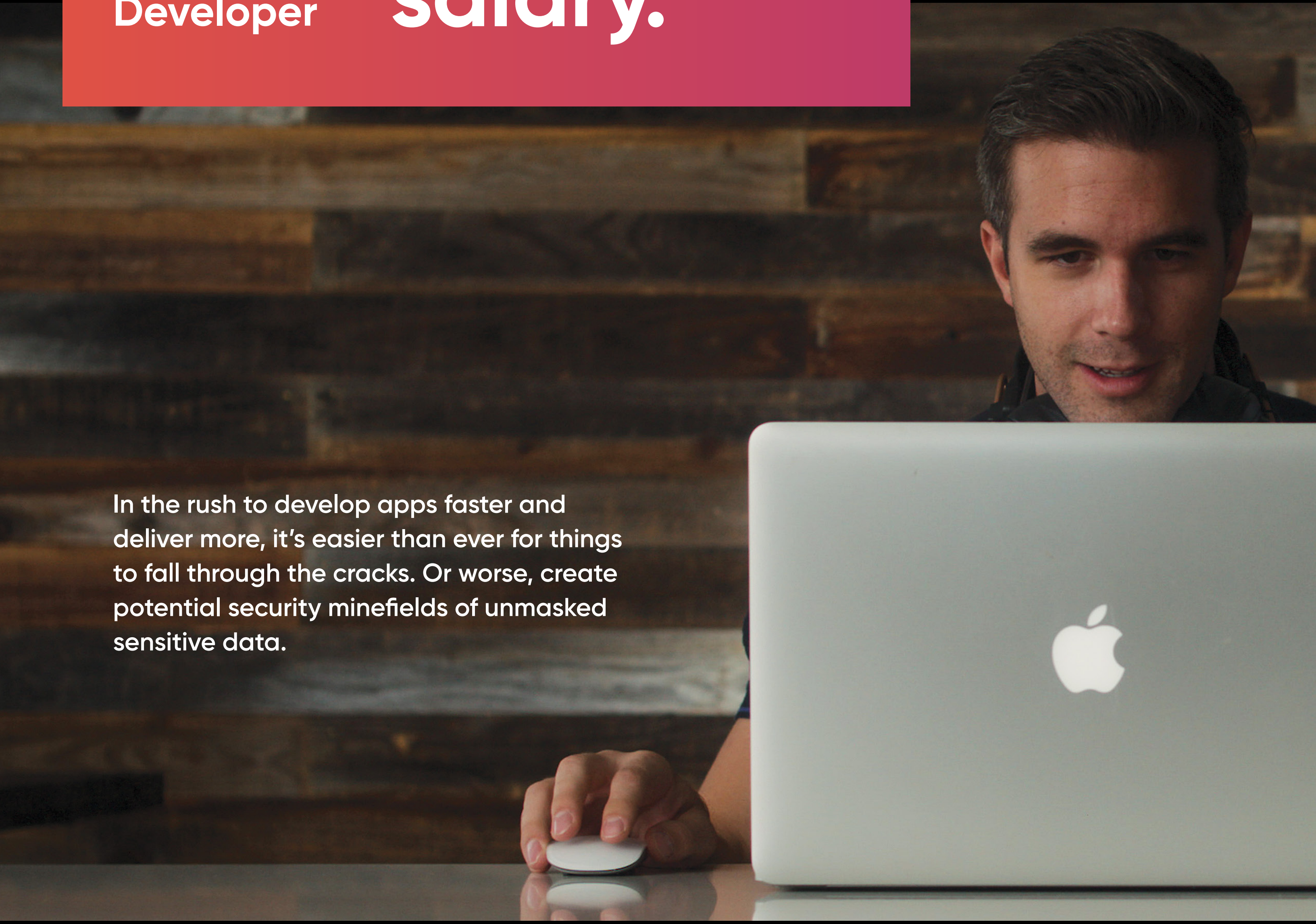
PRODUCED BY

DELPHIX

CONFESSIONS

Developer

"I know everyone's salary."



In the rush to develop apps faster and deliver more, it's easier than ever for things to fall through the cracks. Or worse, create potential security minefields of unmasked sensitive data.

Download the Developer Confessions ebook to discover the biggest frustrations we hear from developers, how to balance access to data with data security, and proven ways to speed up developer productivity.

DOWNLOAD EBOOK

Data Management for DevOps in the Cloud

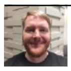
DevOps teams struggle to understand how to incorporate data environments into their Software Development Life Cycle (SDLC) automation practices, especially when they rely on data sources running in the cloud.

Enterprise teams need automated access to masked and secure data to move faster and write higher quality software. Adopting a DataOps approach solves these data management challenges.

For example, a leading financial institution wanted to focus on creating a personalized on-demand experience for every developer and tester as a part of its digital transformation strategy. Agile teams were struggling to improve the customer experience as they did not have access to personal data streams. When developers received data, it was incredibly out-of-date, but database administrators (DBAs) were unable to move faster due to a lack of resources and regulatory constraints.

That's when they decided to take a DataOps approach. Rather than focusing on fulfilling individual requests for data, the DBAs established and secured multiple data source copies as data pods tailored to each agile team. The development teams were able to refresh, reset and share data within their own secure data environments. Consequently, data quality increased and was available earlier in the life cycle, leading to a higher quality of the product.

The Delphix Dynamic Data Platform helps accelerate the adoption of DataOps by providing both developers and DBAs the tools they need. It allows DBAs to focus on security, privacy, and governance rather than fulfilling requests for new data on an ad-hoc basis. Developers are now empowered to control their own secure copy of production data on a self-service basis.



WRITTEN BY DEREK SMART
SENIOR STAFF ENGINEER, DELPHIX

PARTNER SPOTLIGHT

Delphix Dynamic Data Platform

D Ξ L P H I X

Delphix, the company accelerating innovation through DataOps

CATEGORY	RELEASE SCHEDULE	OPEN SOURCE?
Data Management, Data Virtualization and Masking	Every 2 months	No

CASE STUDY

As the largest dental benefits system in the U.S., Dentegra depends heavily on software applications to support the orchestration of core business processes, including contracts management, customer onboarding, and claims processing.

Moving to the cloud is part of Dentegra's long-term digital strategy to improve scalability and time to market across its application portfolio. However, data-related challenges stood in the way of realizing the full potential of cloud.

"The combination of Delphix and AWS gives us the agility we need to succeed in today's application-driven economy. By easily and securely moving data to the cloud, we're able to release new features to the market faster, while also lowering cost and risk."

- Shan Swaminathan, VP of Application Delivery and DevOps, Dentegra

STRENGTHS

- **Accelerated provisioning:** By automating the data provisioning process, Delphix makes it possible to deliver data in minutes, paving the road to continuous deployment.
- **At scale delivery:** The Delphix platform makes it possible to run parallel environments by standing up multiple full copies in the space of one for testing and other activities.
- **Self-service operation:** Delphix enables end users to perform data operations in a few clicks, removing the dependence on operations staff and ticketing queues.
- **Branch and version data like code:** Delphix makes it possible to fork development environments.
- **Integrated data security:** Integrated data masking allows developers to get the data they need, without putting the company or customers at risk.

NOTABLE USERS

- Dentegra
- Paybay
- MetroBank
- Molina
- eHarmony

WEBSITE delphix.com **TWITTER** @Delphix **BLOG** delphix.com/resources/blog

Polyglot Persistence in Practice

BY OREN EINI
CEO AND FOUNDER, **HIBERNATING RHINOS**

QUICK VIEW

01. Modern applications store data in many different formats and locations, for different needs and usages.

02. A crucial part of your system architecture is defining the ownership and flow of the data between these locations.

03. Proper data ownership allows you to segment your application to independent microservices and create a scalable architecture.

It used to be the case that your database choice was limited to which relational database vendor you use (and often, that was settled ahead of time). In the past fifteen years, a lot has changed, including the baseline assumption that you'll have a single database for your application and it will be a relational one.

I'm not about to rehash the SQL vs. NoSQL vs. NewSQL debate in this article. Instead, I want to focus on practical differences between the various approaches and some of the ways that you can use the notion of polyglot persistence to reduce the time, effort, and pain it takes to build modern applications.

Let's talk about the most classic of examples: the online shop. We have the customer, orders, and shipping as our main concerns. It used to be that you would start by defining a shared database schema among all these components. That led to many interesting issues. The way the Customers module works with a customer record and the way the Shipping module works with it are completely different.

It isn't just a matter of different queries and the need to balance different indexes for each of the distinct parts of the system. If a customer changed her name, you don't want to refer to her with the old name. On the other hand, a package that was sent to that customer before the name change needs to still record the name that we sent it as.

If you want to implement a recommendation system for the customers, you'll quickly realize that structuring the data for each quarterly report is going to make your life… unpleasant. In short, a single database and a single schema to rule them all isn't going to work (except in Mordor).

