

LEARNING MADE EASY

Yugabyte Special Edition

Distributed SQL Databases

for
dummies[®]
A Wiley Brand



Discover the benefits
of distributed SQL

—
Identify where NoSQL
falls short

—
Find use cases that
need a new approach

Brought to you
by

 **yugabyteDB**

Floyd Smith

About Yugabyte

YugabyteDB is the 100 percent open-source, cloud-native distributed SQL database for mission-critical applications. It allows businesses to accelerate productivity, reduce costs, and lower risks without vendor lock-in.

The first — and only — open-source, hybrid, multi-cloud, distributed SQL database on the planet, [YugabyteDB](#) delivers a modern data layer. Because it's built to support existing PostgreSQL-compatible applications, new microservices, and cloud-native apps, you no longer need to compromise on performance, consistency, availability, or scalability.

YugabyteDB delivers enterprise-grade RDBMS capabilities like distributed ACID transactions and replication with strong consistency, along with the horizontal scalability and resilience of nonrelational databases. It can be deployed in public, hybrid, and private clouds, as well as in physical, virtual, or container environments within minutes.

In addition to YugabyteDB, we offer three additional Enterprise-focused technology solutions:

- [YugabyteDB Anywhere](#): YugabyteDB delivered as a private database-as-a-service for enterprises
- [YugabyteDB Managed](#): Fully managed YugabyteDB-as-a-Service
- [YugabyteDB Voyager](#): Simplified data migration from legacy databases and cloud databases to YugabyteDB

Fortune 500 companies like Kroger, Wells Fargo, and GM have already adopted YugabyteDB to help them modernize their data layers. This allows them to expedite new innovations, eliminate expensive legacy solutions, and enable a resilient, always-on business, free from vendor and cloud lock-in.

We hope you enjoy the book! Thank you for your interest in distributed SQL databases.



Distributed SQL Databases

Yugabyte Special Edition

by Floyd Smith

for
dummies[®]
A Wiley Brand

Distributed SQL Databases For Dummies®, Yugabyte Special Edition

Published by

John Wiley & Sons, Inc.

111 River St.

Hoboken, NJ 07030-5774

www.wiley.com

Copyright © 2023 by John Wiley & Sons, Inc., Hoboken, New Jersey

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the Publisher. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Trademarks: Wiley, For Dummies, the Dummies Man logo, The Dummies Way, Dummies.com, Making Everything Easier, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc., is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: WHILE THE PUBLISHER AND AUTHORS HAVE USED THEIR BEST EFFORTS IN PREPARING THIS WORK, THEY MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES REPRESENTATIVES, WRITTEN SALES MATERIALS OR PROMOTIONAL STATEMENTS FOR THIS WORK. THE FACT THAT AN ORGANIZATION, WEBSITE, OR PRODUCT IS REFERRED TO IN THIS WORK AS A CITATION AND/OR POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE PUBLISHER AND AUTHORS ENDORSE THE INFORMATION OR SERVICES THE ORGANIZATION, WEBSITE, OR PRODUCT MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING PROFESSIONAL SERVICES. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR YOUR SITUATION. YOU SHOULD CONSULT WITH A SPECIALIST WHERE APPROPRIATE. FURTHER, READERS SHOULD BE AWARE THAT WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ. NEITHER THE PUBLISHER NOR AUTHORS SHALL BE LIABLE FOR ANY LOSS OF PROFIT OR ANY OTHER COMMERCIAL DAMAGES, INCLUDING BUT NOT LIMITED TO SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR OTHER DAMAGES.

ISBN 978-1-394-16889-7 (pbk); ISBN 978-1-394-16890-3 (ebk)

For general information on our other products and services, or how to create a custom *For Dummies* book for your business or organization, please contact our Business Development Department in the U.S. at 877-409-4177, contact info@dummies.biz, or visit www.wiley.com/go/custompub. For information about licensing the *For Dummies* brand for products or services, contact BrandedRights&Licenses@Wiley.com.

Publisher's Acknowledgments

Some of the people who helped bring this book to market include the following:

Project Editor: Elizabeth Kuball

Acquisitions Editor: Ashley Coffey

Editorial Manager: Rev Mengle

Client Account Manager:
Cynthia Tweed

Production Editor:

Tamilmani Varadharaj

Special Help: Rachel Pescador,
Michael Haag, Suda Srinivasan

Introduction

Data-driven organizations need an agile, cloud-native database that matches their modern cloud infrastructure and application architecture. The new data layer must be scalable, highly available, secure, and flexible, without locking the organization in to a single vendor.

