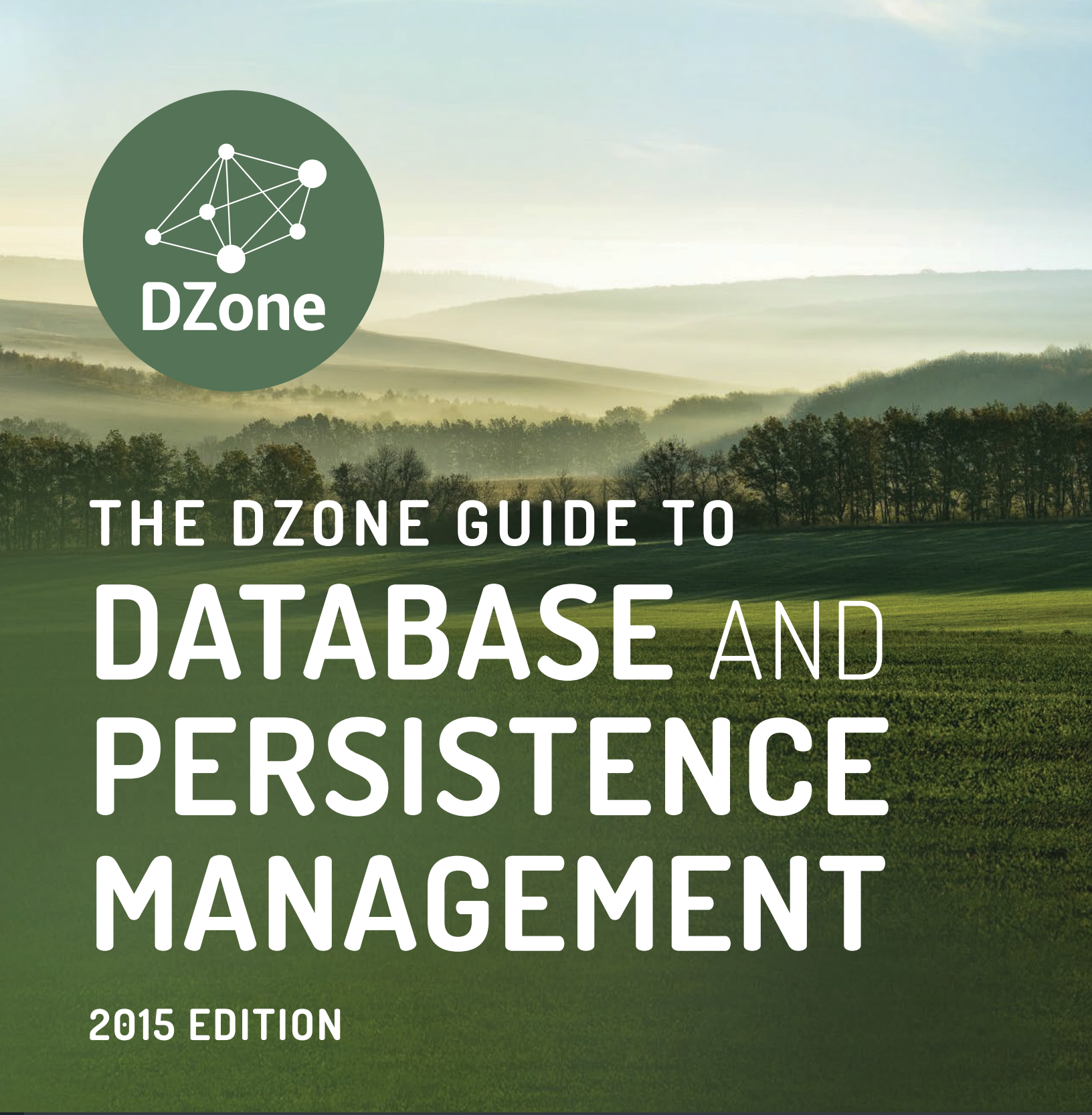


THE DZONE GUIDE TO DATABASE AND PERSISTENCE MANAGEMENT

2015 EDITION

BROUGHT TO YOU IN PARTNERSHIP WITH

Couchbase

MAMMOTH DATA

nuoDB®

Dear Reader,

For thousands of years, the closest things to databases that humans have had were libraries. Even in those days, the ancients knew how out of hand things could get if sorting standards changed or if texts weren't returned to the proper place. Today, we have databases that can access thousands of records in mere milliseconds. Data can be infinitely duplicated and preserved, unlike the ill-fated Library of Alexandria. And yet, errors in sorting, translation, and revision still happen even in our most advanced databases today. Nothing is perfect, but modern standards for data persistence, like SQL and relational schemas, have endured and kept a majority of the world's databases well-organized. Other styles of persistence have existed for many years, but only in the last few years have so many broken into mainstream use, causing a rapid, unprecedented expansion of database types available.

More choices can be a blessing and a curse. Without the right understanding of new technologies and how they relate to your situation, the multitude of persistence options can be paralyzing. Our second guide of 2015 is here to help you navigate this sea of technology and understand where certain options might fit in your organization. You'll also learn about the trends and opinions surrounding modern database technology to get an idea of most companies' recipes for success.

This new guide topic is the result of a split in coverage from last year's Big Data guide. While DZone's *2014 Guide to Big Data* contained content and product information for both databases and analytics platforms, this guide focuses solely on databases and persistence utilities. Later this year, we will cover analytics platforms and data processing tools in a separate guide.

We're proud to present DZone's first pure guide to databases. I hope it gives you some new ideas and helps you build better software!

MITCH PRONSCHINSKE
SENIOR RESEARCH ANALYST
research@dzone.com

Table of Contents

Credits

DZONE RESEARCH

John Esposito
Editor-in-Chief

Jayashree Gopal
Director of Research

Mitch Pronschinske
Sr. Research Analyst

Benjamin Ball
Research Analyst

Matt Werner
Market Researcher

John Walter
Content Curator

Ryan Spain
Content Curator

MARKETING & SALES

Alex Crafts
VP of Sales

Matt O'Brian
Director of Business Development

Ashley Slate
Director of Design

Chelsea Bosworth
Marketing Associate

Chris Smith
Production Advisor

Brandon Rosser
Customer Success Advocate

Jillian Poore
Sales Associate

CORPORATE MANAGEMENT

Rick Ross
CEO

Matt Schmidt
President & CTO

Kellet Atkinson
General Manager

Special thanks *to our topic experts Ayobami Adewole, Eric Redmond, Oren Eini, Jim R. Wilson, Peter Zaitsev, and our trusted DZone Most Valuable Bloggers for all their help and feedback in making this report a great success.*

Summary & Key Takeaways

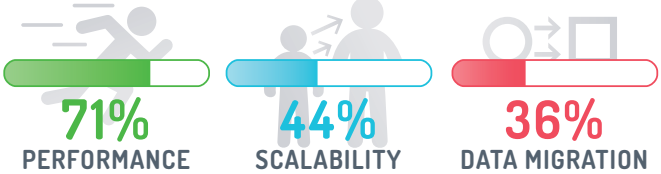
ALMOST ALL NON-TRIVIAL SOFTWARE PRODUCTS REQUIRE a database. Performant, scalable persistence is a constant concern for developers, architects, operations, and business-level managers. Companies of all sizes are now facing unprecedented data management and analysis challenges because of the sheer amount of data being generated, and the speed at which we want to digest it. Some estimates expect the amount of digital data in the world to double every two years [1]. The predictions for data growth are staggering no matter where you look. DZone's *Guide to Database and Persistence Management* is a valuable handbook for understanding and conquering the challenges posed by modern database usage.

The resources in this guide include:

- Side-by-side feature comparison of the best databases and database hosting services.
- Comprehensive data sourced from 800+ IT professionals on database usage, data storage, and experiences.
- A step-by-step database migration checklist.
- An introduction to several innovative new data persistence technologies.
- Forecasts and opinions about the future of database usage.

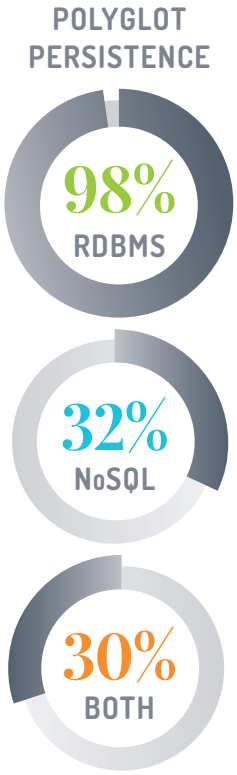
KEY TAKEAWAYS

DATABASE PERFORMANCE IS THE BIGGEST CHALLENGE FOR OPERATIONS AND ENGINEERING Developers and operations professionals both agree that achieving and maintaining high database performance is their biggest challenge related to persistence. 71% of respondents in DZone's 2015 Databases survey said performance was one of their biggest challenges related to databases, which was significantly higher than other challenges such as scalability (44%) and data migration (36%). This highlights the importance of choosing the right database and getting familiar with its query language, so that queries can be optimized and application access can be tuned.



DEVOPS PRACTICES STRUGGLE TO REACH DATABASE ADMINISTRATION Although DZone's 2015 Continuous Delivery survey showed modest growth (+9%) for respondents that extended Continuous Delivery practices to their database, the overall number (30%) is about half the amount of respondents using CD practices in application build management (61%). One key reason why CD isn't more common in database administration might be due to the lack of version control used in databases. In this year's databases survey, respondents were asked if they used any form of version control for their database. Nearly half, 48%, said no.

RDBMS AND THE "BIG 3" DATABASES STILL DOMINATE The 2014 Big Data survey and this year's database survey both confirm that the clear "Big 3" databases in IT are Oracle, MySQL, and Microsoft SQL Server. All three resided in the 50%-57% range for organizational use this year, while the other databases were further behind. PostgreSQL is used in 31% of respondents' organizations and MongoDB (the most-used NoSQL database) is used in 20% of the organizations. Even though NoSQL databases were touted as the next evolution in persistence by developers who were fed up with certain aspects of RDBMS, relational data stores are still a dominant part of the IT industry (98% use a relational database), and the strengths of SQL databases have been reiterated by many key players in the space. Today, experts instead suggest a multi-database solution, or polyglot persistence approach. 30% of respondents' organizations currently use a polyglot persistence approach where at least one relational database and one non-relational database are used.



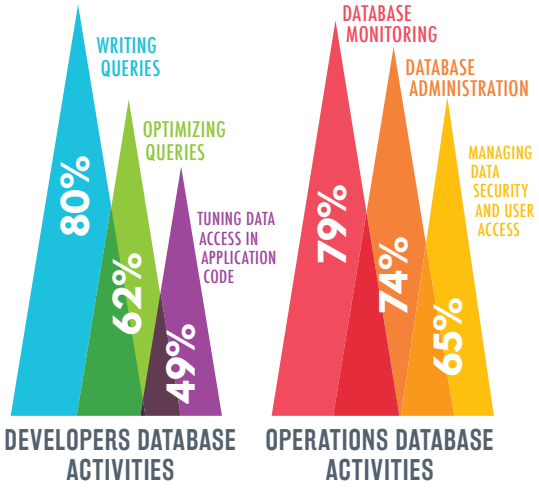
[1] http://www.emc.com/about/news/press/2014/20140409-01.htm

KEY RESEARCH FINDINGS

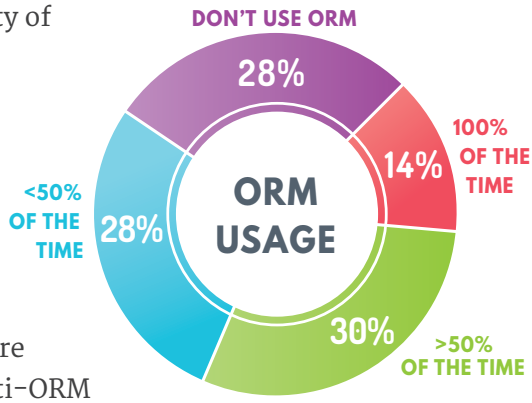
More than 800 IT professionals responded to DZone's 2015 Databases Survey. Here are the demographics for this survey:

- Developers (45%) and development team leads (28%) were the most common roles.
- 63% of respondents come from large organizations (100 or more employees) and 37% come from small organizations (under 100 employees).
- The majority of respondents are headquartered in Europe (40%) or the US (32%).
- Over half of the respondents (69%) have over 10 years of experience as IT professionals.
- A large majority of respondents' organizations use Java (85%). C# is the next highest (40%).