The obvious answer is to split things up — to use different schemas for different pieces of the system and allow them to build their own data models independently. A natural extension of that is to use the best database type for their needs. In this case, I'm not talking about selecting MySQL vs. Oracle but relational vs. document vs. graph vs. column store.

The biggest challenge in such a system — where you have multiple distinct data siloes that operate in tandem — is how you actually manage, understand, track, and work with all the data. This is of particular importance for organizations that need to deal with regulations such as GDPR or HIPAA. Note that this isn't a challenge that is brought upon by the usage of multiple database storage types. This is brought upon because the amount of complexity we have to deal with in our systems is so high that we have to break them apart into individual components to be able to make sense of them.

In the old days, data interchanges used to be done with flat files. If you were lucky, they were self-describing (CSV was an improvement over the status que, if you can believe that). In many cases, they were fixed length, arbitrary formats with vague specifications. Today, we have much better options available for us. We can utilize dedicated ETL tools to move data from one system to another, transforming it along the way.

One thing that hasn't changed in all that time is the absolute requirement of understanding the flow of data in your organization. The recommended way to do that is to define an owner for each piece of data. The Orders service is the owner of all the orders' data. It is the definitive source for all data about orders. If the recommendation engine needs to go through the past orders, it can only do that through published interfaces that are owned by the Orders service team.

A published interface doesn't have to be a SOAP/REST API. It can be a set of files published to a particular directory, an API endpoint in which you can register to get callbacks when data changes, or an internally accessible database, and is meant for external consumption (and is distinct from whatever format the data is kept on internally).

Fifteen years ago, this was called service-oriented architecture; recently, it has become common to call it microservices. Regardless of what it's actually called, the idea of separation of concerns and ownership of data is critical for successfully separating the different parts of the system into individual components.

Today, we can utilize dedicated ETL tools to move data from one system to another, transforming it along the way.

Once clear boundaries have been established, it is much easier to start working on each of the different pieces independently. The Orders service can use a document database, while the recommendation engine will likely use a graph. Analytics and reporting are handled by a column store, and the financial details are recorded in a relational database.

Previously, when I talked about published interfaces between the components, I was careful to not use the term API. Indeed, in most cases, I would recommend against this. When we need a different part of the system to perform something for us, by all means, call to it. But when we need a piece of data, it is usually better to have it already available than to have to seek it out.

Consider a simple case where shipping needs to know whenever a customer has a preferred status so it can apply a different shipping charge to their orders. One way of doing this is to call, for each of the orders that shipping goes through, to the customers' service and ask it about the customer's status. This is a gross violation of service independence. It means that shipping cannot operate if the Customers service is not up and running.

Ideally, we want to have better isolation between them. A better approach is to store the relevant data locally to the Shipping service. This means that the customers' preferred status may take a while longer to update in shipping's database. At worst, that may mean that a customer will not get the preferred rate. That is something that can easily be fixed with a refund and is likely to be rare, anyway. The benefit of separating the systems, on the other hand, is an ongoing effect.

Beyond abstract ideals of architecture, there is also the overall system robustness and performance. By having the data for common operations in-house, so to speak, we can save a lot of time and effort for the entire system. It is easy to forget it during development, but queries and calls to out of process (and machine) have non-trivial cost to them that really adds up. I'll point you toward the fallacies of distributed computing for more details on how common this issue is, and how fatal it can be for your overall system health and performance.

You might have noticed that beyond peeking a bit at the different database types, I haven't actually dealt with the interactions of the different data models. That is because for the most part, this is an implementation detail at the overall system architecture level. What is important is the notion of data ownership and data flow — identifying who gets to modify a piece of data (only its owner) and how it is distributed in in the system.

Once that problem is solved — and it is anything but a simple one — the actual decision of putting the data in a table or a document model is a matter of taste, the specific of the problem at hand, and (most importantly) a reversible decision that has a local, not global, impact.

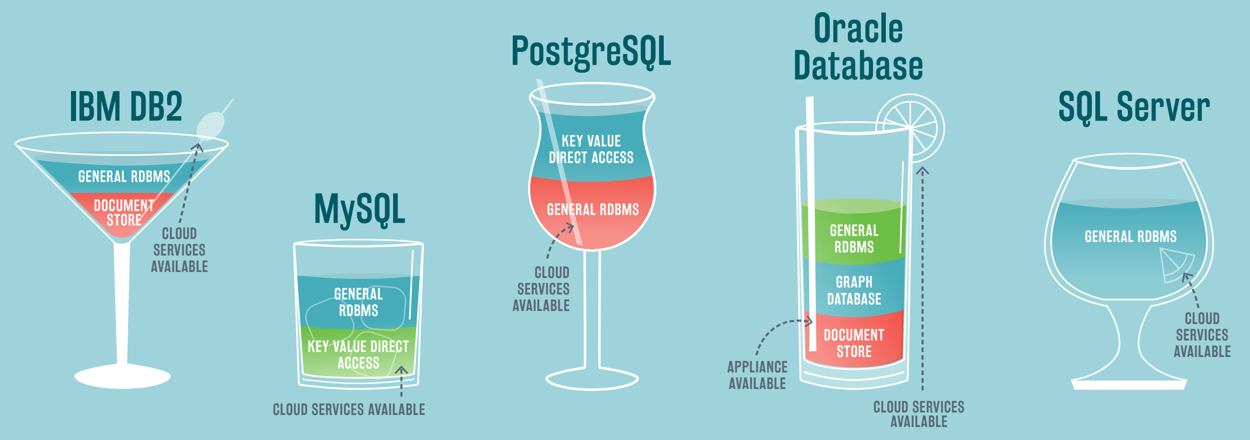
OREN EINI has more than 20 years of experience in the development world with a strong focus on the Microsoft and .NET ecosystem. Recognized as one of Microsoft's Most Valuable Professionals since 2007, Oren is also the author of "DSLs in Boo: Domain Specific Languages in .NET." He frequently speaks at industry conferences such as DevTeach, JAOO, QCon, Oredev, NDC, Yow!, and Progressive.NET. An avid blogger, you can also find him under his pseudonym as Ayende Rahien at ayende.com/Blog/.

THE MIXOLOGY OF DATABASES

With so many different flavors of databases, it's hard to tell what the ingredients are for each one. Here we've mapped a portion of the database landscape onto a menu of mixology! Take your pick of Relational, NoSQL, NewSQL, or Data Grids. And remember, you can pick more than one! That's why they say polyglot persistence is so tasty. *Cheers!*

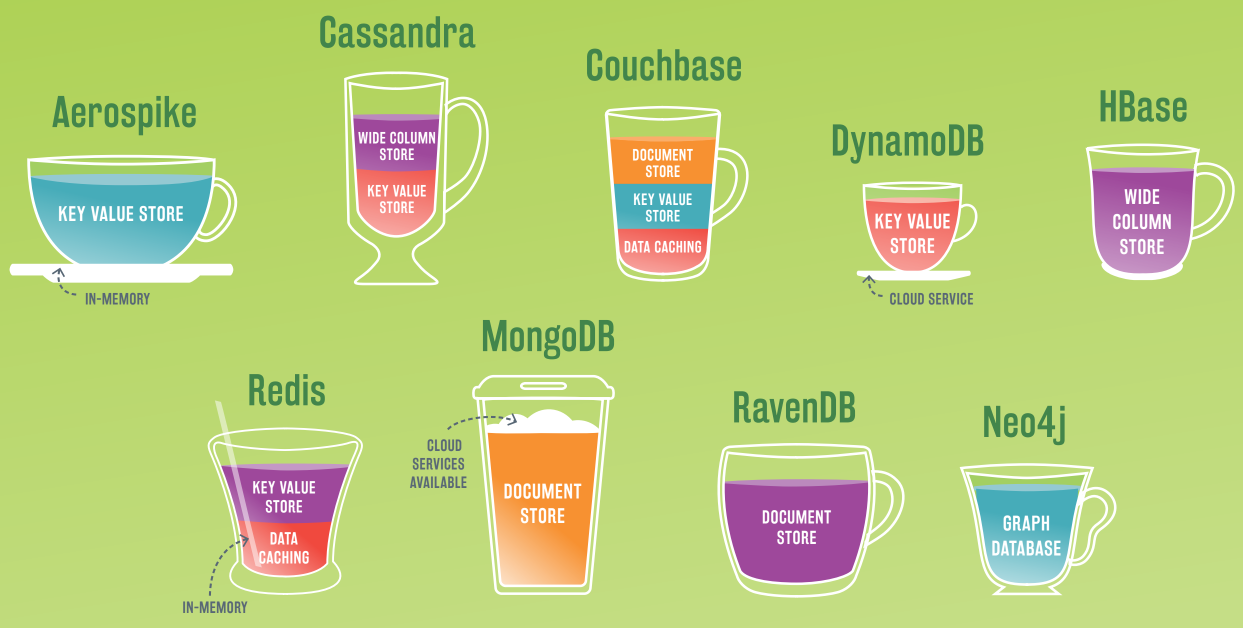
RELATIONAL

(AS REPRESENTED BY LIQUOR GLASSWARE)