About This Book

In this book, I describe how monolithic databases of the past fall short of the challenges of today's cloud-native applications. Initial responses such as NoSQL databases, which lack structure, and NewSQL databases, which aren't fully scalable, have failed to meet the challenge. Only a new approach, distributed SQL, can help organizations take full advantage of the new capabilities available in the cloud. You'll see the benefits of this new approach and how it meets the needs of leading industry segments across a wide range of use cases.

Icons Used in This Book

Throughout the book, I use icons to indicate special information. Here's a guide to what those icons mean:



TIP

The Tip icon indicates information you can apply to your own projects to make them work better, whatever technology you have at hand. Tips can save you time and help you avoid frustration.



REMEMBER

The Remember icon highlights information that's worth retaining after you've put down this book.



TECHNICAL
STUFF

The Technical Stuff icon gives you detailed information related to a particular topic. Information marked by this icon isn't necessary to getting the job done, but it provides depth and interest.

Beyond the Book

This book introduces you to distributed SQL and shows you how you can use it to enable critical applications in your organization for the cloud. If you want resources beyond what this book offers, here's some insight for you:

- » **“Migrating from Monolithic to Cloud Native Operational Databases”** (<https://info.yugabyte.com/2022-white-paper-migrating-from-monolithic-to-cloud-native-1p>): A white paper from Yugabyte that describes how new software development and delivery models require new database technology
- » **“Distributed SQL vs. NewSQL”** (www.yugabyte.com/blog/distributedsql-vs-newsq1/): A blog post from Yugabyte that describes how NewSQL falls short of meeting the challenges of cloud-native applications, while distributed SQL meets them
- » **“Why Are NoSQL Databases Becoming Transactional?”** (www.yugabyte.com/blog/nosql-databases-becoming-transactional-mongodb-dynamodb-faunadb-cosmosdb/): A blog post from Yugabyte that explains why NoSQL databases are trying to add transaction capabilities and describes how distributed SQL does that job better

- » Understanding data-first business needs
- » Speeding digital transformation
- » Defining the modern data stack

Chapter 1

The Rise of the Modern Data-First Enterprise

Enterprises build and deploy applications differently today compared to a decade ago. Proprietary scale-up computing infrastructure has given way to scale-out commodity servers in the cloud. Microservices have replaced monolithic applications. Waterfall development and organizational silos are out; agile development and DevOps are in.

Why? Because you need to innovate faster to seize new opportunities ahead of your competitors. These trends, which have been underway for several years, accelerated during the pandemic and with the rise of digital-first initiatives.

Cloud infrastructure and cloud-native applications promise speed and agility like never before. You can now provision servers in a matter of minutes rather than days, and build and release your application in a fraction of the time it used to take. This has enabled businesses to go from shipping one big release every few months to delivering several new releases daily.

While significant progress and investments have been made in the digital transformation of applications and infrastructure, one part of the tech stack has remained largely unchanged: the transactional database.

Cloud-native applications still rely on traditional monolithic databases for their systems of record and engagement. These databases, designed before the cloud era, were not architected for the demands of modern applications. They take days or weeks to provision and configure. Scaling entails manual data *sharding* (dividing the data up into smaller chunks using a database field designated as the shard key) or deploying a cache in front of the database and dealing with coherence issues. Resilience requires bolt-on replication solutions. Geo-distributing the database for compliance or performance is a nonstarter. The result: expensive trade-offs, slow innovation, complex operations, and poor customer experience.

Modern applications and microservices require a modern cloud-native transactional database architected with the needs of these applications in mind. In this chapter, I describe the database requirements of today's applications.

Understanding What Modern Data-First Businesses Need

Businesses today are increasingly data-first. They understand that delivering new services and improving their customer experiences depends on capturing and delivering accurate and fast data. They start with a data architecture that considers how data can be quickly, efficiently, and securely captured, stored, and used. They implement this strong data foundation before they build applications that use the data.

Why is data a top priority for these businesses? Transactional data is at the heart of customers' experiences in every industry. Retailers depend on mobile app shopping carts and real-time inventories to help customers make purchasing decisions. Financial services companies need to support exponential increases in micropayments, constant checking of account balances and transactions, and a geographically distributed customer base. Essentially, transactional data is at the core of what these companies must do to meet their business goals.