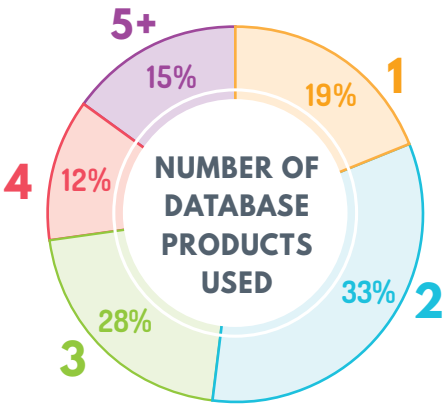
DEVELOPERS FOCUS ON QUERIES, OPERATIONS FOCUSES ON MONITORING AND ADMIN There were significant differences between the database-related activities of developers and operations professionals. Developers spend most of their time writing queries (80%), optimizing queries (62%), and tuning data access in application code (49%). Operations professionals focus on database monitoring (79%), database administration (74%), and managing data security and user access (65%). Operation professionals also write queries frequently (65%), but they are involved in diagnosing production database issues (62%), tuning database configuration (50%), and their other three primary jobs mentioned above much more often than developers.

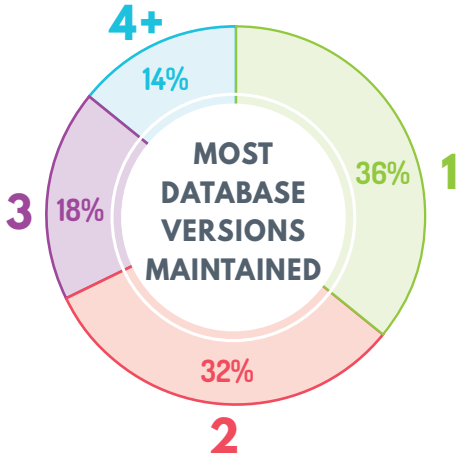


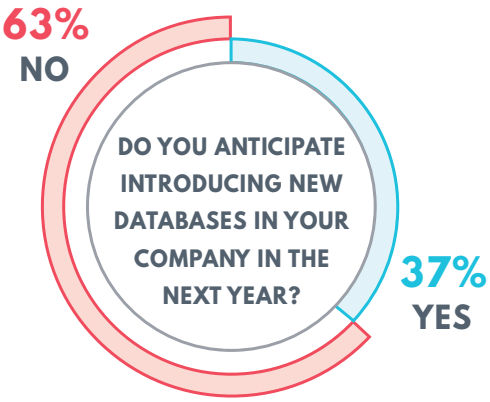
OBJECT-RELATIONAL MAPPERS ARE USED IN MODERATION For years, developers have constantly argued about whether Object-Relational Mapping tools (ORMs) are a useful solution, or a detrimental anti-pattern. According to the survey results, a majority of developers use ORMs, but not all of the time. Slightly more than a quarter of respondents don't use ORMs (28%). While there are still vocal anti-ORM developers out there, the silent majority seems to understand the risks, limitations, and benefits of ORMs enough to take a balanced approach, using them where they are appropriate.



FEW DATABASE ADMINISTRATION TASKS ARE AUTOMATED A majority of respondents automate database backup (65%) and database monitoring (56%), but most



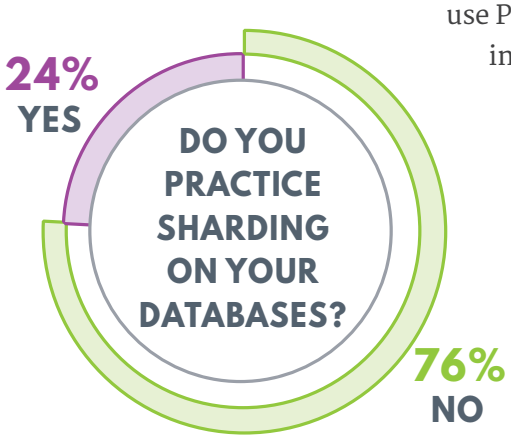




respondents do other database administration tasks manually. About one fourth of respondents automated database change management (25%), documentation generation (24%), and installation/patching (26%). Tasks such as tuning database configuration, optimizing queries, and diagnosing production database issues were mainly manual.

ONE TYPE OF DATABASE IS USUALLY NOT ENOUGH

81% of respondents used two or more database products in their organization. The largest percentage of respondents use two products (33%). Another interesting statistic is the number of different versions that respondents' organizations maintain for a given database. Most maintain one (36%), but a fair number also maintain two (32%), revealing that tech companies aren't afraid to use multiple products, but they do dislike the difficulty of maintaining several versions.



DATABASES MOVING TOWARD CLOUD INFRASTRUCTURE AND DISTRIBUTED ARCHITECTURE

56% of respondents have distributed architectures with many database instances. 45% of those respondents make their own internal distributed infrastructure while the other 11% use cloud infrastructure services.

Regarding the environments where respondents run their databases, 59% run them virtualized in a private datacenter, 53% run them on bare metal in a private datacenter, and 22% use a private cloud. Hybrid cloud, PaaS, IaaS, and DBaaS are all used by less than 10%. However, in the next year, fewer respondents expect that their databases will be virtualized in a private datacenter (58%) or hosted on bare metal in a private datacenter (46%), and more expect to be hosting them in the private cloud (33%). There are also 2%–3% increases in the number of respondents who expect to use PaaS, IaaS, and DBaaS. One other interesting statistic about distributed database environments is the usage of sharding, which seems to have stabilized after more experts began explaining when it is advantageous to shard and when it is harmful.

MANY ORGANIZATIONS ARE CONSIDERING NEW DATABASES

While the majority of organizations (63%) are satisfied with their current array of databases, a significant number of respondents expect to add a new database product to their stacks in the next year (37%). With the ever-growing number of new database options and persistence strategies, it's prudent for some developers to anticipate some changes in technology.

A REVIEW OF Persistence Strategies

BY ANDREW C. OLIVER

THE BIG DATA AND NOSQL TRENDS ARE, AT BEST, misnamed. The same technologies used to process large amounts of data are sometimes needed for moderate amounts of data that then must be analyzed in ways that, until now, were impossible. Moreover, there are "NewSQL" databases, or ways to run SQL against NoSQL databases that can be used in place of traditional solutions.

Many of these types of databases are not all that new, but interest in them wasn't high when they were first invented, or the hardware or economics were wrong. For the last 30 years or so, the tech industry assumed the relational database was the right tool for every structured persistence job. A better idea is to lump everything under "use the right tool for the right job."

Your RDBMS isn't going away. Some tasks are definitely tabular and lend themselves well to relational algebra. However, it is possible that some of the more expensive add-ons for your RDBMS will go away. Scaling a traditional RDBMS is difficult at best. Partitioning schemes, multi-master configurations, and redundancy systems offered by Oracle, SQL Server, and DB2 are expensive and problematic at best. They often fail to meet the needs of high-scale applications. That said, try joining two or three small tables and qualifying by one or two fields on Hadoop's Hive and you will be rather underwhelmed with its performance. Try putting truly tabular data into a MongoDB document and achieving transactional consistency and it will frustrate you.

Organizations need to identify the patterns and characteristics of their systems, data, and applications and use the right data technologies for the job. Part of that is understanding the types of data systems available and their key characteristics.

KEY-VALUE STORES

To some degree all NoSQL databases derive from key value stores. These can be thought of as hash maps. Their capabilities are fairly limited: they just look up values by their key. Most have alternative index capabilities that allow you to look up values by other characteristics of the values, but this is relatively slow and a good indication that maybe a K-V store is not the best choice. "Sharding" or distributing the data across multiple nodes is very simple. The market is crowded with K-V stores such as Aerospike, Redis, and Riak, all of which are simple to implement.

COLUMN-FAMILY DATABASES

Many K-V stores also operate as column family databases. Here the "value" evolves into a multidimensional array. These are again most efficient while looking up values. However, the "secondary indexes" are often implemented with pointer tables, meaning a hidden table with that field as the key points to the key of the original table. This is essentially O(1)+O(1) for a lookup time, which is not as good as O(1), but still decent. The two most popular, Cassandra and HBase, are both based on Hadoop but have different write semantics. HBase offers strong write integrity; Cassandra offers "eventual consistency." If you have a fair number of reads and writes, and a dirty read won't end your world, Cassandra will concurrently handle them well. A good example of this is Netflix: if they are updating the catalog and you see a new TV show, but the seasons haven't been added and you click on the title and there is a momentary glitch where you only see season one until the next time you click—would you even notice? However, this is unlikely to happen because by the time you click it was likely added. Obviously this is no way to do accounting, but it is perfectly fine for a lot of datasets. HBase offers more paranoid locking semantics for when you need to be sure the reads are clean. This obviously comes at a cost in terms of concurrency, but is still about as good as row-locking in an RDBMS. Time series data is classically given as an optimal use case for column-family databases, but there are lots of other strong use cases such as catalogs, messaging, and fraud detection.

COLUMN-ORIENTED RDBMS

These are the "new hot thing" with SAP's HANA being so well marketed. The only reason you can't have a distributed RDBMS is because they didn't evolve that way. Granted, distributed joins can be heavy, but if optimized correctly they can be done more efficiently than, say, MapReduce against Hadoop's Hive (which is notoriously slow for many types of queries). Column-oriented relational databases may also perform better than traditional RDBMS for many types of queries. Conceptually, think of turning a table sideways: make your rows columns and columns into rows. Many values

in an RDBMS are repetitive and can be "compressed" with either pointers or compression algorithms that work well on repetitive data. This also makes lookups or table scans much faster, and data partitioning becomes easier. In addition to HANA, there is also Splice Machine, which is built on top of HBase and Hadoop. The promises of these databases include high-scale distribution with column-oriented performance and good compression performance, and they still offer transactional integrity equivalent to a traditional RDBMS.

DOCUMENT DATABASES

To some, these are the holy grail of NoSQL databases. They map well to modern object-oriented languages and map even better to modern web applications sending JSON to the back-end (i.e. Ajax, jQuery, Backbone, etc.). Most of these natively speak a JSON dialect. Generally the transactional semantics are different, but comparable to an RDBMS. That is, if you have a "person" in an RDBMS, their data (phone numbers, email addresses, physical addresses) will be distributed across multiple tables and require a transaction for a discrete write. With a document database, all of their characteristics can be stored in one "document," which also can be done in discrete writes. These are not so great if you have people, money, and other things all being operated on at once, involving operations that complete all or none because they do not offer sufficient transactional integrity for that. However, document databases scale quite well and are great for web-based operational systems that operate on a single big entity, or systems that don't require transactional integrity across entities. There are also those who use document databases as data warehouses for complex entities with moderate amounts of data where joining lots of tables caused problems. MongoDB and Couchbase are typically the leaders in this sector.

GRAPH DATABASES

Complex relationships may call for databases where the relationships are treated as first order members of the database. Graph databases offer this feature along with better transactional integrity than relational databases for the types of datasets used with Graphs—the relationships can be added and removed in a way similar to adding a row to a table in an RDBMS, and that too is given all-or-none semantics. Social networks and recommendation systems are classic use cases for graph databases, but you should note that there are a few different types of graph databases. Some are aimed more at operational purposes (Neo4j) while others are aimed more at analytics (Apache Giraph).

RDBMS

Some datasets are very tabular, having use cases that lend themselves well to relational algebra. They require lots of different things to happen in all-or-none transactions, but don't expand to tens of terabytes. This is the relational database's sweet spot, and it is a big one. Even things that do not fit nicely but have been made to work after a decade of hammering may not be worth the migration cost. This is not about paring down your toolkit, it is about adding tools so you always have the right one for the job.

WHAT ABOUT HADOOP?

Hadoop is not a database. HDFS (Hadoop Distributed File System) distributes blocks of data redundantly across nodes, similar to Red Hat's Gluster, EMC's IFS, or Ceph. You can think of it as RAID over several servers on the network. You store files on it much as you would any file system; you can even mount it if you like. It uses MapReduce, which is a framework for processing data across a network. This was popularized by Hadoop but is quickly giving way to DAG (distributed acyclic graph) frameworks such as Spark.

TRY PUTTING TRULY TABULAR DATA INTO A MONGODB DOCUMENT AND ACHIEVING TRANSACTIONAL CONSISTENCY AND IT WILL FRUSTRATE YOU.

While the original use cases for Hadoop were pure analytics or data warehousing models where data warehousing tools were not sufficient or economically sensible, the ecosystem has since expanded to include a number of distributed computing solutions. Storm allows you to stream data, Kafka allows you to do event processing/messaging, and other add-ons like Mahout enable machine learning algorithms and predictive analytics. The use cases are as diverse as the ecosystem. Chances are if you need to analyze a large amount of data, then Hadoop is a fine solution.