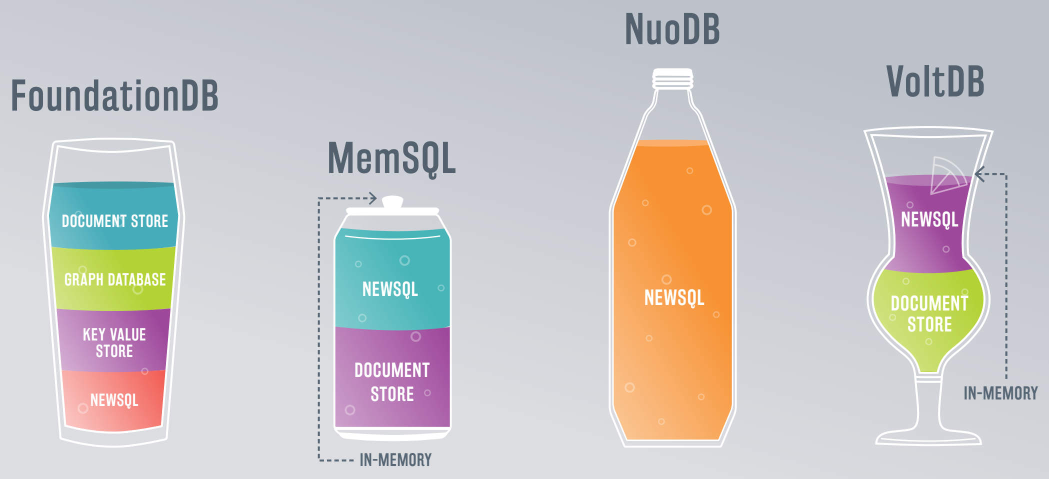
NOSQL

(AS REPRESENTED BY COFFEE AND ESPRESSO GLASSWARE)



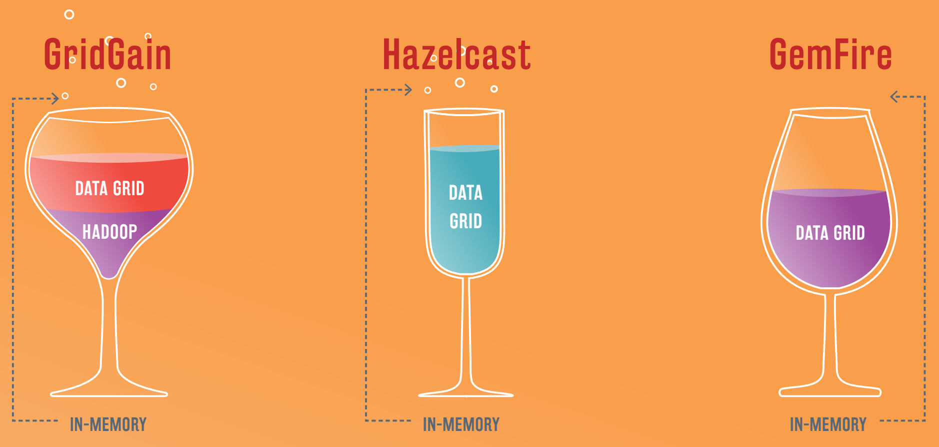
NEW SQL

(AS REPRESENTED BY BEER CONTAINERS)



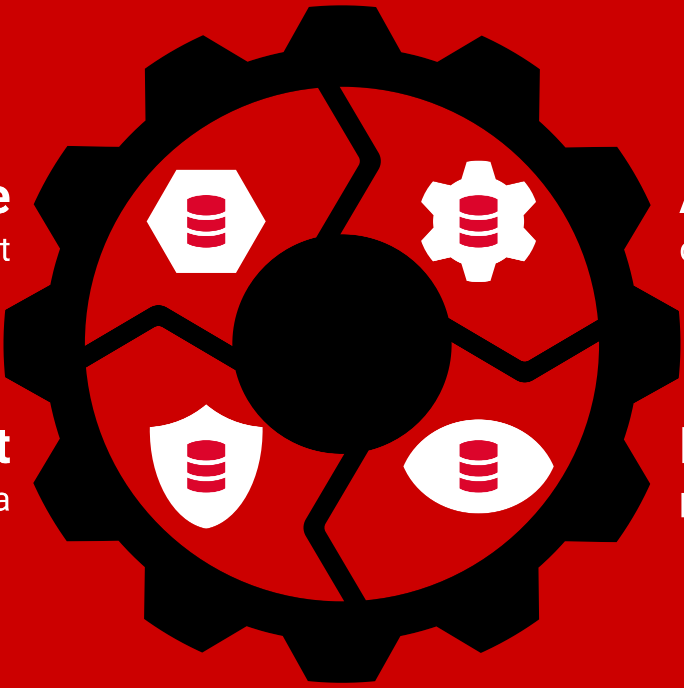
DATA GRIDS

(AS REPRESENTED BY WINE & CHAMPAGNE GLASSWARE)



Compliant Database DevOps

Deliver value quicker while keeping your data safe



Standardize
team based development

Automate
database deployments

Protect
& preserve data

Monitor
performance & availability

Redgate helps IT teams balance the need to deliver software faster with the need to protect and preserve business critical data.

You and your business benefit from a DevOps approach to database development, while staying compliant and minimizing risk.

Find out more at
www.red-gate.com/DevOps

redgate

The Rise and Rise of Compliant Database DevOps

Over the last five years, the Accelerate State of DevOps Report from DORA has consistently shown that higher software delivery performance produces powerful business outcomes. The 2018 report, for example, reveals that the best performing organizations which adopt DevOps release changes 46 times more frequently and have a change failure rate that is 7 times lower

There's a welcome addition ixn the latest report, however, because it now calls out database development as a key technical practice in DevOps. It's a notable shift in our understanding of DevOps workflows because it moves the database from being a bottleneck to a participant.

The report reveals that teams which do continuous delivery well version control database changes and manage them in the same way as the application. Interestingly, when it goes on to talk about overall software delivery performance, it states:

"Those that develop and deliver quickly are better able to experiment with ways to increase customer adoption and satisfaction, pivot when necessary, and keep up with compliance and regulatory demands."

Compliance and regulatory demands have entered the picture because new data protection laws are coming into play and consumers are now more aware of how their privacy can be compromised. That said, how can the promise of releasing changes to the database faster be balanced with the need to keep data safe and remain compliant with legislation?

The answer lies in adopting a workflow where protecting data is baked into the four key stages of database development.

STANDARDIZED TEAM-BASED DEVELOPMENT

Version controlling database code is a good way to prevent conflicts and maintain one source of truth during development. Teams should also consider introducing consistent coding styles and using static code analysis tools so that the code is easier to understand and there is a quality control gate at the point new code is written.

AUTOMATED DEPLOYMENTS

As well as providing an audit trail of changes to demonstrate compliance, version control also opens the doors to automating ongoing development. Continuous integration, for example, can trigger an automated build and test code changes as soon as they are checked in. When a build succeeds, an artifact can also be generated for use with a release management tool.

PERFORMANCE AND AVAILABILITY MONITORING

If databases are updated more often, performance monitoring becomes crucial in highlighting deployment problems, particularly if databases are under heavy load. With data protection regulations requiring organizations to monitor and manage access to personal data and report any breaches, the availability of databases should also be monitored to provide alerts to potential issues.

PROTECTING AND PRESERVING DATA

The final step in compliant database DevOps is to protect personal data during development. Most developers like to work with a copy of the production database to test their changes against, yet those same databases invariably contain sensitive data. No surprise then that Gartner's 2018 Market Guide for Data Masking predicts the percentage of companies using data masking or practices like it will increase from 15% in 2017 to 40% in 2021.

SUMMARY

The rise of database DevOps is already underway, as shown by its inclusion in the Accelerate State of DevOps Report. Conversely, perhaps, the automation which DevOps encourages and the audit trails it provides across the development process ease compliance so that companies can deliver value faster while keeping data safe.

WRITTEN BY MARY ROBBINS
PRODUCT MARKETING MANAGER, DEVOPS, REDGATE

Minding the Database: Devs vs. DBAs

BY SANJAY CHALLA
KEY PRODUCT EVANGELIST, **DATICAL**

QUICK VIEW

01. Database development is increasingly done by application developers, not just DBAs.

02. In addition, there is demand for faster database change releases to support faster software delivery.

03. New tools that meet the needs of both DBAs and developers are required for organizations to successfully transform and modernize software delivery.

There has been a lot of attention and focus of late on cloud platforms, containers, and microservices. Fueled by the hype and promises, software teams of all sizes and shapes are looking to take advantage of these new methods of delivering software. However, many of these teams are conveniently omitting one of the most important elements of a functional end-user software experience: the underlying data and databases. At the end of the day, whether using containerized microservices or not, the application is meaningless without data. The dirty detail that nobody seems to talk about is how to manage and evolve the persistence layer so that it can support an application layer architected to leverage the most exciting serverless cloud platform or coolest new microservices solution.

To add further intrigue to this glaring omission, there has been a major shift in the industry wherein *application* developers are responsible for initially authoring database changes. If the term "application developer" is confusing, the trend can be put another way: the responsibility of writing the first draft of database code has shifted away from database professionals such as DBAs and data architects. That immediately begets the question: what does this mean for the database?

There has always been some friction between application teams and infrastructure teams — or put in a more familiar way — between Dev and Ops. Application teams generate change and constantly need the infrastructure to keep up. Ops teams, on the other hand, want to keep systems stable and available — and change can be risky for these groups. As expected, the shifting trend — where application developers author the database code needed to support the application feature or enhancement — adds to the tension that has long existed between Dev and Ops teams.

To be clear, the changes that have fallen to generalist developers are simply the changes needed to directly support an update to the application. The broader data architecture efforts, query performance tuning, security patches, system upgrades, and other work outside of directly supporting a change in the application are still firmly in the purview of DBAs and data architects. Fundamentally though, the challenge lies in supporting the opposing needs of those looking to make small, incremental changes to the structure and logic of a database and those looking to maintain the uptime, performance, and availability of a database.

Historically, the tools for making changes to and maintaining the database were focused on meeting the needs of DBAs as they were primarily responsible for both functions. Given the glacial pace of change relative to today, these tools were optimized more for operations than for development. The primary focus was on keeping databases up, running, and performant. As software development and delivery cycles have greatly accelerated, these tools have struggled to keep up. At the heart of the matter is that these tools were not designed for rapid, iterative, incremental changes — much less *parallel development* with feature or trunk-based approaches.

To address this gap, a new class of tools, optimized for rapid, iterative development across parallel workflows are needed to support the growing legions for application developers that are now authoring database changes have emerged. A lexicon of terms has quickly formed to describe the two different approaches that now exist. Tools that are better suited for understanding the overall state of the database, primarily meant for operations teams, data architects, and DBAs are often described as "state-based" or "declarative" tools. Meanwhile, tools that are better suited to making small changes to a database, primarily meant for developers needing to modify the database, are described as "migration-based" or "imperative" tools.

Both out-of-box vendor tools — for example, Microsoft's DACPAC, SSMS, and SSDT tools — and old, established proprietary tools — such as Quest Software's TOAD — are state-based, focusing on the ideal state definition and using a comparison (or "diff") to generate a migration script for each target database that needs to be evolved. Meanwhile, a lot of relatively newer tools, which are focused enabling Agile development teams track and migrate databases — such as Datical and Liquibase — are migration-based.

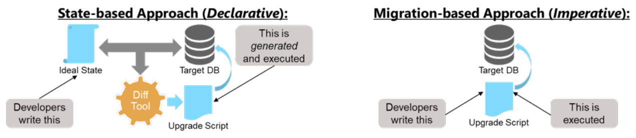


Figure 1: An illustration of how state-based and migration-based approach evolving the current state of a database.

A natural follow-up at this point is: what is the best approach — or better, what's the best tool? This depends heavily on the needs of the team. If working with a data-heavy application in an Agile framework, with a steady stream of database changes that are a part of every application sprint, it makes sense to adopt a migration-based tool. Conversely, if database changes are rare, many of the free, vendor-provided state-based tools may suffice. Veterans like Martin Fowler have addressed this topic extensively and strongly promote migration-based approaches to Agile development teams that need to rapidly and incrementally make changes to databases.

Why is the migration-based approach better suited to modern Agile and DevOps focused teams? It's because migration-based approaches better map to the key principles of Agile and DevOps workflows. In the name of accelerating velocity, improving quality, and ensuring transparency, these modern software workflows espouse the following:

- Small, incremental changes that are directly tied to business requirements
- Automated validation, build, and deployment of code changes
- Enabling self-service and fast feedback loops
- Repeatable, consistent deployments that are done with immutable artifacts ("build once, deploy often" is the mantra)
- Planning for and accommodating the flexibility to change requirements or implementation midway through a sprint

To begin, the state-based approach does not promote small, incremental changes. Developers work with and are consequently empowered to edit the entire state definition — which does not reinforce making small changes. Even if developers are disciplined and make small changes to the state definition, the generated migration script is often hundreds of lines long even for relatively simple changes. Worse, the generated script may also need to be edited in many cases to prevent data loss or other undesirable outcomes — especially as many changes cannot easily be derived from a comparison (for example: changing a relationship from one-to-many to many-to-many, or renaming a column). With all the manual intervention required, the state-based approach does not easily allow for developer self-service nor fast feedback loops.

Making matters worse, the state-based approach cannot adhere to the "build once, deploy often" best-practice as each deployment involves a comparison and script generation step that might yield different results. This adds risk to database deployments, and prevents deployments from being automated — as each generated script requires manual inspection — and potentially manual modification — prior to deployment. Lastly, teams lack the flexibility with this approach to accommodate changes. If there is a change in requirements or an issue discovered in testing, a developer needs to make a change to the state model, get a new migration script generated from the tool, review and possibly hand-edit the migration, and only then proceed forward.

As an alternative, the migration-based approach puts the burden on developers to focus on the migration. Once the migration itself has been explicitly defined, it's possible for intelligent automation to validate the small, incremental change, build it into an immutable artifact, and enable consistent, automated downstream deployments. By treating the migration as the first-class object of interest, it becomes possible to enable developer self-service, fast feedback loops, and the flexibility to quickly rework an individual migration instead of contending with the entire state model and a script generation process as demanded by the state-based approach.

Commercial solutions like Datical build upon the migration-based approach. For example, Datical's simulation capability allows teams to take a validated artifact of database changes and forecast the impact on a given target database. This type of rich, context-aware feedback can be automated, allowing better developer self-service and improving database code quality.

In summary, the needs of operators and developers are different, and as developers increasingly take on database code changes, it's important for organizations to use the proper tools and processes to reduce friction while improving throughput and quality. This doesn't meant that teams need to discard the old state-based tools; instead, it means that teams looking to bring Agile and DevOps practices to their database code changes need to invest in a migration-based solution to better enable developers while allowing operators to continue relying on state-based tools to manage system uptime.

SANJAY CHALLA With years of experience in product management and marketing, Sanjay understands software engineering tools and processes. At Datical, Sanjay serves as a key product evangelist, helping tell the Datical story across audiences - from executives to developers. Prior to Datical, Sanjay was the Director of Product at Hypori. Sanjay holds a Bachelor of Science degree from Georgia Tech.

Building Enterprise Performance Into a Graph Database

BY JO STICHBURY
FREELANCE TECHNICAL WRITER

QUICK VIEW

01. Graph databases are an increasingly popular choice of NoSQL database, with use cases that include recommendation engines, fraud detection, rules engines, text processing and text analytics in media and publishing, and discovery projects.

02. Graph DBs are not new, but uptake has been affected by some disadvantages, as first-generation systems have struggled to maintain high performance on queries and been difficult to scale.

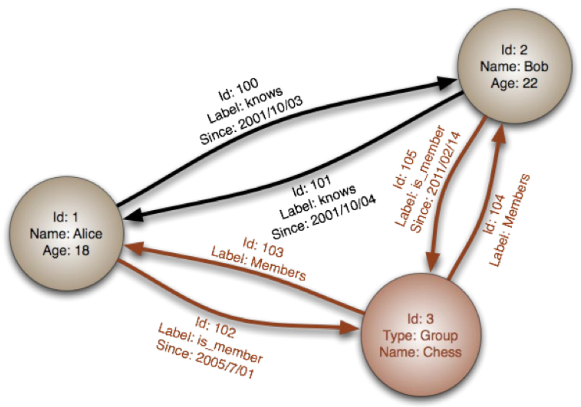
03. A second-generation graph database, Memgraph, is used as an example of the approaches used to resolve issues perceived to be a block to graph database uptake, focusing on speed, scalability, simplicity, and security to ensure suitability for enterprise usage.

This article introduces some of the key concepts of graph databases and their use cases. It explains how a focus on speed and scalability allows a modern "second-generation" graph database to tackle enterprise-level use cases to enable delivery of high-performance machine learning.

INTRODUCTION

The NoSQL ("not only SQL") space presents a number of different data models and database systems which can be more suitable than relational systems for particular types of data and use cases. Among these are graph databases, in which data is formed of a collection of nodes, the edges that connect them, and a set of properties. Nodes are entities, such as people or items, edges represent the relationships between the nodes, and properties are information about the nodes and edges.