PUTTING IT TOGETHER

When you understand the characteristics of the different types of storage systems offered, the characteristics of your data, and the scenarios that your data can be used for, choosing the right set of technologies is not a daunting task. The era of one big hammer (the RDBMS) to suit every use case is over.

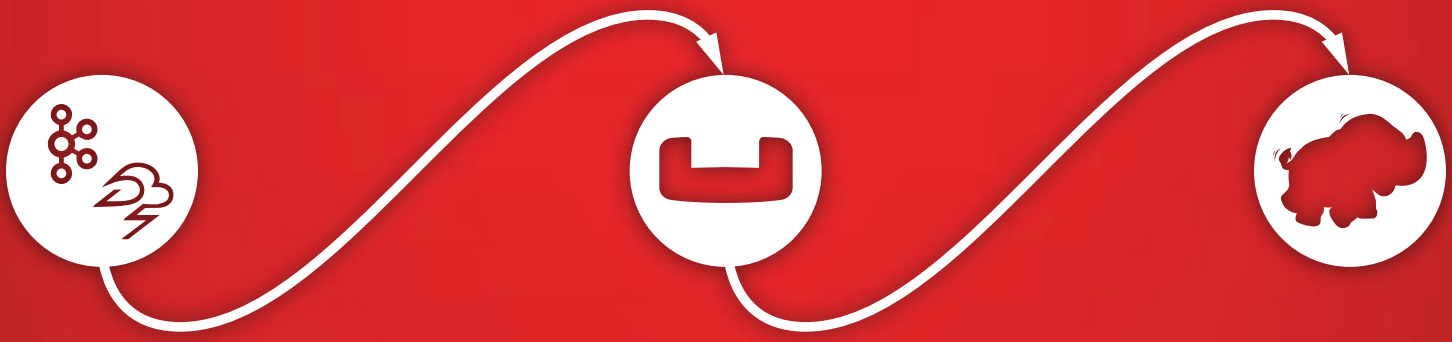
ANDREW C. OLIVER is the president and founder of Mammoth Data, a Durham based Big Data/NoSQL consulting company specializing in Hadoop, MongoDB and Cassandra. Oliver has over 15 years of executive consulting experience and is the Strategic Developer blogger for InfoWorld.com.

FILLING HADOOP *by the* BUCKET?

Can't afford to be a day late or a bucket short?

Rely on a high-performance, distributed database with streaming integration for Kafka and Storm to fill Hadoop.

Try **Couchbase Server.**



Interested in learning more? See how PayPal leverages Couchbase Server and Hadoop to power real-time analytics.

COUCHBASE.COM/PAYPAL

Couchbase

Building a Faster Real-time Big Data Solution

There was a time when big data meant Hadoop. Big Data was really just offline analytics. That's no longer the case.

In today's world, Hadoop can no longer meet big data challenges by itself. The explosion of user-generated data is being followed by an explosion of machine-generated data. Hadoop is great for storing and processing large volumes of data, but it's not equipped to ingest it at this velocity. This has led to a new generation of systems engineered specifically for real-time analytics, including Storm and Couchbase Server.

THERE ARE THREE BIG DATA CHALLENGES:

1. **Data volume:** The amount of data being generated.
2. **Data velocity:** The rate at which data is being generated.
3. **Information velocity:** The rate at which information must be generated from raw data.

Hadoop addresses data volume. It can store and process a lot of data, later. It scales out to store and process more data. Hadoop doesn't address data velocity. However, it meets offline analytical requirements.

Couchbase Server addresses data velocity. It is a high-performance NoSQL database that can store a lot of data, in real time. It scales out to store a lot of data, faster. It meets operational requirements.

Storm addresses information velocity. It scales out to process streams of data, faster. It meets real-time analytical requirements.

Integrating Hadoop, Couchbase Server, and Storm creates a faster big data solution. Hadoop and Couchbase Server form the inner core. Hadoop enables big data analysis. Couchbase Server enables high performance data access. Storm provides the outer core – a stream processing system.

The best thing about building a real-time big data solution is designing the architecture. It's like playing with Legos and putting the pieces together that best fit your use case. What will your architecture look like?

WRITTEN BY
Shane Johnson, *Senior Product Marketing Manager, Couchbase*

CLASSIFICATION DOCUMENT AND KEY-VALUE STORE WITH INTEGRATED CACHING

Couchbase Server by Couchbase

Couchbase

A high-performance distributed database with integrated caching, cross data center replication, mobile data access, and Hadoop integration certified by Cloudera and Hortonworks.

STORAGE MODEL
Document, Key-Value, Distributed cache

LANGUAGE DRIVERS
Java .NET Node.js Go PHP C Python Ruby

ACID ISOLATION
Locking

TRANSACTIONS SUPPORTED
Compare and Swap (CAS)

CASE STUDY PayPal integrated Couchbase Server, Storm, and Hadoop to build the foundation of its real-time analytics platform. Clickstream and interaction data from all channels is pushed to the platform for real-time analysis. The data is pulled in a stream processor, Storm, for filtering, enrichment, and aggregation. After the data is processed, it's written to Couchbase Server where it's accessed by rich visualization tools. With these tools, PayPal can monitor all traffic in real-time. In addition, PayPal leverages views in Couchbase Server to perform additional aggregation and filtering. Finally, the data is exported from Couchbase Server to Hadoop for offline analysis.

INDEXING CAPABILITIES
Incremental MapReduce

REPLICATION
- Synchronous
- Asynchronous

CUSTOMERS
- AOL
- eBay
- LivePerson
- PayPal
- AT&T
- LinkedIn
- Neiman Marcus

BUILT-IN TEXT SEARCH?
Via ElasticSearch and Solr

BLOG blog.couchbase.com | TWITTER @couchbase | WEBSITE couchbase.com

Firebase, Meteor & Cognito: Just Enough Database for a Mobile World

BY BEN TEESE

EVERYBODY IS BUILDING MOBILE APPS THESE days, both natively and for the web. Part of the value of almost any non-trivial app relates to its ability to store data for a user. This can range from something as obvious as a to-do list app, whose entire value-proposition is its ability to remember things for a user, to less obvious cases like mobile games, which may want to store game state or high-scores for the user.

Rather than just storing this data on a single device, users also now increasingly expect apps to make their data quickly and easily accessible across all of their devices. Furthermore, when they have multi-device access, they don't want to have to worry about whether internet connectivity is missing or intermittent. The app should be able to store it locally and then synchronize with the cloud later.

As soon as remote data storage becomes a requirement for an app, a whole bunch of new back-end infrastructure needs to be set up by the developer. Even with the rise of the cloud infrastructure, this is still a lot of work.

- You need to **pick app server and data store software**, then run a bunch of wiring code to pipe data between the client and database.
- If you want **offline data access** to work, you need to build local storage into your client and get the local-remote syncing logic right.
- If you want **real-time data syncing** between clients, you need to introduce real-time web protocols, or simulate them using techniques like long-polling.
- Finally, to **secure your user's data** so that only they can see it, you need to either build your own infrastructure to securely store and verify user credentials, or integrate with services like Facebook or Twitter to allow people to use their existing credentials.

In this article, we'll look at three platforms that aim to give developers a quick and easy way to implement the common mobile app use case of "storing stuff" for users across devices. All of these frameworks can run in offline mode and offer mechanisms for automatic syncing. They all also have the ability to integrate with OAuth-based identity services.

It's important to understand that these platforms are not relational databases and thus do not offer features like JOIN queries or full ACID transactions. Instead, their goal is to provide *just enough* data storage capabilities for a mobile app to satisfy the storage needs of its users. However, they each have their own limitations and trade-offs.

FIREBASE

The first platform we'll cover—and probably the best known—is Firebase, an API for storing and syncing data. Recently acquired by Google, Firebase provides client-side libraries for Android, iOS, all major JavaScript browser frameworks, Node.js, and Java. These libraries give clients impressive real-time data-synchronization capabilities. The Firebase platform is essentially a fully-hosted service that stores JSON in a tree accessed via RESTful URLs. One thing to note is that when you load a node in the tree, you get all of the data under that node. Consequently, to avoid downloading too much stuff to the client, it's important to minimize the depth of the tree.

Firebase allows data to be shared between users. To restrict how data can be accessed by particular users, it provides a server-side rules language. However, it's important to understand that Firebase has no application layer by default—you essentially go directly from your client to the database. Because of this, the rules language also includes capabilities for doing data validation.

Firebase has simple querying capabilities. However, it's important to configure indexing rules on query criteria to keep performance manageable. Queries that you run with Firebase are *live*—if the data underlying the query changes after the query has been run, then the query results will automatically update.

METEOR

Meteor is a platform for building web and mobile apps. Like Firebase, it has very impressive data synchronization capabilities, including support for live queries. However, Meteor goes even further by allowing you to write code that can be run on both the client and the server-side. This gives you a great deal of flexibility from a security and data validation perspective. The catch is that *all* development

has to be done with Javascript, HTML, and CSS. If you want to deploy your app to an app store, the best you can do is package it all up inside a native wrapper. By default, Meteor uses MongoDB as a data store, which basically means that you get Mongo's query capabilities. There is also early support for Redis and plans for other databases in future.

It is very easy to get up-and-running with Meteor, as it bundles both an application server container and a local Mongo instance out of the box. It also provides command-line tools for automatically packaging your client and server-side code and deploying it to the cloud with a hosted app server and datastore. This saves you from having to setup a complex build and deployment process yourself. The Meteor platform is comprised of a number of libraries and tools. While they can be used in isolation, they work best together. For example, Meteor's live-updating view library (Blaze) is designed to integrate seamlessly with its remote data syncing capabilities.

One downside of this integration is that, to get up and running quickly, you have to learn the framework in its entirety, putting aside any skills you might already have with Javascript MVC frameworks like AngularJS, Ember.js, or React. Some work has been done integrating Meteor with AngularJS to get the best of both worlds, but it is only at an early stage.

COGNITO

Cognito is a new offering from Amazon Web Services (AWS) that provides a quick and easy way to authenticate users and store data for them. It integrates tightly with AWS's existing identity management infrastructure, but can also be used with other OAuth providers. Like Firebase, clients go directly to the database rather than through an application layer. To make this secure, no data can be shared between users. Any additional data validation you do will have to be duplicated between different client types, as there is no shared server to put it on.

Interestingly, Cognito has gone with a native-first approach to its client-side libraries. Full support for both the identity and data synchronization components of Cognito is available for iOS and Android, but the synchronization component for the Javascript libraries is still only in developer preview mode. These libraries provide simple support for syncing data between local and remote storage. However, in contrast to Meteor and Firebase, developers have to manually trigger when they want synchronization to happen, and handle any conflicts themselves.

Cognito's data storage capabilities are very basic. An app can have multiple datasets, each of which can be considered roughly equivalent to a table. Keyed records can be inserted into each dataset. However, you can only look up records by their key—any more sophisticated querying would require a full scan of the dataset. That said, if Cognito's data-storage capabilities are not enough, you can also use it to connect to other AWS services like S3 or DynamoDB. However, you lose its offline and synchronization capabilities when you do this.

All of these platforms herald a shift away from the traditional three-tier, request-response approach to storing and accessing user data.

WHICH ONE YOU MIGHT USE

Which of these frameworks you use depends very much on your storage needs. If you have really simple requirements and you're looking specifically for user authentication as a service, Cognito should be enough, especially if you are already familiar with the Amazon ecosystem. However, its inability to query or share data between users will be a deal-breaker for many scenarios.

For the majority of use cases, Firebase is probably a better fit, as it gives you a far more powerful datastore. The biggest risk with Firebase is that your security and validation requirements won't be met by Firebase's rules mechanism. If that's the case, you could start using Firebase in your own app server, or move onto a full-stack framework like Meteor. The natural trade-off of the full-stack approach is that you get more locked in to the specifics of the platform. In the case of Meteor, this means you have to fully embrace JavaScript and the Meteor way of doing things.

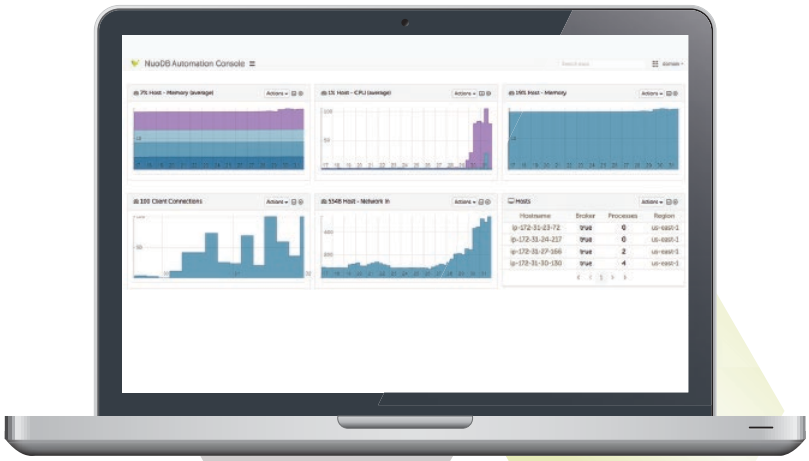
Regardless of which you pick, all of these platforms herald a shift away from the traditional three-tier, request-response approach to storing and accessing user data. To one degree or another, they take care of the minutiae of data storage, ensuring our user data is instantly available everywhere, all the time.