The underlying data model of common graph databases is either an RDF triple store or a labelled property graph, and you can find out more in an article here on DZone, which describes the differences between the two approaches. RDF databases and property graphs might be native products, but they can also be built on top of other database types, such as a NoSQL database like Hadoop or Cassandra.



A property graph, illustrating nodes, properties, and edges from Wikimedia Commons [CC0].

GRAPH DATABASE USE CASE

Interconnection is the key differentiator for graph databases. The edges represent relationships more directly than other database systems and can reveal patterns about the connections between data without the use of foreign keys or MapReduce. Graph databases are excellent for navigating a relationship structure quickly — for example, the number of steps needed to get from one node to another.

Wherever you use a relational database, you can use a graph database; to get the most from it, you need to structure your data to take advantage of the relationships. Typical use cases include recommendation engines, fraud detection, rules engines, text processing and text analytics in media and publishing, and discovery projects that explore a “data lake” to uncover previously unseen relationships between entities — an approach which has applications in security and research. A great example is the recent use of a graph database by the International Consortium of Investigative Journalists (ICIJ). Their analysis of the “Panama Papers” revealed highly interconnected networks of offshore tax structures. This latter use case is sometimes referred to as cognitive computing since it uses data mining and pattern recognition.

Graph databases are excellent for navigating a relationship structure quickly — for example, the number of steps needed to get from one node to another.

In the current age of interconnected data, graph databases are showing increased uptake within large online systems and have become the fastest growing category of NoSQL databases. A recent report from IBM surveyed over 1,300 entrepreneurs and developers worldwide about their current and planned use for graph databases, and 43% of respondents were reported to be using or planning to use graph technology. New graph databases are emerging, and existing vendors are adding graph capabilities to their current solutions. Some of the key players include Neo4J, TitanDB, Blazegraph, HypergraphDB, OrientDB, IBM Graph, and Amazon’s Neptune.

HISTORICAL DISADVANTAGES OF GRAPH DATABASES

Neither graph computing as a field nor graph databases as a technology are particularly new; both have been around for at least 30 years. Given the exciting possibilities that they offer, you might wonder why they have not been more widely adopted before now. Early adopters of the first generation of graph databases have reported some issues, which include the following:

- Some disk-based graph databases are unable to handle a constant stream of updates and stay sufficiently responsive to provide high performance on queries.
- Not all traditional graph databases are suitable for transactions and some are more focused on providing analytical frameworks.
- It is difficult to determine how to shard or partition data to get best performance out of a graph database system as it scales.
- It can be difficult to know how to model domain data onto a graph, since it requires a specialist skill set — that is, proficiency in defining an ontology.
- When the data model is defined, it is essential that data loaded into the graph database is consistent with that ontology. Graph databases, like other NoSQL databases, delegate adherence to a schema to the application system and do not enforce a schema, so the data entering the database needs guaranteed consistency up front.

INTRODUCING MEMGRAPH

In the rest of this article, I am going to look how the next generation of graph databases is tackling some of the drawbacks of the trailblazing, early graph database systems. I will use Memgraph as an example because I have access to the team for technical support, and I would also encourage you to reach out to them if you have questions. But, as ever, I should point out that other graph databases are available. Memgraph is a next-generation graph database system which sits in-memory and has been optimized for real-time transactional use cases, aiming to handle highly concurrent operational and analytical workloads within the enterprise space.

The Memgraph architecture sets out to eliminate the issues leveled at earlier generation graph databases by prioritizing “four pillars:” speed, scale, simplicity, and security. Let’s look at the first two of these in some more detail.

SPEED

Memgraph supports real-time use cases for enterprises with colossal amounts of data. The team has achieved high performance benchmarks, demonstrating low latency and high throughput on high volumes of read and write queries (with ACID transactions) on real-world scale datasets. For reasons of space and timing, I can't share the benchmark results here, but they will be published within the next few weeks, and I can say that they stack up well when compared against incumbent graph database systems.

The Memgraph architecture sets out to eliminate the issues leveled at earlier generation graph databases by prioritizing "four pillars:" speed, scale, simplicity, and security.

Memgraph is capable of loading tens of thousands of nodes/edges per second on a single machine. The database achieves high throughput by using highly concurrent (and sometimes even lock-free) data structures and multi-version concurrency control (MVCC). This ensures that writes never block reads and vice versa. Traditional databases manage concurrency with global locks, which results in some processes blocking others until they complete and release the lock. Memgraph's implementation of MVCC provides snapshot isolation level which allows better performance than serializability, while avoiding most of the concurrency anomalies.

Additionally, Memgraph uses a highly concurrent skip list for indexing purposes. Skip lists represent an efficient technique for searching and manipulating data, delivering concurrency and performance benefits beyond those seen for other graph databases and disk-based databases which use B-Trees to store indexes.

SCALE

On a single machine, Memgraph scales up to the size of a main memory or disk space where properties could be stored on disk. In a distributed system, the graph is automatically repartitioned in the background to improve query execution time and scalability.

Memgraph features a distributed query planning and execution engine. Each plan is divided into two; a plan that will be executed on the machine where the query is received and a plan that will be executed on the other machines. Nodes are allowed to exchange data during the execution process to maximize performance.

In its distributed environment, Memgraph offers a custom and high-performance implementation of common graph algorithms such as level-synchronous parallel breadth-first search (BFS), which in many cases outperforms a single machine BFS.

To improve query performance and scalability, Memgraph runs a dynamic graph partitioning algorithm in the background to minimize the number of crossing edges between machines whilst keeping the cluster optimally balanced.

AI AND ANALYTICS

Memgraph can integrate with the most popular machine learning tool such as TensorFlow and PyTorch, readily allowing the stored data to be used to train a model. To facilitate training and production deployment across the enterprise space, the Memgraph client will be wrapped in a TensorFlow operation to allow direct querying into Memgraph. A similar approach will be used for PyTorch and ONNX deep learning, with the goal of making the data within Memgraph easily accessible to data scientists — for example, those developing systems for fraud detection in finance and retail.

Thanks to its in-memory architecture, Memgraph is also suitable for running analytical workloads. Algorithms like BFS and Weighted Shortest Path constitute the core of the query execution stack.

SUMMARY

Graph databases are increasing in popularity, and though the next-generation graph technology is still very much under development, it promises to eliminate some of the current disadvantages and deliver a powerful backend for use in enterprise and beyond.

If you have specific questions about Memgraph, you may find them answered by their FAQ, but you can find out more by contacting the team.

JO STICHBURY is a freelance technical writer with over 20 years' experience in the software industry, including 8 years of low-level mobile development. Jo typically writes about machine intelligence, high performance computing, electric and driverless vehicles, and the future of transport. She holds an MA and a PhD in Natural Sciences from the University of Cambridge.

diving deeper

INTO **DATABASE**

zones

Database dzone.com/database
The Database Zone is DZone's portal for following the news and trends of the database ecosystems, which include relational (SQL) and nonrelational (NoSQL) solutions such as MySQL, PostgreSQL, SQL Server, NuoDB, Neo4j, MongoDB, CouchDB, Cassandra and many others.

Big Data dzone.com/big-data
The Big Data/Analytics Zone is a prime resource and community for Big Data professionals of all types. We're on top of all the best tips and news for Hadoop, R, and data visualization technologies. Not only that, but we also give you advice from data science experts on how to understand and present that data.

Cloud dzone.com/cloud
The Cloud Zone covers the host of providers and utilities that make cloud computing possible and push the limits (and savings) with which we can deploy, store, and host applications in a flexible, elastic manner. The Cloud Zone focuses on PaaS, infrastructures, security, scalability, and hosting servers.

twitter

@SQLEspresso
@BrentO
@KimberlyLTripp
@victoria_holt
@mmarie
@GFritchey
@pinaldave
@sqlgirl
@wendy_dance
@catherinew

refcardz

Data Warehousing
From descriptions to diagrams and integration patterns, this newly updated Refcard walks you through each aspect of data warehousing. Gain a complete understanding of data modeling, infrastructure, relationships, attributes, and speedy history loading and recording with atomic data.

DevOps for Database
Download this new Refcard to get started with Database Release Automation and eliminate bottlenecks. Learn the key best practices that your DevOps database solution should meet in order for you to get the most out of your investment.

Understanding Data Quality
This Refcard will show you the key places data derives from, characteristics of high-quality data, and the five phases of a data quality strategy that you can follow.

podcasts

SQL Data Partners Podcast
Learn about new and familiar topics in the worlds of SQL Server and professional development.

The NoSQL Database Podcast
Listen to interviews with top developers and database leaders to learn everything you need to know about NoSQL databases.

The Voice of the DBA
Learn about databases from the perspective of a data professional.

courses

Data Wrangling With MongoDB
Learn about wrangling data from diverse sources and shape it to enable data-driven applications.

Intro to Relational Databases
See how to write code using a database as a backend to store application data both reliably and safely.

SQL for Data Analysis
Learn how to use SQL to extract data, perform aggregations, manipulate subqueries, write efficient SQL queries, and more.

BUILDING THE SOLUTIONS
THAT MOVE US FORWARD
MATTERS.

Our software helps health professionals deliver care, businesses prosper, and governments serve their citizens. With solutions that are more reliable, intuitive, and scalable than any other, we drive the world's most important applications and help pave the way to a brighter tomorrow.

Learn more at InterSystems.com

The power behind what matters.

InterSystems®
Health | Business | Government

Streamline the Path From Big Data to Value Creation

Enterprises creating the next generation of applications need to consider the most efficient path to business value. In our services-based economy, customer experiences matter most. Business value comes from developing applications that create the ultimate experience.

The ultimate experience, depending on your industry, requires knowing as much as you can about your customers, your patients, and your citizens. Having that knowledge when they are about to make a decision that matters is critical in cementing positive impressions that will last. Weaving together multiple data sources with unique storage systems and processing platforms required can slow application development and create scalability challenges. It also complicates governance, risk management, and compliance. If developers are not building applications that leverage all possible sources of information, Big Data, and real-time analytics to create experiences that can be quickly adapted to new engagement models, they will fall short of their targets. Enterprises need to consider using a comprehensive, consolidated data platform to build transformational applications. The alternative — deploying multiple disparate technologies and systems — adds complexity and application development costs.

InterSystems has been delivering reliable, scalable, and innovative technology to thousands of customers for more than 30 years. The InterSystems Data Platform reduces complexity without sacrificing functionality, performance, or flexibility. By delivering world-class customer support, we focus on our customers' needs so they can focus on building applications that matter – elevating the bar for the ultimate in customer experience.

WRITTEN BY JULIE LOCKNER
DIRECTOR, PRODUCT MARKETING AND PARTNER PROGRAMS, INTERSYSTEMS

DZONE.COM/GUIDES

PARTNER SPOTLIGHT

InterSystem Caché

"With Caché, we obtain considerably superior performance and scalability than is possible with other databases." - *William o'mullane, scientific operations manager of the Gaia Mission, European Space*

RELEASE SCHEDULE

Major release annually, 2 minor releases annually

OPEN SOURCE?

No

CASE STUDY

The European Space Agency (ESA) launched an ambitious mission to chart a three-dimensional map of the Milky Way. Because an enormous amount of data will need to be quickly stored and

analyzed, ESA has selected Caché as the advanced database technology to support the scientific processing of the Gaia mission.

Gaia will spend five years monitoring a billion stars in our galaxy. In the course of its operation, AGIS must be able to insert up to 50 billion Java objects into a database within seven days. Caché

is the only database ESA found that could provide the necessary performance and scalability with only moderate hardware requirements. The information Gaia collects will allow scientists to learn a great deal about the origin, structure, and evolutionary history of our galaxy.

PRODUCT

InterSystem Caché is the industry's only multi-workload, multi-model NoSQL and relational database, with embedded data and application interoperability via Ensemble, built-in structured and unstructured data analytics, and a comprehensive rapid application development environment without sacrificing high performance, scalability, reliability, and security.

STRENGTH

Complete data platform for reliable, complex applications

NOTABLE CUSTOMERS

- Ontario Systems
- MFS
- TD Ameritrade
- ESA
- Netsmart
- WS Trends

WEBSITE intersystems.com **TWITTER** @InterSystems **BLOG** intersystems.com/blog

Executive Insights on the Current and Future State of Databases

BY TOM SMITH
RESEARCH ANALYST AT **DEVADA**

QUICK VIEW

1. The keys to a successful database strategy are understanding the business needs the database needs to fulfill, followed by the quartet of availability, scalability, performance, and security.

2. The biggest changes in the database ecosystem have been the cloud and the proliferation of databases.

3. Organizations are pursuing a polyglot, hybrid, multi-model, cloud strategy, and using the best of breed databases available to get the job done.

To gather insights on the current and future state of the database ecosystem, we talked to IT executives from 22 companies about how their clients are using databases today and how they see use, and solutions, changing in the future. Here's who we talked to:

- Jim Manias, Vice President, Advanced Systems Concepts, Inc.
- Tony Petrossian, Director, Engineering, Amazon Web Services
- Dan Potter, V.P. Product Management and Marketing, Attunity
- Ravi Mayuram, SVP of Engineering and CTO, Couchbase
- Patrick McFadin, V.P. Developer Relations, DataStax
- Sanjay Challa, Senior Product Marketing Manager, Datical
- Matthew Yeh, Director of Product Marketing, Delphix
- OJ Ngo, CTO, DH2i
- Navdeep Sidhu, Head of Product Marketing, InfluxData
- Ben Bromhead, CTO and Co-founder, Instaclustr
- Jeff Fried, Director of Product Management, InterSystems
- Dipti Borkar, Vice President Product Marketing, Kinetica
- Jack Norris, V.P. Data and Applications, MapR
- Will Shulman, CEO, mLab
- Philip Rathle, V.P. of Products, Neo4j
- Ariff Kasam, V.P. Products, , NuoDB
- Josh Verrill, CMO, NuoDB
- Simon Galbraith, CEO and Co-founder, Redgate Software
- David Leichner, CMO, SQream
- Arnon Shimoni, Product Marketing Manager, SQream
- Todd Blashka, COO , TigerGraph
- Victor Lee, Director of Product Management, TigerGraph
- Mike Freedman, CTO and Co-founder, TimescaleDB
- Ajay Kulkarni, CEO and Co-founder, TimescaleDB
- Chai Bhat, Director of Product Marketing, VoltDB
- Neil Barton, CTO, WhereScape

1. The keys to a successful database strategy are **understanding the business needs the database needs to fulfill, followed by the quartet of availability, scalability, performance, and security.** You need to understand the precise aspects of the work load so you can choose the right database. Determine the database to use on a case-by-case basis – what are you doing, where are you based, what are you trying to achieve, and what are you using in your infrastructure? Understand how the use case ties back to the business and the business outcomes you are trying to achieve. You have to understand the functional and non-functional requirements of the business.

You have to take advantage of the elasticity of cloud platforms. Availability and scalability are key to 80% of database customers. Ideally, all of the components work in concert to provide a safe, available, and high-performance database solution.

2. There were nearly as many opinions of how companies can get a handle on the vast amounts of data they're collecting as people I spoke with. **aloguing, automation, indexing, building for scale, and Hadoop** were the suggestions mentioned more than once.

Data cataloguing and automation tools using metadata are encouraged in order to know where data is and how it's being processed. The exponential growth of data, along with the increasing sprawl of data as it spreads around companies, is making the cataloging and identification of data an important issue for every company. Companies need to adopt an automation solution with capabilities that support big data automation to solve two major pain points: integration and scheduling.

Build an index that can be queried quickly. Data is normalized for storage and quick access by indexing and by writing good queries for ingesting and accessing the data. A core competency for any database is how it scales – if you need more capacity you can just add more nodes.

The collection side problem is solved with Hadoop file systems. Move to Hadoop on premise or cloud and help reassemble a subset of the data as needed for analytics.

3. The biggest changes in the database ecosystem have been the **cloud and the proliferation of databases.** These changes are being impacted by digital transformation, including the cloud and containers, which can provide built-in high availability, automated patching, dynamic scalability, and backup management with point-in-time recovery. Databases are being deployed within containers for speed, agility, scalability, ease of deployment, use, configuration, and installation. There's a war between big cloud vendors on the scale and capabilities of their respective database platforms.

We're seeing a lot of new databases that are specialized for particular use cases. Today it's a polyglot environment with many companies employing several models with the ability to do different types of processing in the same database. We're seeing NoSQL going back to SQL, as well as the emergence of document and graph databases based on business needs.

4. While there were **more mentions of graph** than others when asked about adoption of databases, its clear **organizations are pursuing a polyglot, hybrid, multi-model, cloud strategy using the best tool to get the job done.** Graph and document databases are growing faster than SQL and relational; however, SQL and relational will always be there. Respondents saw high rates of adoption of cloud platforms because they scale up and down as needed and are much more cost effective.

5. There were **more than a dozen applications and another dozen industries** in which databases are being used to solve business problems. The most frequently mentioned application was improving the customer experience by offering application personalization and recommendations. The most frequently mentioned verticals were financial services and retail.

In one example a respondent shared, a financial services firm has applications that rely on a core set of databases with different teams working on different applications needing access to the same databases. Any database changes affect applications and thus need to be tested. An ephemeral test production system was implemented without a large scale to keep speed up and cost down. This accelerated the test process and the client was able to test faster with greater fidelity.