BEN TEESE is a senior consultant at Shine Technologies. He spent 10 years as a back-end Java developer before moving into front-end development with Rails and, more recently, Javascript SPAs. He spends most of his time building mobile web apps and, occasionally, iOS apps.

Modern
applications require
modern *databases*

Stop tying your new application to 30+ year old technology.

It's time to try a database as **innovative** *as you are.*



NuoDB's scale-out
SQL database offers:

- **Scale-Out Performance**
- **Continuous Availability**
- **Geo-Distribution**

Download the NuoDB Community Edition for free today!

www.nuodb.com/download

nuoDB®

WHEN TO SHARD? OR, WHY SHARD AT ALL?

Sharding has enabled companies to scale out a traditional SQL database. But not without consequence: The introduction of sharding has also delivered fragility, application complexity, and operational difficulty.

The ideal scenario starts with the database on one machine, and scales out as you need additional capacity, cache, parallelism, or support for more clients. But a traditional SQL database makes this impossible.

But what if you revisit the entire architecture of the relational database? By divorcing processing power requirements from data replication, you can provide a system that scales horizontally; that is always consistent; replicates data correctly and consistently; and in the face of failure, always fails correctly.

You still get the view of a single logical database, regardless of how many hosts you're running on, processes you have going, or datacenters you're running in. Since you're still programming to a single logical database, your developers are free to write their applications without having to think about operational concerns. And you can still maintain ACID compliance.

Plus, your operations people are free to scale the database however they need to, to take advantage of their resources. Then, when you need to bring things online on demand, or when you no longer need additional resources, and you want to shed them, you can do that without affecting the application.

The modern datacenter requires an elastically scalable DBMS that can quickly add new machines to a running database.

To view a video explaining how to build a scalable application without sharding, visit www.nuodb.com/no-shard.

WRITTEN BY
Michael Waclawiczek, *VP of Marketing and Operations, NuoDB*

CLASSIFICATION NEWSQL DATABASE

NuoDB by NuoDB

Scale-out SQL database, on-demand scalability, geo-distributed deployment, peer-to-peer distributed architecture.

STORAGE MODEL
Relational, Key-Value, Graph

LANGUAGE DRIVERS
C | C++ | C# | Go | Java | Node.js

ACID ISOLATION
- Serializable/linearizable

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions on a single node

CASE STUDY UAE Exchange, headquartered in Abu Dhabi, offers money transfer, foreign currency exchange, remittance, and related financial services. The company is a major player in these segments with over 725 branches in 32 countries serving 7.9 million customers. UAE Exchange handles 6.1% of the world's total remittances. One of the key elements in NuoDB, which was critical for us, was its ability to scale out/in on demand based on the application load. The ability to work across multiple datacenters, ease of provisioning interfaces and security out-of-the-box have all impressed us. When we were evaluating new database technologies, our primary aim was to ensure a low learning curve for our teams and a faster ramp up on leveraging the selected platform. NuoDB is ACID compliant and supports the ANSI SQL standard as well stored procedures. These factors played a crucial role in the adoption of NuoDB. Jayakumar V, Senior Manager, IT, UAE Exchange.

INDEXING CAPABILITIES
Primary Key

REPLICATION
- Asynchronous

CUSTOMERS
- Dassault Systemes
- Kodiak Networks
- EnterpriseWeb
- UAE Exchange
- Palo Alto Networks
- Auto Zone
- iCims
- Zombie

BUILT-IN TEXT SEARCH?
No

BLOG nuodb.com/techblog | TWITTER @nuodb | WEBSITE nuodb.com

THE STATE OF THE STORAGE ENGINE

BY BARON SCHWARTZ

Readers of this guide already know the database world is undergoing rapid change. From relational-only, to NoSQL and Big Data, the technologies we use for data storage and retrieval today are much different from even five years ago.

Today's datasets are so large, and the workloads so demanding, that one-size-fits-all databases rarely make much sense. When a small inefficiency is multiplied by a huge dataset, the opportunity to use a specialized database to save money, improve performance, and optimize for developer productivity and happiness can be very large. And today's solid-state storage is vastly different from spinning disks, too. These factors are forcing fundamental changes for database internals: the underlying algorithms, file formats, and data structures. As a result, modern applications are often backed by as many as a dozen distinct types of databases (polyglot persistence). These trends signal significant, long-term change in how databases are built, chosen, and managed.

MOST COMPANIES CAN AFFORD ONLY ONE OR TWO PROPER IN-DEPTH EVALUATIONS FOR A NEW DATABASE.

TEXTBOOK ARCHITECTURES LOSE RELEVANCE

Many of today's mature relational databases, such as MySQL, Oracle, SQL Server, and PostgreSQL, base much of their architecture and design on decades-old research into transactional storage and relational models that stem from two classic textbooks in the field—known simply as Gray & Reuters and Weikum & Vossen. This "textbook architecture" can be described briefly as having:

- Row-based storage with fixed schemas
- B-Tree primary and secondary indexes
- ACID transaction support
- Row-based locking
- MVCC (multi-version concurrency control) implemented by keeping old row versions

But this textbook architecture has been increasingly questioned, not only by newcomers but by leading database architects such as Michael Stonebraker. Some new databases depart significantly from the textbook architecture with concepts such as wide-row and columnar storage, no support for concurrency at all, and eventual consistency. It's worth noting that although NoSQL databases represent obvious changes in the data model and language—how developers access the database—not all NoSQL databases innovate architecturally. Coping with today's data storage challenges often requires breaking from tradition architecturally, especially in the storage engine.

LOG-STRUCTURED MERGE TREES

One of the more interesting trends in storage engines is the emergence of log-structured merge trees (LSM trees) as a replacement for the venerable B-Tree index. LSM trees are now about two decades old, and LevelDB is perhaps the most popular implementation. Databases such as Apache HBase, Hyperdex, Apache Cassandra, RocksDB, WiredTiger, and Riak use various types of LSM trees.

LSM trees work by recording data, and changes to the data, in immutable segments or runs. The segments are usually organized into levels or generations. There are several strategies, but the first level commonly contains the most recent and active data, and lower levels usually have progressively larger and/or older data, depending on the leveling strategy. As data is inserted or changed, the top level fills up and its data is copied into a segment in the second level. Background processes merge segments in each level together, pruning out obsolete data and building lower-level segments in batches. Some LSM tree implementations add other features such as automatic compression, too. There are several benefits to this approach as compared to the classic B-Tree approach:

- Immutable storage segments are easily cached and backed up
- Writes can be performed without reading first, greatly speeding them up
- Some difficult problems such as fragmentation are avoided or replaced by simpler problems
- Some workloads can experience fewer random-access I/O operations, which are slow

- There may be less wear on solid-state storage, which can't update data in-place
- It can be possible to eliminate the B-Tree "write cliff," which happens when the working set no longer fits in memory and writes slow down drastically

Although many of the problems with B-Tree indexes can be avoided, mitigated, or transformed, LSM tree indexes aren't a panacea. There are always trade-offs and implementation details. The main set of trade-offs for LSM trees are usually explained in terms of amplification along several dimensions. The *amplification* is the average ratio of the database's physical behavior to the logical behavior of the user's request, over the long-term. It's usually a ratio of bytes to bytes, but can also be expressed in terms of operations, e.g. number of physical I/O operations performed per logical user request.

- **Write amplification** is the multiple of bytes written by the database to bytes changed by the user. Since some LSM trees rewrite unchanging data over time, write amplification can be high in LSM trees.
- **Read amplification** is how many bytes the database has to physically read to return values to the user, compared to the bytes returned. Since LSM trees may have to look in several places to find data, or to determine what the data's most recent value is, read amplification can be high.
- **Space amplification** is how many bytes of data are stored on disk, relative to how many logical bytes the database contains. Since LSM trees don't update in place, values that are updated often can cause space amplification.

In addition to amplification, LSM trees can have other performance problems, such as read and write bursts and stalls. It's important to note that amplification and other issues are heavily dependent on workload, configuration of the engine, and the specific implementation. Unlike B-Tree indexes, which have essentially a single canonical implementation, LSM trees are a group of related algorithms and implementations that vary widely.

There are other interesting technologies to consider besides LSM trees. One is Howard Chu's LMDB (Lightning Memory-Mapped Database), which is a copy-on-write B-Tree. It is widely used and has inspired clones such as BoltDB, which is the storage engine behind the up-and-coming InfluxDB time-series database. Another LSM alternative is Tokutek's fractal trees, which form the basis of high-performance write and space-optimized alternatives to MySQL and MongoDB.

EVALUATING DATABASES WITH LOG-STRUCTURED MERGE TREES

No matter what underlying storage you use, there's always a trade-off. The iron triangle of storage engines is this:

You can have **sequential reads without amplification**, **sequential writes without amplification**, or an **immutable write-once design**—*pick any two*.

Today's emerging Big Data use cases, in which massive datasets are kept in raw form for a long time instead of being summarized and discarded, represent some of the classes of workloads that can potentially be addressed well with LSM tree storage (time-series data is a good example). However, knowledge of the specific LSM implementation must be combined with a deep understanding of the workload, hardware, and application.

ALTHOUGH NOSQL DATABASES REPRESENT OBVIOUS CHANGES IN THE DATA MODEL AND LANGUAGE, NOT ALL NOSQL DATABASES INNOVATE ARCHITECTURALLY.

Sometimes companies don't find a database that's optimized for their exact use case, so they build their own, often borrowing concepts from various databases and newer storage engines to achieve the efficiency and performance they need. An alternative is to adapt an efficient and trusted technology that's almost good enough. At VividCortex, we ignore the relational features of MySQL and use it as a thin wrapper around InnoDB to store our large-scale, high-velocity time-series data.

Whatever road you take, a good deal of creativity and experience is required from architects who are looking to overhaul their application's capabilities. You can't just assume you'll plug in a database that will immediately fit your use case. You'll need to take a much deeper look at the storage engine and the paradigms it is based on.

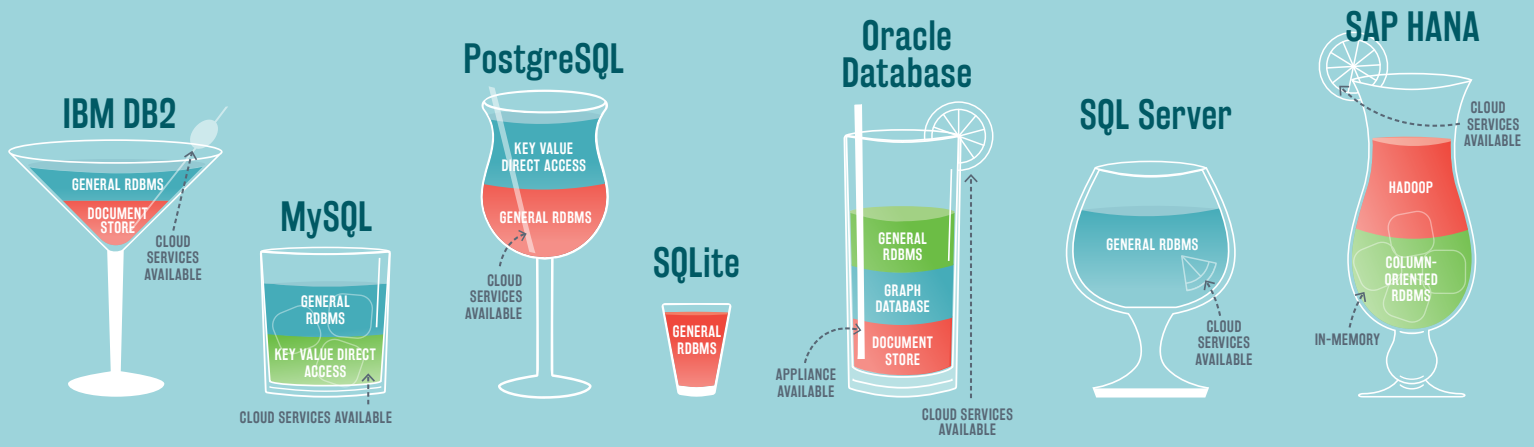
BARON SCHWARTZ is the founder and CEO of VividCortex, the best SaaS for seeing what your production database servers are doing. He is the author of *High Performance MySQL* and many open-source tools for MySQL administration. He's also an Oracle ACE and frequent participant in the PostgreSQL community.