A real-time application of a retail company predictively and actively engages customers with highly personalized experiences with tools that can collect, explore, analyze, and act on multiple streams of data instantaneously. These tools allow retail businesses to make data-driven decisions to improve personalization, recommendations, and customer experience.

6. The most common issues companies are having with databases are: **1) data management; 2) database sprawl; and 3) choosing the right database to solve the business problem.**

Loading data into the system, preparation of the data, and lack of proper indexing are all issues for clients. Design mistakes and data quality issues are more complex, although the wrong database can increase the likelihood of mistakes and data quality harder to maintain.

"Database sprawl" has continued to be one of the biggest issues affecting companies today. As applications continue to evolve, their requirements have led to a growing number of point solutions at the data layer. The ability to coordinate disparate technologies in diverse and intricate conditions with the lack of a single point of control is challenging. You need someone who knows custom scripting.

Another common issue is users who are forcing the database engine to do something it's not designed to do. Every database has a fit and a purpose or use case. It's important to understand that use case and then use the appropriate database technology.

7. The biggest opportunities in the evolution of databases are **integration and using AI/ML to provide real-time analytics.** Given the number of databases, integrating multiple technologies will become more important. We're seeing the beginning with JSON, Python, and R. Databases are taking ML into production and deploying at large scale. While ML models require considerable time and expense, the value they provide is tremendous.

8. The biggest concerns around databases today are **the proliferation of databases and keeping the data in those databases secure**. It's hard to track all of the new databases that come out every week (DB-Engines currently tracks 343). It's challenging to decide on a technology that will be around for the long haul. Data infrastructure is becoming more fragmented. Coordinating across databases and siloes is very difficult. There is a lot of confusion around which database to use to solve a particular problem. DB-Engines is a good resource, but organizations can't find the talent to take advantage of all of the new technologies.

Unfortunately, security is still an afterthought. With GDPR and the forthcoming California privacy regulations, security capabilities are becoming more important – including encryption, access, auditing AI models, and auditing test and training data. If we don't improve data security, it will erase any gains made through new technology advances.

9. Developers do not need advanced database knowledge or skills as much as they **need to know SQL, understand the different kinds of databases, and which are the most viable in each category.** SQL is still the most useful and used language, so it's still important to understand it and how to write scripts. Try the different database technologies, spin up a cloud and begin playing in minutes. Get hands on with different databases to determine the best fit for your needs. As you get more involved with a particular database, you will learn how to optimize it to solve your business problem.

TOM SMITH is a Research Analyst at DZone who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.

MariaDB

Enterprise reliability.

Open source innovation.

Download MariaDB today to to try the latest features in MariaDB TX.

Mariadb.com/downloads

The Return of Relational (with JSON)

There was a time when developers and architects had to choose between relational and non-relational databases. While relational databases provide strong consistency, non-relational databases deliver flexibility and scalability. However, the line between relational and non-relational databases is blurring as consistency and flexibility/scalability are no longer mutually exclusive.

While leading non-relational databases have yet to add transactions and schema constraints, a number of relational databases have added support for semi-structured data. MariaDB Server 5.3 introduced dynamic columns, enabling different rows to have different columns. MariaDB Server 10.2 introduced a comprehensive set of SQL functions for reading, writing and querying JSON documents stored in columns.

MariaDB Server provides true flexibility, the ability to support both structured and semi-structured data with the same database. The data can be structured or semi-structured, or it can be partially structured and partially semi-structured. While structured data can be modeled as columns, semi-structured data can be modeled as JSON documents. For example, relational models for fixed data can be extended with JSON documents to support-variable data.

In addition, MariaDB Server includes a distributed storage engine (Spider) for read, write and storage scalability, distributing native partitions across multiple database servers. With JSON functions and Spider, MariaDB Server provides consistency, flexibility and scalability. MariaDB TX, built on MariaDB Server, includes the world's most advanced database proxy (MariaDB MaxScale) and administrative tools for managing MariaDB Server deployments.

WRITTEN BY SHANE JOHNSON
SENIOR DIRECTOR OF PRODUCT MARKETING, MARIADB CORPORATION

PARTNER SPOTLIGHT

MariaDB TX

MariaDB

Born of the community. Raised in the enterprise. It's the result of engineering leadership and community innovation – a modern database for everyone.

RELEASE SCHEDULE	OPEN SOURCE?	PRODUCT
MariaDB TX 3.0, May 2018	Yes	DBMS

CASE STUDY

Red Hat developed Red Hat Single Sign-On (based on Keycloak), and chose to run it on MariaDB TX because they needed scalability and high availability. The solution required a resilient architecture based on hybrid cloud infrastructure to prevent user sessions from being interrupted in the event of a major outage. MariaDB TX, with synchronous, multi-master replication (MariaDB Cluster), enabled Red Hat to create a multi-site, hybrid-cloud solution by deploying a database cluster with nodes running in the datacenter and on public cloud infrastructure. With a full open source stack from the OS to the database to the middleware, Red Hat created a single sign-on solution to meet modern scalability and high availability demands.

STRENGTH

- Supports JSON and GeoJSON with a comprehensive set of SQL functions
- Includes the world's most advanced database proxy, MariaDB MaxScale
- Features best-in-class security with a built-in firewall and data masking
- Provides complete HA/DR with replication, clustering and restore/rollback
- Includes advanced SQL: common table expressions and window functions

NOTABLE CUSTOMERS

- DBS Bank
- Deutsche Bank
- Red Hat
- Verizon
- O2

WEBSITE mariadb.com **TWITTER** @mariadb **BLOG** mariadb.com/resources/blog

Solutions Directory

This directory contains databases and database performance tools to help you store, organize, and query the data you need. It provides free trial data and product category information gathered from vendor websites and project pages. Solutions are selected for inclusion based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Actian	Actian X	RDBMS	30 days	actian.com
Actian	Actian NoSQL	Object-oriented	30 days	actian.com/data-management/nosql-object-database
ActiveRecord	ActiveRecord	ORM	Included in Rails	guides.rubyonrails.org/active_record_basics.html
Aerospike	Aerospike Server	In-memory, KV	Open source	aerospike.com/download/server/3.15.0.1
Altibase	Altibase HDB	In-memory, NewSQL	Open-source version available	altibase.com
Amazon Web Services	DynamoDB	KV, document, DBaaS	Free tier available	aws.amazon.com/dynamodb
Amazon Web Services	SimpleDB	Column store	Free tier available	aws.amazon.com/simpledb
Apache Foundation	Apache Cassandra	KV, wide column	Open source	cassandra.apache.org

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Apache Foundation	Apache Hbase	Wide column	Open source	hbase.apache.org
Apache Foundation	Apache Ignite	In-memory, Hadoop, data grid	Open source	ignite.apache.org
Apache Foundation	Apache OpenJPA	ORM	Open source	openjpa.apache.org
Apple	Core Data	ORM	Included in iOS & macOS	developer.apple.com/documentation/coredata
ArangoDB	ArangoDB	Graph, document, KV	Open source	arangodb.com
Atlassian	ActiveObjects	ORM	Included in Jira & Confluence	developer.atlassian.com/docs/atlassian-platform-common-components/active-objects
Basho	Riak	Document, KV	Open source	basho.com/products
CakeDC	CakePHP	ORM	Open source	cakephp.org
Canonical	Storm	ORM	Open source	storm.canonical.com
Couchbase	Couchbase Server	KV, document, data caching	Open source	couchbase.com
CUBRID	CUBRID	RDBMS	Open source	cubrid.org
Datical	Datical	Application release automation for Databases	Demo available by request	datical.com
DBmaestro	Database Release Automation	Continuous delivery for databases	14-day free trial	dbmaestro.com/products/database-release-automation
Delphix	Delphix	Dynamic data plaform	Demo available by request	delphix.com
Eclipse Foundation	EclipseLink	ORM	Open source	eclipse.org/eclipselink
Embarcadero	InterBase	RDBMS	Available by request	embarcadero.com/products/interbase