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)

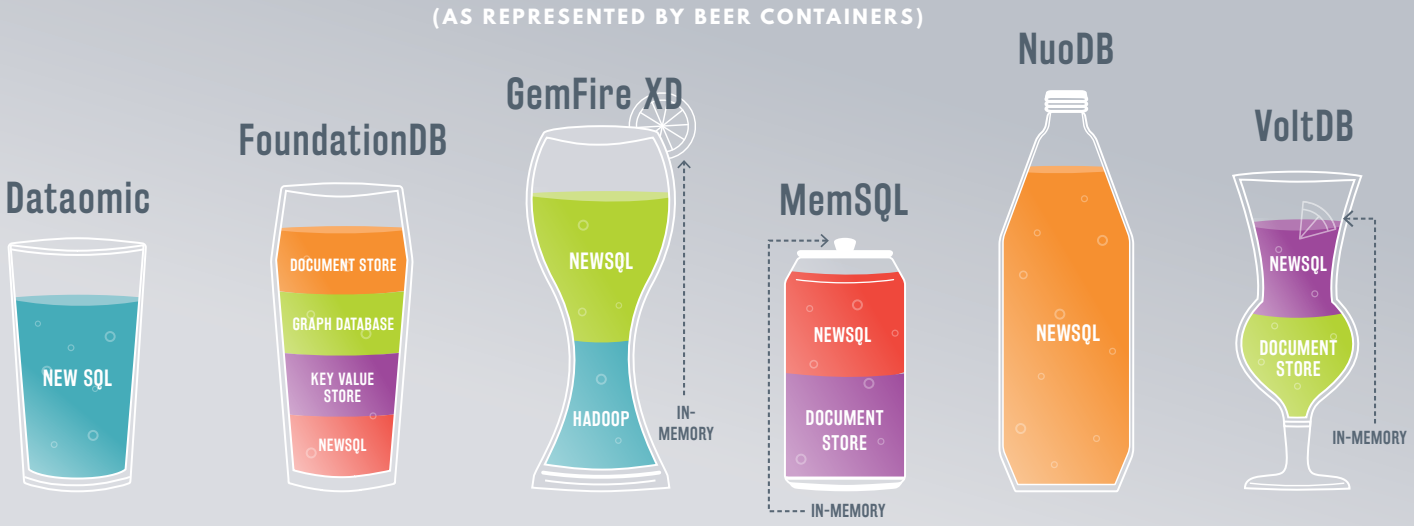


NOSQL

(AS REPRESENTED BY COFFEE AND ESPRESSO GLASSWARE)

NEW SQL

(AS REPRESENTED BY BEER CONTAINERS)



DATA GRIDS

(AS REPRESENTED BY WINE & CHAMPAGNE GLASSWARE)

3 Reasons

WHY IT'S OKAY TO STICK WITH SQL

BY LUKAS EDER

THE PAST DECADE HAS BEEN AN EXTREMELY exciting one in all matters related to data. We have had:

- An ever increasing amount of data produced by social media (once called "Web 2.0")
- An ever increasing amount of data produced by devices (a.k.a. the Internet of Things)
- An ever increasing amount of database vendors that explore "new" models

Marketers, publishers, and evangelists coined the term "Big Data" and "NoSQL" to wrap up the fuzzy ideas around persistence technology that have emerged in the last decade. But how big is "Big Data"? Do you do "Big Data"? Do you *need* to do "Big Data"?

The truth is... you don't. You can keep using the same architecture and relational databases you've always had, for these three reasons.

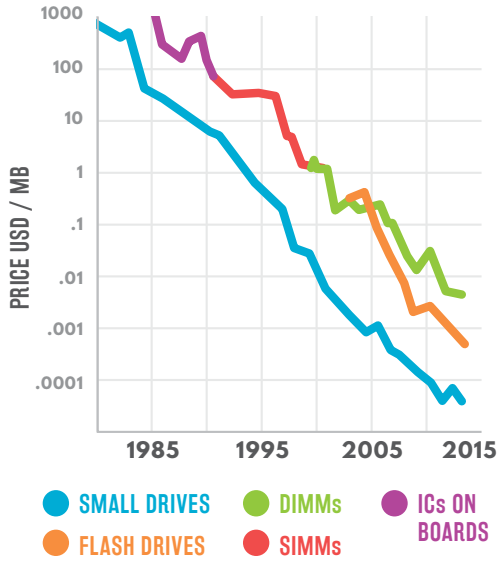
1. RAM PRICES ARE CRUMBLING

For a while, economists and researchers have been predicting the end of Moore's Law. With single-CPU processing power reaching the limits of what's possible in non-quantum physics, we can no longer scale up processing power cheaply. Multi-core processors have emerged, and vendors have encouraged sharing data across several machines for a long time.

But do we really need to distribute data onto several machines and deal with the struggles introduced by the CAP theorem in distributed environments? The truth is, you may not have to pay this price just yet!

While CPUs aren't getting faster anymore, RAM continues to get cheaper. Today, a database server that can keep your whole database in memory constantly is fairly affordable.

HISTORICAL COST OF COMPUTER MEMORY & STORAGE [1]



Several vendors of commercial RDBMS have implemented in-memory computing capabilities and column stores in their standard products: Oracle 12c, SQL Server 2012, SAP HANA, just to name a few. One of the most interesting examples of a "classic" application that runs on a single centralized RDBMS is the Stack Exchange platform, with around 400 GB of RAM and 343M queries per day [2].

This architecture has been the preferred way for any software system that profited from the free ride offered by Moore's Law in the past: "Just throw more hardware at it, and it'll run again." The nightmare of having to distribute such a system is postponed yet again.

In other words: The primary source of contention, the disk, is no longer the bottleneck of your application, and you can scale out on a single machine.

2. SQL IS THE BEST LANGUAGE FOR QUERYING

Winston Churchill may have been known for saying something like: *"SQL is the worst form of database querying. Except for all the other forms."*

QUEL may have been the better language to start with from a technical perspective, but due to a cunning move by Oracle and IBM in the early 80s, SQL and the ANSI/ISO SQL standard won the race. One of the most fundamental criticisms of SQL was that it is not really a relational language. This was recognized as early as in 1983 by none other than C.J. Date in his paper "A Critique of the SQL Database Language," but it was too late [3]. SQL had already won the race. Why?

SQL has been designed to be used primarily by humans to create ad-hoc queries. It is remarkable that the SQL language is pretty much the only declarative language

that has survived and maintained a continuous level of popularity.

Declarative thinking is hard enough when most people prefer to instruct machines in an imperative style. We can see attempts at creating such languages not only in databases, but also in general-purpose languages, such as Java. Specifically, Java EE is full of annotations that are used to tag types and members for an interpreter to pick up those tags and inject behavior. Once you combine JPA, JAXB, JAX-RS, JAX-WS, EJB, perhaps even Spring, and many other tools in a single application, you will immediately see how hard it is to still understand what all those declarative items mean, and how and when they interact. Usually, the behavior is not specified or ill-specified.

SQL is a comparatively simple declarative language, with a lot of limitations. This is good, because the lack of language features allows for implementations to quickly meet the SQL standard's requirements. In fact, this generality of the SQL language has even lead to a variety of NoSQL databases adopting SQL or imitating SQL closely:

- SQL on Hadoop
- Phoenix for HBase
- Cassandra's CQL resembles SQL
- JCR-SQL2 for content repositories

Not to mention, of course, the numerous actual SQL database implementations out there.

While popular ORMs help abstract away the tedious work of implementing CRUD with SQL, there has been no replacement thus far for SQL when it comes to querying and bulk data processing—it doesn't matter if your underlying storage is relational, or non-relational.

In other words: The non-relational-ness of SQL is one if the language's primary advantages, allowing technologies to incorporate other aspects of data processing, such as XML, OLAP, JSON, column stores, and many more.

3. PROCEDURAL LANGUAGES ARE IDEAL FOR COMPUTING

A third advantage that relational databases have over non-relational stores is the tight integration of procedural languages with the SQL language.

If scaling vertically is your strategy, you want to leverage as many cores as possible from your single database server, therefore, you want to run the computations as closely as possible to the data in your server's RAM. For many years, enterprise software architects have attempted to move all business logic into a middle tier, preferably written in J2EE and later in Java EE, helping large software vendors sell extremely expensive middleware components as a complement to their already expensive databases.

This may have been reasonable in the past since databases weren't as powerful 20 years ago as they are today. Today, however, commercial SQL optimizers are extremely powerful. All you need to get your SQL up to speed is appropriate indexing. If you decide to use a platform that doesn't allow you to use SQL, it will greatly increase your total cost of ownership for data processing logic. It is very hard to implement optimal execution plans manually, in the form of algorithms in your general-purpose imperative language, such as Java. There are ways around this, however, with APIs like jOOQ, which is a product our company works on that seamlessly integrates SQL or procedure calls into Java.

For everything that is not feasible with SQL, you can use your relational database's procedural language, which allows you to implement more complex and more explicit algorithms directly in the database, keeping business logic very close to the data. Not only can such languages access the data very quickly, they can also interact with SQL for bulk data processing.

In other words: Moving some computation-heavy logic into the database may be your best choice for a variety of use cases. I predict that more tech firms will start building around in-memory databases and this will inevitably translate into a higher popularity for procedural SQL languages like Transact-SQL (T-SQL), PL/SQL and SQLScript.

RDBMS HAVE WON IN THE PAST & WILL WIN AGAIN

"When all you have is a hammer, every problem will look like a nail"

There couldn't be a better metaphor for RDBMS. They're hammers. They're Swiss-Army-Hammers with millions of tools, and most developers can go as far as they need to with just that hammer. So, keep scaling up. With the hammer: SQL.

[1] http://www.jcmit.com/mem2014.htm
[2] http://stackexchange.com/performance
[3] http://dl.acm.org/citation.cfm?id=984551

LUKAS EDER is the founder and CEO of Data Geekery GmbH, located in Zurich, Switzerland. Data Geekery is a leading innovator in the field of Java/SQL interfacing. Their flagship product jOOQ is offering frictionless integration of the SQL language into Java.

Every company **wants** to put their existing data to work.

Actually doing it is a different story.

MAMMOTH DATA

That's where Mammoth Data comes in. We're Big Data consultants, change managers and risk mitigators. We make data-driven decision making possible.

THE LEADER IN BIG DATA CONSULTING

www.mammothdata.com | (919) 321-0119

Integrating Hadoop in a Business Intelligence Project

A business intelligence project is not about technology: it is about the people using the technology, how they communicate and what they want to do.

FUELING AN EXECUTIVE STRATEGY

Executive strategy combined with continued involvement is absolutely indispensable. First, the organization needs to figure out exactly what kind of intelligence they want to gather. If operational excellence is the primary goal then more time will be focused on measuring costs and efficiencies, but if you are trying to seek customer intimacy the project will focus on customer intelligence, social networks, and recommendations.

CONNECTING THE DOTS

Team interviews conducted by an outside party are essential. This enables your team to explain the ins and outs of the company without assuming prior knowledge. These interviews usually begin rather open-ended and end with focused questions to understand how each individual works with data, how they make decisions, and how that fits into the overall executive strategy.

TAKING THE BS OUT OF BI

Frequently these projects are implemented with BI tools like Pentaho or Tableau, though, some operational systems do not keep history and often data warehouses are too focused to provide for different uses of the data. Sometimes the solution is creating a new way to make use of the existing systems. Other times turning to Hadoop or, more particularly, Hive to integrate and provide a new view of the data into a data lake or similar structure.

WRITTEN BY
Elizabeth Edmiston, *VP of Engineering, Mammoth Data*

Mammoth Data

Mammoth Data is a data architecture consulting company specializing in new data technologies, distributed computing frameworks, like Hadoop, and NoSQL databases. By combining NoSQL and Big Data technologies with a high-level strategy, Mammoth Data is able to craft systems that capture and organize unstructured data into real business intelligence.

CASE STUDY Mammoth Data is assisting a major utility company in their smart grid analytics project. The project involves integrating data from disparate sources into a Hadoop Hive cluster. This involved getting and analyzing feeds, designing Pig-based load processes with Oozie, architecting a Data Lake and assisting the customer in integrating BI and Analytics tools.

TECHNOLOGIES
- Hadoop
- Cassandra
- Couchbase
- MongoDB
- Neo4j

VERTICALS
- Biotechnology
- Education/Non-profits
- Finance
- Healthcare
- Retail
- Technology
- Utilities

PARTNERS
- Cloudera
- Cloudbees
- Couchbase
- DataStax
- Hortonworks
- MongoDB
- Neo Technology

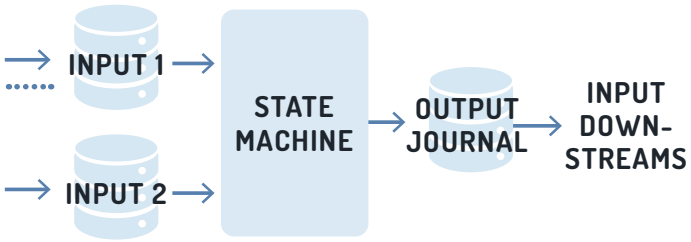
SERVICES
- Creation of an executive data strategy
- Analysis, design and implementation of Data-Driven company strategies
- Business Intelligence project analysis, design and development
- Initial install of a new technology
- Architect a data lake or data warehouse
- Implementation of new technology or migration from a legacy system

EMAIL info@mammothdata.com | TWITTER @mammothdataco | WEBSITE mammothdata.com

HOW TO MANAGE Deterministic Data-Driven Services

BY PETER LAWREY

How does the journaling of all data inputs and outputs for key systems help make it easier to test, maintain, and improve performance? What are some of the practical considerations of using this approach? One way to model a complex event-driven system is as a series of functions or state machines with the data inputs and a data output. Recording and replaying data for such state machines can help you test these systems for reliable behavior in terms of data output and timings. It can also make restarting and upgrading a running system easier.



Using Recorded Inputs and Outputs to Perform Extensive Tests

To create a small test for a state machine you can record real examples of events/messages/generic input and confirm that those inputs produce the expected messages/output. These tests are useful but have limited application. If you want to examine rare bugs or improve the latency distribution of your state machine, you need a far longer recording—ideally every event for a week.

Although replaying the input data of a system can reproduce realistic tests, you want to replay the output data of a system to reproduce the transient decisions made in production, like which event was processed next, or what the actual spacing of those events was down to the microsecond. Timestamps can be a guide, but are not reliable. If you want to upgrade a system in production, replaying from the output ensures the new system will honor the decisions already made, even if the new system would have made different choices.

Recording every event in and out of a system can help build data-driven tests, provide a simple recovery mechanism, and handle micro-bursts gracefully.

The Penalty of Recording Everything in Production

Making long recordings from a production system can raise concerns about the use of disk space and also slowing production response time. However, decreasing costs for disk space and the use of memory-mapped files can make extensive, detailed recording of every event practical. Output data can include meta information such as the history of an event with timing of key stages in the process. Systems that have introduced this recording technique successfully see lower latencies, even when recording every event in production. This is one of the best ways to ensure you have a deterministic latency profile for high-frequency trading systems.

Reproducing and Fixing Obscure Issues with Confidence

One of the challenges of fixing production bugs is

reproducing them in development. When you record all the inputs, you can't just debug and find out why the millionth event of the day was processed the way it was. You can fix a bug with confidence by showing that the root cause has been fixed, instead of finding a symptom of the bug, fixing that, and hoping it was the root cause.

Testing from recorded data allows you to test systems in isolation on a test server or development PC. The test doesn't depend on a database that could be either inaccessible (as it is in production) or regularly changing (making exact reproduction very difficult or impossible). Non-reproducible inputs, such as user inputs, can be replayed in an automated fashion with realistic timings.

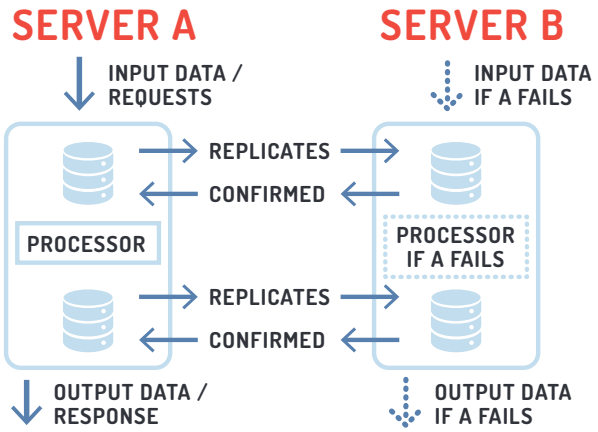
Journalling, Low Latency, and Resilience

A key requirement for many systems is the ability to restart on a failure. Being able to quickly replay or reload all the data driving a system is important to having acceptable restart times. By definition, deterministic software should fail the same way every time, so if there is a software bug, the solution is to reproduce this in testing, and release a fix. You have to have confidence that no decision already made will be changed by replaying the outputs already given by the old system. To test that replaying from the output works, you can chain the system to another instance of itself. State machine A produces an output that state machine A2 reads in replay mode. You can determine that the state machines have the same state at any point in a unit test.

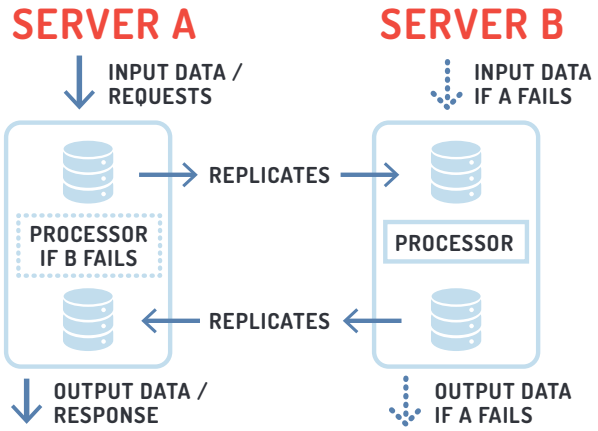
Best practice in Java is a 25 microsecond latency, 99% of the time, from an input on the network to an output. In C/C++, latencies are closer to 10 microseconds.

Some services can be horizontally distributed easily. They can exploit natural parallelism in the problem they are trying to solve by using many CPUs or machines. Some back office services need operations that are serialized, or have a poor degree of parallelism. To achieve high performance in this environment, you need a low latency system, ideally low latency with a high degree of consistency. These systems often need some form of redundancy in the event of a hardware failure.

How do we combine low latency, high availability, and persistence? Often you need to write to another device on the network, because waiting for data to write to the disk can be too slow. In this case, you trade disk latency for network latency. One of the key bottlenecks is the network round-trip time: the time it takes to send a message to a second machine and get a response back. One solution to this is to



have the second machine process the message. Server A receives data that must be processed and then must send a response. The critical time is from request to response. We want to avoid a loss of input or output as much as possible. One solution is to have Server A send the input to Server B, wait



for the response, process the message, send the output to Server B, and wait for confirmation before sending the message on. In this case, you pay the RTT (Round-Trip Time) twice. A faster solution is to have Server B do the processing.

Server A receives an input event, writes to a memory-mapped file, and sends it to Server B, which also writes it to a memory-mapped file. Server B processes the messages and creates the output, which is then written to a memory-mapped file on B and is replicated back to A. Lastly, Server A responds to the input event. At this point, there are two copies of the inputs and outputs on both machines, and you only have one RTT latency. The widely accepted best practice in Java is a 25 microsecond latency, 99% of the time, from an input on the network to an output. In C/C++, latencies are closer to 10 microseconds.

Keeping Flow Control as Light as Possible

When you have bursting data activity, you need some form of flow control to prevent systems from being overloaded or messages from being lost. One of the benefits of recording every input event is that you reduce the need for push back on the data producer. There is still flow control in the network, and the memory-mapped file places limits on how much uncommitted data you can have, but you now have a system that consumes data as fast as it can to an open-ended queue or journal, minimizing the need to ever push back on the producer.

By definition, deterministic software should fail the same way every time, so if there is a software bug, the solution is to reproduce this in testing, and release a fix.

Having a light touch flow control means you can timestamp your data more accurately (you get the data ASAP, not when you would like to get it), and you can replay your data without the real environment's flow control complicating timing reproduction significantly.

When you have a series of such systems, delays are exposed for what they are. A producer is very rarely slowed by a slow consumer, and you see clearly when the consumer couldn't keep up. If you use flow control, the producer conspires with the consumer to not make it look bad by giving it less work to do. You could record the flow control messages as well, but it is hard to estimate what the impact on your latency will be.

Monitoring the Behavior of Your System

Monitoring the performance of your system in a natural way without impacting the performance can be a challenge. While you can monitor the outputs in real-time by reading them—writing as much data as you might need—performance results need to be very lightweight to be credible.

Writing a detailed latency monitoring system, like replay systems for testing, can be an afterthought. If you build the system around consumers constantly replaying data from producers, you have implemented a replay system already. If you include a history of system timings in these messages, this history naturally falls out at the end of your final output; it is there along with the message, so you can also determine the significance (or insignificance) of the output event. Building a tool to read this history and present it in real-time can be surprisingly trivial.

Persisting Every Message is Valuable and Practical for Many Systems

Recording every event in and out of a system can help build data-driven tests, provide a simple recovery mechanism, and handle micro-bursts gracefully. The cost of implementing such a solution is falling and can lead to significant cost savings in development time and downtime for your key back-end systems.

PETER LAWREY is the lead developer of OpenHFT's Chronicle Queue. He has the most answers on StackOverflow for Java and the JVM and is the founder of the Performance Java User's Group. His blog "Vanilla Java" has had over 3 million views.

DIVING DEEPER INTO
DATABASE & PERSISTENCE MANAGEMENT

TOP TEN #DATABASE TWITTER FEEDS

 @markuswinand

 @spyced

 @markcallaghan

 @antirez

 @tmcallaghan

 @PeterZaitsev

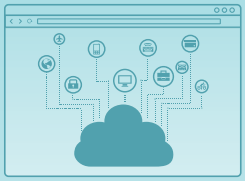
 @pbailis

 @xaprb

 @lukaseder

 @DatabaseFriends

DZONE DATABASE ZONES



NoSQL Zone dzone.com/mz/nosql

NoSQL (Not Only SQL) is a movement devoted to creating alternatives to traditional relational databases and excelling where they are weak. The NoSQL Zone is a central source for news and trends of the non-relational database ecosystem, which includes solutions such as Neo4j, MongoDB, CouchDB, Cassandra, Riak, Redis, RavenDB, HBase, and others.

SQL Zone dzone.com/mz/sql

The SQL Zone is DZone's portal for following the news and trends of the relational database ecosystem, which includes solutions such as MySQL, PostgreSQL, SQL Server, NuoDB, VoltDB, and many other players. In addition to these RDBMSs, you'll find optimization tricks, SQL syntax tips, and useful tools to help you write cleaner and more efficient queries.

Big Data Zone dzone.com/mz/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.

DZONE DATABASE REFCARDZ

JDBC Best Practices
bit.ly/1EbzQjm

JDBC Best Practices has something for every developer. JDBC was created to be the standard for data access on the Java Platform. This DZone Refcard starts off with JDBC basics including descriptions for each of the 5 Driver Types. This is followed up with some advanced JDBC on Codeless Configuration, Single Sign-on with Kerberos, Debugging and Logging.

MongoDB: Flexible NoSQL for Humongous Data
bit.ly/1EbAIdh

MongoDB is a document-oriented database that is easy to use from almost any language. As of August 2014, MongoDB is by far the most popular NoSQL database. This Refcard is designed to help you get the most out of MongoDB.

Apache Cassandra
bit.ly/1yTuwN6

Apache Cassandra is a high performance, extremely scalable, fault tolerant (i.e., no single point of failure), distributed non-relational database solution. Cassandra combines all the benefits of Google Bigtable and Amazon Dynamo to handle the types of database management needs that traditional RDBMS vendors cannot support.

Essential Couchbase APIs
bit.ly/1B0CCBH

Couchbase is an open source NoSQL database optimized for interactive web and mobile applications. The data model is JSON-based, so it's familiar, efficient, and extremely flexible. This Refcard offers a snippet-rich overview of connection, storage, retrieval, update, and query operations on Couchbase databases from Java, Ruby, and .NET applications.

TOP 3 DATABASE WEBSITES

High Scalability highscalability.com

Use the Index, Luke use-the-index-luke.com

ODBMS Industry Watch odbms.org/blog

TOP 4 DATABASE NEWSLETTERS

NoSQL Weekly nosqlweekly.com

DB Weekly dbweekly.com

Postgres Weekly postgresweekly.com

MySQL Weekly mysqlnewsletter.com

DATA MIGRATION CHECKLIST

Data migration is the process of transferring data from one system to another, which can mean changing the storage type, data formats, database, application or computer system. Most IT shops have to deal with data migration at some point, and making mistakes in a migration project can lead to a loss of data. This checklist is designed to help ensure that you don't miss any important steps in a general data migration project.