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
EnterpriseDB	EDB Postgres Platform	RDBMS	Open source	enterprisedb.com/products/edb-postgres-platform
FairCom	c-treeACE	NewSQL, KV direct access	N/A	faircom.com/products/c-treeace
Firebird	Firebird	RDBMS	Open source	firebirdsql.org
Google	BigTable	Column store	$300 credit for 12 months	cloud.google.com/bigtable
Google	Cloud Spaner	RDBMS, NewSQL	$300 credit for 12 months	cloud.google.com/spanner
Gridgain	GridGain In-Memory Computing	In-memory, Hadoop, data grid	30 days	gridgain.com/products/gridgain-products
Hazelcast IMDG	Hazelcast	In-memory, data grid	Open source	hazelcast.com/products
Hibernating Rhinos	RavenDB	Document	Open source	ravendb.net
IBM	IBM DB2	RDBMS	Available by request	ibm.com/analytics/us/en/db2
IBM	Informix	RDBMS, transactional	Available by request	ibm.com/analytics/us/en/technology/informix
InfluxData	InfluxEnterprise	Stream processing and analytics	Available by request	influxdata.com/products
InterSystems	Cache	Object-oriented, relational	N/A	intersystems.com/products/cache/#technology
JOOQ	JOOQ	ORM for Java	Open-source version available	jooq.org
MariaDB	ClustrixDB OLTP Database	NewSQL, transactional	Available by request	clustrix.com/oltp-database
MariaDB	MariaDB TX	RDBMS, document, MySQL family	Open source	mariadb.com/products/solutions/oltp-database-tx
MarkLogic	MarkLogic	Transactional	Free tier available	marklogic.com/product/marklogic-database-overview

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
MemSQL	MemSQL	In-memory, NewSQL	Available by request	memsql.com/product
Micro Focus	Vertica Enterprise	RDBMS, column store	Free tier available	vertica.com/product/on-premise
Microsoft	Entity Framework	ORM	Part of .NET framework	docs.microsoft.com/en-us/ef
Microsoft	SQL Server 2019	RDBMS	180-day preview available	microsoft.com/en-us/sql-server/default.aspx
MongoDB	MongoDB	Document	Open source	mongodb.com
MyBatis	MyBatis	ORM	Open source	blog.mybatis.org/p/products.html
Neo4j	Neo4j	Graph	Free tier available	neo4j.com
Nhibernate	Nhibernate	ORM for .NET	Open source	nhibernate.info
NuoDB	NuoDB	NewSQL	Free tier available	nuodb.com
OpenText	OpenText Gupta SQLBase	RDBMS	Available by request	opentext.com/what-we-do/products/specialty-technologies/opentext-gupta-development-tools-databases/opentext-gupta-sqlbase
Oracle	JDBC	ORM	Free solution	oracle.com/technetwork/database/application-development/jdbc/downloads/index.html
Oracle	MySQL Community Edition	RDBMS	Open source	mysql.com
Oracle	Oracle Database	RDBMS, document	Free solution	oracle.com/database/index.html
Oracle	Toplink	ORM	Free solution	oracle.com/technetwork/middleware/toplink/overview/index.html
OrientDB	OrientDB	Document, graph, KV	Open source	orientechnologies.com
OrmLite	OrmLite	ORM	Open source	ormlite.com

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Percona	Percona Server for MySQL	RDBMS, MySQL family	Free solution	percona.com/software/mysql-database/percona-server
Pivotal	Gemfire	In-memory, data grid	Free solution	pivotal.io/pivotal-gemfire
PostgreSQL	PostgreSQL	RDBMS	Open source	postgresql.org
Red Gate Software	SQL Toolkit	CI, source control, change management for databases	14 days	red-gate.com/products
Red Hat	Hibernate	ORM	Open source	hibernate.org
Redis Labs	Redis	In-memory, KV, data caching	Open source	redis.io
Redis Labs	Redis Cloud	In-memory, KV, data caching, DBaaS	Free tier available	redislabs.com/redis-enterprise/cloud
SAP	SAP HANA Platform	In-memory, column-oriented RDBMS	Free tier available	sap.com/products/hana.html
ScaleOut	ScaleOut StateServer	In-memory, Hadoop, data grid	30 days	scaleoutsoftware.com/products
Software AG	Adabas	Stream processing	Available by request	adabas.com
Solarwinds	Database Performance Analyzer	Database performance tuning	14 days	solarwinds.com/database-performance-analyzer
Splice Machine	Splice Machine	NewSQL, RDBMS	Free tier available	splicemachine.com/product/features
SQLite	SQLite	RDBMS	Open source	sqlite.org
Studio 3T	Studio 3T	IDE for MongoDB	Free tier available	studio3t.com
SymmetricDS	SymmetricDS	Database replication	Open source	symmetricds.org
Teradata	Aster Database	Specialist analytic	Available by request	teradata.com
VoltDB	VoltDB	In-memory, NewSQL	30 days	voltdb.com

GLOSSARY

ACID (ATOMICITY, CONSISTENCY, ISOLATION, DURABILITY): A term that refers to the model properties of database transactions, traditionally used for SQL databases.

BASE (BASIC AVAILABILITY, SOFT STATE, EVENTUAL CONSISTENCY): A term that refers to the model properties of database transactions, specifically for NoSQL databases that need to manage unstructured data.

CONTINUOUS DEPLOYMENT: A software development practice in which every code change goes through the entire pipeline & is put into production automatically, resulting in many production deployments every day.

CONTINUOUS INTEGRATION: A software development process whereby a branch of source code is rebuilt every time code is committed to the source control system; the process is often extended to include deployment, installation, & testing of applications in production environments.

DATABASE ADMINISTRATOR (DBA): Someone who oversees successful database production in terms of security, maintenance, design, and implementation.

DATABASE MANAGEMENT SYSTEM (DBMS): A suite of software & tools that manage data between the end user & the database.

DATA WAREHOUSE: A collection of individual computers that work together & appear to function as a single system. This requires access to a central database, multiple copies of a database on each computer, or database partitions on each machine.

DEVELOPMENT CYCLE: The iteration of steps to take the development of a feature or product from conception to release.

DEVOPS: An IT organizational methodology where all teams in the organization, especially development teams & operations teams, collaborate on both the development & deployment of software to increase software production agility & achieve business goals.

DOCUMENT STORE: A type of database that aggregates data from documents rather than defined tables & is used to present document data in a searchable form.

FAULT TOLERANCE: A system's ability to respond to hardware or software failure without disrupting other systems.

GRAPH STORE: A type of database used for handling entities that have a large number of relationships, such as social graphs, tag systems, or any link-rich domain; it is also often used for routing & location services.

HADOOP: An Apache Software Foundation framework developed specifically for high scalability, data-intensive, distributed computing. It is used primarily for batch-processing large datasets very efficiently.

HIGH AVAILABILITY (HA): Refers to the continuous availability of resources in a computer system even after component failures occur. This can be achieved with redundant hardware, software solutions, & other specific strategies.

INDEX-FREE ADJACENCY: A way to efficiently process data in a graph with connected nodes that point to each other in a database.

IN-MEMORY: As a generalized industry term, it describes data management tools that load data into RAM or flash memory instead of hard-disk or solid-state drives.

KEY-VALUE STORE: A type of database that stores data in simple key-value pairs. They are used for handling lots of small, continuous, potentially volatile reads & writes.

KNOWLEDGE BASE: A type of database that often uses AI functionality and that stores complex data that is both structured & unstructured.

MICROSERVICES: A development method of designing your applications as modular services that seamlessly adapt to a highly scalable & dynamic environment.

NEWSQL: A shorthand descriptor for relational database systems that provide horizontal scalability & performance on par with NoSQL systems.

NOSQL: A class of database systems that incorporates other means of querying outside of traditional SQL and does not use standard relational structures.

OBJECT-RELATIONAL MAPPER (ORM): A tool that provides a database abstraction layer to convert data between incompatible type systems using object-oriented programming languages instead of the database's query language.

PERSISTENCE: Refers to information from a program that outlives the process that created it, meaning it won't be erased during a shutdown or clearing of RAM; databases provide persistence.

POLYGLOT PERSISTENCE: Refers to an organization's use of several different data storage technologies for different types of data.

RELATIONAL DATABASE: A database that structures interrelated datasets in tables, records, & columns.

REPLICATION: A term for the sharing of data in order to ensure consistency between redundant resources.

SCHEMA: A term for the unique data structure of an individual database.

SHARDING: Also known as "horizontal partitioning," sharding refers to the splitting of a database into several pieces, usually to improve the speed & reliability of an application.

STRONG CONSISTENCY: A database concept that refers to the inability to commit transactions that violate a database's rules for data validity.

STRUCTURED QUERY LANGUAGE (SQL): A programming language designed for managing & manipulating data; used primarily in relational databases.

TIME SERIES DATABASE: A database that is optimized to handle arrays of numbers indexed by time, otherwise known as time series data.

INTRODUCING THE

Open Source Zone

Start Contributing to OSS Communities and Discover Practical Use Cases for Open-Source Software

Whether you are transitioning from a closed to an open community or building an OSS project from the ground up, this Zone will help you solve real-world problems with open-source software.

Learn how to make your first OSS contribution, discover best practices for maintaining and securing your community, and explore the nitty-gritty licensing and legal aspects of OSS projects.

COMMITTERS & MAINTAINERS

COMMUNITY GUIDELINES

LICENSES & GOVERNANCE

TECHNICAL DEBT

Visit the Zone

BROUGHT TO YOU IN PARTNERSHIP WITH flexera