01. ASSESSMENT

- [ ] Establish a migration management team with key stakeholders
- [ ] Identify the data to be migrated
- [ ] Analyze the current data environment in comparison to the new environment
- [ ] Find out if any data format conversions will be necessary
- [ ] Determine how much data cleaning and manipulation is required
- [ ] Review data governance policies
- [ ] Have a data backup system in place
- [ ] Project how much storage capacity is required for the migration and if any data growth is expected
- [ ] Determine any other required tooling, costs, risks, and time frames associated with the migration

02. PLANNING

- [ ] Build a migration plan and timetable with the migration management team, and include some flexibility
- [ ] Acquire or build any tools necessary for migration
- [ ] Establish which permissions consultants or CRM vendors should have when working with your data (if applicable)
- [ ] Map the old database schema to the new database and prepare any data migration software or custom scripts
- [ ] Decide whether data will be cleaned before or after migration
- [ ] Notify users of any downtime
- [ ] Determine your organization's metrics for success in this migration
- [ ] Run tests with sample data to rehearse the data migration process
- [ ] Test administrators' knowledge on the new system's features (e.g. reporting, export)

03. ASSESSMENT

- [ ] Revise migration plan if necessary
- [ ] Configure migration environment
- [ ] Identify any inconsistencies, missing fields, or fields that need consolidation, conversion, or parsing
- [ ] Perform backup or replication of the data to be migrated

04. VALIDATION

- [ ] Test the data to ensure its integrity and full importation
- [ ] Verify that the database schema is correct
- [ ] Perform a rollback if data corruption occurs
- [ ] Prepare a report on the statistics related to the migration
- [ ] Business owners need to perform acceptance testing
- [ ] Check to see if your organization's goals for success were met
- [ ] Document the process steps if no prior documentation exists

*Authored with Ayobami Adewole

MIGRATION SOFTWARE

Below are a few existing tools for data migration. You'll have to choose based on OS environments, original and target hardware platforms, database vendors, and database versions. Key features you should look for include rollback and migration speed.

STORAGE MIGRATION TOOLS

- **Rsync:** Open source and provides fast incremental file transfer
- **IBM Softek Transparent Data Migration Facility:** Migrates data at the volume or block level, supports multiple platforms
- **SanXfer:** Migrates server data volumes and is optimized for cloud environments
- **Hitachi Data Systems Universal Replicator:** Provides storage array-based replication

DATABASE MIGRATION TOOLS

- **Microsoft SQL Server Migration Assistant (SSMA):** For migrating MySQL or Oracle database to SQL Server
- **MySQL Workbench Wizard:** For easy database migration to MySQL
- **Oracle SQL Developer:** For migrating non-Oracle databases to Oracle
- **EDB Migration Toolkit:** For database migration to PostgreSQL
- Other database migration tools include **Flyway**, **SwisSQL**, **ESF Database Migration Toolkit**, and **Ispirer's SQLways**

APPLICATION MIGRATION SOFTWARE TOOLS

Application vendors normally provide custom tools to migrate data from one application to the other.

SOLUTIONS DIRECTORY

Notice to readers: The data in this section is impartial and not influenced by any vendor.

This directory of database management systems and data grids provides comprehensive, factual data gathered from third-party sources and the tool creators' organizations. Solutions in the directory are selected based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability. The solution summaries underneath the product titles are based on the organization's opinion of its most distinguishing features. Understand that not having certain features is a positive thing in some cases. Fewer unnecessary features sometimes translates into more flexibility and better performance.

CLASSIFICATION IN-MEMORY, KV

Aerospike Server by Aerospike

Aerospike was designed to make specific use of SSDs without a filesystem, and features advanced peer-to-peer clustering with automated management.

STORAGE MODEL
Key-Value - SSD or Disk

LANGUAGE DRIVERS
C C# Go Java Node.js Ruby

TRANSACTIONS SUPPORTED
Compare-and-set

NON-SQL QUERY LANGUAGES
- AQL

ACID ISOLATION
- Read committed

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
Based on Open Source

BUILT-IN TEXT SEARCH?
No

TWITTER @AerospikeDB

WEBSITE aerospike.com

CLASSIFICATION IN-MEMORY, NEWSQL

Altibase HDB by Altibase

Altibase is a hybrid in-memory database, providing the speed of memory plus the storage of disk in one single, unified engine.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
C C++ C# Java PHP

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions on a single node

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Serializable/linearizable

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Complex Indexing

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
No

TWITTER @altibase

WEBSITE altibase.com

CLASSIFICATION GRAPH, DOCUMENT, KV

ArangoDB by ArangoDB

ArangoDB features a microservice framework, optional transactions, and integrates with cloud operating systems like Mesosphere and Docker Swarm.

STORAGE MODEL
Document, Key-Value, Graph

LANGUAGE DRIVERS
C# Go Java Node.js Ruby Python

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions on a single node

NON-SQL QUERY LANGUAGES
- AQL

ACID ISOLATION
- Serializable/linearizable

REPLICATION
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
Yes

TWITTER @arangodb

WEBSITE arangodb.com

CLASSIFICATION SPECIALIST ANALYTIC

Aster Database by Teradata

Teradata focuses on hadoop capabilities and multistructured formats, particularly for the data warehouse market.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
C# Java

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions on a single node

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Snapshot
- Serializable/linearizable

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
Yes

TWITTER @teradata

WEBSITE teradata.com/Teradata-Aster-Database/

CLASSIFICATION NEWSQL, KV DIRECT ACCESS

c-treeACE by FairCom Corporation

c-treeACE features a small footprint, scalability, and data integrity coupled with portability, flexibility, and service.

STORAGE MODEL
Relational, Key-Value, Wide Column/Big Table

LANGUAGE DRIVERS
C C++ C# Java Python PHP

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- CTDB
- JDTB
- ISAM

ACID ISOLATION
- Snapshot

REPLICATION
- Asynchronous

INDEXING CAPABILITIES
Complex Indexing

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
No

TWITTER @FairCom_Corp

WEBSITE faircom.com

CLASSIFICATION OBJECT-ORIENTED

Cache by InterSystems

Cache is scalable in terms of users and data, both vertically and horizontally, while running more efficiently and using less hardware.

STORAGE MODEL
Object Model

LANGUAGE DRIVERS
C C++ C# Java Node.js Python

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- MDX

ACID ISOLATION
- Serializable/Linearizable

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Complex Indexing

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
Yes

TWITTER @InterSystems

WEBSITE InterSystems.com

CLASSIFICATION KV, WIDE COLUMN

Cassandra

Apache Cassandra was originally developed at Facebook and is currently used at major tech firms like Adobe and Netflix.

STORAGE MODEL
Wide Column/Big Table

LANGUAGE DRIVERS
C# Go Java Node.js Ruby Python

TRANSACTIONS SUPPORTED
None

NON-SQL QUERY LANGUAGES
- CQL

ACID ISOLATION
- None

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
Via Solr and Datastax

TWITTER @cassandra

WEBSITE cassandra.apache.org

CLASSIFICATION NEWSQL

ClustrixDB by Clustrix

ClustrixDB offers seamless scale-out and built-in high availability, perfectly suited for e-commerce applications.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
C Java Ruby Python PHP

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Serializable/linearizable

REPLICATION
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
No

TWITTER @clustrix

WEBSITE clustrix.com

CLASSIFICATION KV, DOCUMENT, DATA CACHING

Couchbase Server by Couchbase

RESEARCH PARTNER

A memory-centric architecture, featuring an integrated cache, support for distributed caching, key-value storage, and document handling.

STORAGE MODEL
Document, Key-Value

LANGUAGE DRIVERS
C C++ C# Go Java Node.js

TRANSACTIONS SUPPORTED
Compare-and-set

NON-SQL QUERY LANGUAGES
- N1QL

ACID ISOLATION
- MVCC

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Complex Indexing

OPEN SOURCE
Based on Open Source

BUILT-IN TEXT SEARCH?
Via ElasticSearch and Solr

TWITTER @couchbase

WEBSITE couchbase.com

CLASSIFICATION KV, DBAAS

DynamoDB by Amazon

Amazon DynamoDB is a fully managed NoSQL database that supports both document and key-value data models for consistency and low latency.

STORAGE MODEL
Document, Key-Value - Ordered

LANGUAGE DRIVERS
Java Node.js Ruby Python Perl PHP

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- DSL

ACID ISOLATION
- Snapshot

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
Yes

TWITTER @awscloud

WEBSITE aws.amazon.com

CLASSIFICATION KV, GRAPH, DOCUMENT, NEWSQL

FoundationDB by FoundationDB

FoundationDB is a multi-model database that combines scalability, fault-tolerance, and high performance with ACID transactions.

STORAGE MODEL
Relational, Document, Ordered Key-Value, Graph

LANGUAGE DRIVERS
C C++ C# Go Java Node.js

TRANSACTIONS SUPPORTED
Multi-key ACID transactions

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Serializable/linearizable

REPLICATION
- Synchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
Via Lucene

TWITTER @FoundationDB

WEBSITE foundationdb.com

CLASSIFICATION IN-MEMORY, DATA GRID

Hazelcast by Hazelcast

Hazelcast features an in-memory data grid and distributed execution framework using Java standard APIs, with transparent distribution and full JCache support.

STORAGE MODEL
Key-Value - RAM

LANGUAGE DRIVERS
C++ C# Java REST

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Repeatable reads

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Complex Indexing

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
No

TWITTER @hazelcast

WEBSITE hazelcast.com

CLASSIFICATION WIDE COLUMN

HBase

HBase excels at random, real-time read/write access to very large tables of data atop clusters of commodity hardware.

STORAGE MODEL
Wide Column/Big Table

LANGUAGE DRIVERS
C C++ C# Java Python PHP

TRANSACTIONS SUPPORTED
None

NON-SQL QUERY LANGUAGES
- DSL

ACID ISOLATION
- N/A

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Primary Key

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
No

TWITTER @Hbase

WEBSITE hbase.apache.org

CLASSIFICATION RDBMS

Ingres by Actian

Ingres contains enhancements to easily internationalize customer applications with reduced configuration overhead, simple admin tools, and improved performance.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
Java .NET ODBC

TRANSACTIONS SUPPORTED
Compare-and-set, arbitrary multi-statement transactions spanning multiple nodes

NON-SQL QUERY LANGUAGES
- QUEL

ACID ISOLATION
- Serializable/linearizable

REPLICATION
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
Based on Open Source

BUILT-IN TEXT SEARCH?
Yes

TWITTER @ActianCorp

WEBSITE actian.com

CLASSIFICATION IN-MEMORY, HADOOP, DATA GRID

In-Memory Data Fabric by GridGain

GridGain works on top of existing databases. It supports key-value stores, SQL access, M/R, streaming/CEP, and Hadoop acceleration.

STORAGE MODEL
Key-Value - RAM

LANGUAGE DRIVERS
C++ C# Java Scala HTTP REST Memcached

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- Customer-definable

ACID ISOLATION
- Serializable/linearizable

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
Via Lucene

TWITTER @gridgain

WEBSITE gridgain.com/products/in-memory-data-fabric/

CLASSIFICATION RDBMS

InterBase by Embarcadero

Multi-device, fast, secure, admin free, small footprint scalable embeddable relational database for Windows, Linux, Mac OSX & iOS, Android.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
C C++ C# Java Delphi Python

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions on a single node

NON-SQL QUERY LANGUAGES
- GDML

ACID ISOLATION
- Snapshot
- Serializable/linearizable

REPLICATION
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
No

TWITTER @interbase

WEBSITE embarcadero.com/products/interbase

CLASSIFICATION RDBMS, MYSQL FAMILY

MariaDB by MariaDB

While remaining application compatible with MySQL, MariaDB adds many new capabilities to address challenging web and enterprise application use cases.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
C C++ C# Go Java Ruby

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions on a single node

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Snapshot

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
Yes

TWITTER @mariadb

WEBSITE mariadb.com

CLASSIFICATION IN-MEMORY, NEWSQL

MemSQL by MemSQL

MemSQL is a real-time databases for transactional analytics. It provides instant access to real-time and historical data through a familiar SQL interface.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
C C++ C# Go Java Node.js

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Read committed

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
No

TWITTER @memsql

WEBSITE memsql.com

CLASSIFICATION DOCUMENT

MongoDB by MongoDB

MongoDB blends linear scalability and schema flexibility with the rich query and indexing functionality of the RDBMS.

STORAGE MODEL
Document

LANGUAGE DRIVERS
C C++ C# Java Node.js Ruby

TRANSACTIONS SUPPORTED
Compare-and-set

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Read uncommitted

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
Yes

TWITTER @mongodb

WEBSITE mongodb.org

CLASSIFICATION RDBMS

MySQL Community Edition by Oracle

MySQL Community Edition is the free downloadable version of the popular open source database, supported by a large community of developers.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
C C++ C# Java Ruby Python

TRANSACTIONS SUPPORTED
None

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- N/A

REPLICATION
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
Yes

TWITTER @mysql
WEBSITE mysql.com

CLASSIFICATION GRAPH

Neo4j by Neo Technology

Neo4j is a purpose-built graph database designed to unlock business value in real time from data and its relationships.

STORAGE MODEL
Graph

LANGUAGE DRIVERS
Go Java Node.js Ruby Python Perl

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions on a single node

NON-SQL QUERY LANGUAGES
- Cypher

ACID ISOLATION
- Serializable/linearizable

REPLICATION
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
Yes

TWITTER @neo4j
WEBSITE neo4j.com

CLASSIFICATION NEWSQL

NuoDB by NuoDB

RESEARCH PARTNER

NuoDB is a scale-out SQL database with on-demand scalability, geo-distributed deployment and peer-to-peer distributed architecture.

STORAGE MODEL
Relational, Key-Value, Graph

LANGUAGE DRIVERS
C C++ C# Go Java Node.js

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions on a single node

NON-SQL QUERY LANGUAGES
- RDF

ACID ISOLATION
- Serializable/linearizable

REPLICATION
- Asynchronous

INDEXING CAPABILITIES
Primary Key

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
No

TWITTER @nuodb
WEBSITE nuodb.com

CLASSIFICATION RDBMS, GRAPH, DOCUMENT

Oracle Database by Oracle

Oracle Database features a multitenant architecture, automatic data optimization, defense-in-depth security, high availability, clustering, and failure protection.

STORAGE MODEL
Relational, Document, Graph

LANGUAGE DRIVERS
C C++ C# Go Java Node.js

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- CQL
- XQuery

ACID ISOLATION
- Snapshot
- Serializable/linearizable

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
Yes

TWITTER @oracledatabase
WEBSITE oracle.com

CLASSIFICATION RDBMS, DOCUMENT, GRAPH

OrientDB by Orient Technologies

OrientDB is a multi-model NoSQL DBMS that combines graphs, documents, and key-value into one scalable, high performance product.

STORAGE MODEL
Document, Ordered Key-Value, Graph

LANGUAGE DRIVERS
PHP Python C# Java Node.js Ruby

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- Gremlin
- SPARQL
- Orient SQL

ACID ISOLATION
- Serializable/linearizable

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Complex Indexing

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
Via Lucene

TWITTER @OrientDB

WEBSITE orientechnologies.com

CLASSIFICATION RDBMS, MYSQL FAMILY

Percona Server by Percona

Percona server is an open source drop-in replacement for MySQL with high performance for demanding workflows.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
C C++ C# Go Java Node.js

TRANSACTIONS SUPPORTED
Single node and master-to-slave

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Snapshot
- Serializable/linearizable

REPLICATION
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
Yes

TWITTER @percona

WEBSITE percona.com

CLASSIFICATION RDBMS

Postgres Plus Advanced Server by EnterpriseDB

Postgres Plus Advanced Server integrates enterprise-grade performance, security and manageability enhancements into PostgreSQL.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
C++ C# Java Node.js Ruby Python

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions on a single node

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Serializable/linearizable
- Read committed
- Repeatable Read

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Complex Indexing

OPEN SOURCE
Based on Open Source

BUILT-IN TEXT SEARCH?
Yes

TWITTER @enterprisedb

WEBSITE enterprisedb.com

CLASSIFICATION DOCUMENT

RavenDB by Hibernating Rhinos

RavenDB features multi-master replication, dynamic queries, strong support for reporting, and self optimizing.

STORAGE MODEL
Document, Key-Value - SSD or Disk

LANGUAGE DRIVERS
C# Java Node.js Ruby Python PHP

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- Lucene

ACID ISOLATION
- Snapshot

REPLICATION
- Asynchronous

INDEXING CAPABILITIES
Complex Indexing

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
Yes

TWITTER @RavenDB

WEBSITE ravendb.net

CLASSIFICATION IN-MEMORY, KV, DATA CACHING

Redis Labs Enterprise Cluster by Redis Labs

Redis Labs Enterprise Cluster allows users to create and manage scalable, highly available Redis databases on-premises.

STORAGE MODEL
Key-Value - RAM

LANGUAGE DRIVERS
C C++ C# Go Java Node.js

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Serializable/linearizable

REPLICATION
- Asynchronous

INDEXING CAPABILITIES
None

OPEN SOURCE
Based on Open Source

BUILT-IN TEXT SEARCH?
No

TWITTER @RedisLabsInc

WEBSITE redislabs.com

CLASSIFICATION DOCUMENT, KV

Riak by Basho

Riak is a highly available, highly scalable, fault-tolerant distributed NoSQL database.

STORAGE MODEL
Key-Value - Eventually Consistent

LANGUAGE DRIVERS
C# Erlang Python Ruby Java Node.js

TRANSACTIONS SUPPORTED
None

NON-SQL QUERY LANGUAGES
- MapReduce

ACID ISOLATION
- None, BASE system

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
Yes

TWITTER @basho

WEBSITE basho.com

CLASSIFICATION IN-MEMORY, COLUMN-ORIENTED RDBMS

SAP HANA Platform by SAP

SAP HANA is an in-memory, column-oriented ACID database and application platform for spatial, predictive, text, graph, and streaming analytics.

STORAGE MODEL
Relational, Column-oriented, Graph

LANGUAGE DRIVERS
C C++ C# Go Java Node.js

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- Odata
- XMLA

ACID ISOLATION
- Snapshot
- Serializable/linearizable

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Key, Complex Indexing

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
Yes

TWITTER @SAPinMemory

WEBSITE hana.sap.com

CLASSIFICATION IN-MEMORY, HADOOP, DATA GRID

ScaleOut StateServer by ScaleOut Software

ScaleOut's architecture integrates a low latency in-memory data grid and a data-parallel compute engine. Ease-of-use is built into all functions.

STORAGE MODEL
Key-Value - Ordered and RAM

LANGUAGE DRIVERS
C C++ C# Java

TRANSACTIONS SUPPORTED
Compare-and-set

NON-SQL QUERY LANGUAGES
- Microsoft LINQ

ACID ISOLATION
- Serializable/linearizable

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Complex Indexing

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
No

TWITTER @ScaleOut_Inc

WEBSITE scaleoutsoftware.com

CLASSIFICATION NEWSQL, HADOOP

Splice Machine by Splice Machine

Splice Machine combines the scalability of Hadoop, the ANSI SQL and ACID transactions of an RDBMS, and the distributed computing of HBase.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
Java

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Snapshot

REPLICATION
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
Via Solr

TWITTER @splicemachine

WEBSITE splicemachine.com

CLASSIFICATION RDBMS

SQL Server by Microsoft

SQL Server is considered the de facto database for .NET development. Its ecosystem is connected to numerous .NET technologies.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
C# Java Ruby PHP Visual Basic

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions on a single node

NON-SQL QUERY LANGUAGES
- Transact-SQL
- MDX

ACID ISOLATION
- Snapshot
- Serializable/linearizable

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Primary and Secondary Keys

OPEN SOURCE
No

BUILT-IN TEXT SEARCH?
Yes

TWITTER @microsoft

WEBSITE microsoft.com/en-us/server-cloud/products/sql-server/Buy.aspx

CLASSIFICATION RDBMS

SQLite

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
C C++ C# Go Java Ruby

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Serializable/linearizable

REPLICATION
- No replication

INDEXING CAPABILITIES
Primary Key

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
Yes

TWITTER N/A

WEBSITE sqlite.org

CLASSIFICATION IN-MEMORY, NEWSQL

VoltDB by VoltDB

VoltDB combines the capabilities of an operational database, real-time analytics, and stream processing in one platform.

STORAGE MODEL
Relational

LANGUAGE DRIVERS
C++ C# Go Java Node.js Python

TRANSACTIONS SUPPORTED
Arbitrary multi-statement transactions spanning arbitrary nodes

NON-SQL QUERY LANGUAGES
- None

ACID ISOLATION
- Serializable/linearizable

REPLICATION
- Synchronous
- Asynchronous

INDEXING CAPABILITIES
Complex Indexing

OPEN SOURCE
Yes

BUILT-IN TEXT SEARCH?
No

TWITTER @voltdb

WEBSITE voltdb.com

glossary

a

Aggregate A cluster of domain objects that can be treated as a single unit. An ideal unit for data storage on large distributed systems.

ACID (Atomicity, Consistency, Isolation, Durability) A term that refers to the model properties of database transactions, traditionally used for SQL databases.

b

BASE (Basic Availability, Soft State, Eventual Consistency) A term that refers to the model properties of database transactions, specifically for NoSQL databases needing to manage unstructured data.

B-Tree A data structure in which all terminal nodes are the same distance from the base, and all nonterminal nodes have between *n* and 2*n* subtrees or pointers. It is optimized for systems that read and write large blocks of data.

c

Complex Event Processing An organizational process for collecting data from multiple streams for the purpose of analysis and planning.

d

Database Clustering Connecting two or more servers and instances to a single database, often for the advantages of fault tolerance, load balancing, and parallel processing.

Data Management The complete lifecycle of how an organization handles storing, processing, and analyzing datasets.

Data Mining The process of discovering patterns in large sets of data and transforming that information into an understandable format.

Database Management System (DBMS) A suite of software and tools that manage data between the end user and the database.

Data Warehouse A collection of accumulated data from multiple streams within a business, aggregated for the purpose of business management.

Distributed System A collection of individual computers that work together and 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.

Document Store A type of database that aggregates data from documents rather than defined tables and is used to present document data in a searchable form.

e

Eventual Consistency The idea that databases conforming to the BASE model will contain data that becomes consistent over time.

f

Fault-Tolerance A system's ability to respond to hardware or software failure without disrupting other systems.

g

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 and location services.

h

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, and other specific strategies.

i

In-Memory As a generalized industry term, it describes data management tools that load data into memory (RAM or flash memory) instead of hard disk drives.

j

Journaling Refers to the simultaneous, real-time logging of all data updates in a database. The resulting log functions as an audit trail that can be used to rebuild the database if the original data is corrupted or deleted.

k

Key-Value Store A type of database that stores data in simple key-value pairs. They are used for handling lots of small, continuous, and potentially volatile reads and writes.

l

Lightning Memory-Mapped Database (LMDB) A copy-on-write B-Tree database that is fully transactional, ACID compliant, small in size, and uses MVCC.

Log-Structured Merge (LSM) Tree A data structure that writes and edits data using immutable segments or runs that are usually organized into levels. There are several strategies, but the first level commonly contains the most recent and active data.

m

MapReduce A programming model created by Google for high scalability and distribution on multiple clusters for the purpose of data processing.

Multi-Version Concurrency Control (MVCC) A method for handling situations where machines simultaneously read and write to a database.

n

NewSQL A shorthand descriptor for relational database systems that provide horizontal scalability and 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 follow standard relational database rules.

o

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.

p

Persistence Refers to information from a program that outlives the process that created it, meaning it won't be 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.

r

Relational Database A database that structures interrelated datasets in tables, records, and columns.

Replication A term for the sharing of data so as to ensure consistency between redundant resources.

s

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

Sharding Also known as "horizontal partitioning," sharding is where a database is split into several pieces, usually to improve the speed and 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 and manipulating data; used primarily in relational databases.

w

Wide-Column Store Also called "BigTable stores" because of their relation to Google's early BigTable database, these databases store data in records that can hold very large numbers of dynamic columns. The column names and the record keys are not fixed.

DZONE RESEARCH GUIDES

"Better, more concrete, and closer to reality than reports from Gartner and Forrester..."

Robert Zakrzewski, DZone Member

DZONE GUIDE TO
BIG DATA

Discover data science obstacles and how the influx of data is influencing new tools and technologies.

DOWNLOAD

DZONE GUIDE TO
CONTINUOUS DELIVERY

Understand the role of containerization, automation, testing, and other best practices that move Continuous Delivery beyond simply an exercise in tooling.

DOWNLOAD

DZONE GUIDE TO
ENTERPRISE INTEGRATION

Expand your knowledge on integration architecture, REST APIs and SOA patterns.

DOWNLOAD

UPCOMING GUIDES IN 2015:

- Cloud Development
- Mobile Development
- Internet of Things
- Performance & Monitoring
- Software Quality
- Team Collaboration

Get them all for free on DZone! dzone.com/research