The distributed applications using this data empower their customers, streamline operations, and support fast innovation. Those applications are increasingly based on microservices running in containers and Kubernetes across private and public clouds. And they have requirements that can't be met by monolithic databases or eventually consistent alternatives like NoSQL. Modern applications require the following six key characteristics.

High performance

Modern applications require a database that delivers high performance along with the deployment flexibility of consistent transactions across clouds. Providing high performance requires support for the following:

- » High ingest rates (for example, when bringing in streaming data from external sensors or other sources)
- » Fast transactions, allowing applications to experience low latency on data-layer operations
- » Fast data transfers out of the database (for example, when delivering query results)

For business-critical enterprise applications, it's not uncommon for a modern database to be expected to handle 1 million transactions or more per second, while supporting thousands of concurrent connections.

Massive scaling on demand

As application demands surge, which can happen at unexpected times in today's viral world, the database needs to scale quickly without incurring any downtime.

In the world of transactional databases, scaling can mean being able to process a greater number of transactions in parallel or storing more data. Traditional vertical scaling is costly and complex. In a cloud-native world, we need to scale seamlessly by adding new servers (compute and storage resources) and letting the system intelligently rebalance across the cluster. And then if the demand subsides, the database should be able to shrink back to save costs.

Continuous availability

Given the criticality of the transactional database for applications, it's essential that these databases remain available, surviving just about any disaster scenario. And in the case of such an event, they need to recover quickly if the server they're running on goes down. Every minute a database is down can mean millions of dollars of lost revenue and poor customer experience.

Deployment flexibility

Modern applications require a consistent data layer that provides a wide range of deployment flexibility, including

- » The ability to run in public cloud, private cloud, and hybrid cloud environments
- » The ability to run in containers (the modern option), on virtual machines, or on bare metal
- » The ability to run in any environment based on the most widely used container orchestration software, Kubernetes
- » Support for a wide range of replication and geo-distribution options



TIP

By meeting these requirements, a database can support any application, whether internal to the business or customer-facing, legacy or modern, entirely designed and supported by the business or composed of one or more external components.

Standard interface

No one wants to learn another proprietary language or interface just so they can talk to a new database. Modern databases need to offer interfaces that are compatible with familiar and widely used database application programming interfaces (APIs).

PostgreSQL is the most popular choice for compatibility because it is widely used by enterprises and is also open source. As a result, PostgreSQL features a rich ecosystem of compatible frameworks, applications, drivers, and tools. The best PostgreSQL-compatible databases are not only *wire compatible* (you can communicate with the database using PostgreSQL client drivers) but also *feature compatible* (the database supports all the features of PostgreSQL including advanced features like triggers, partial indexes, and stored procedures).

A modern, data-first business requires its data layer to be plug-and-play with all these different components needed for successful application development and delivery. Offering PostgreSQL compatibility allows developers to be instantly productive on the new database.



Compatibility across a limited set of functionalities is not true compatibility. A truly PostgreSQL-compatible database must also support advanced relational database management system (RDBMS) features, such as triggers, functions, stored procedures, and strong secondary indexes.

A security-first mindset

Security is crucial to all modern applications throughout development and delivery. This is especially true across the data layer, comprising not only data at rest, but data in transit, as required to complete database functions.

A modern database must be built from the ground up with data security in mind so that organizations can maintain a robust security posture across an increasingly distributed footprint.

Data must be encrypted at rest and in flight. The database must support multi-tenancy and encrypt data separately per tenant. And the data has to reside in specified geographic regions to meet compliance requirements and to support geographically based access controls.

DO TRADITIONAL DATABASES MEET MODERN REQUIREMENTS?

Traditional databases were developed before the public cloud and cloud-native technologies like containers and Kubernetes were widely used.

As a result, they often work well enough for small and new projects but fail to deliver the scale, flexibility, and simplicity expected of cloud-native solutions.

(continued)

(continued)

Developers and organizations with a large set of legacy databases face a tough decision: At what point does the complexity and costs of their existing environment reach a point where modernization is essential to continued growth and innovation? And do they stay with a newer, proprietary database or cloud-specific database, or investigate a modern, open solution designed with today's (and tomorrow's) needs in mind?

More developers and organizations are looking for a way to evolve to a modern distributed database that supports familiar languages, like PostgreSQL, on a distributed, cloud-native architecture. Newer green-field projects have learned that starting with a modern data layer is the best choice for modern applications.

Accelerating Digital Transformation

Digital transformation is real. Today, companies from small and medium-size businesses, to start-ups, and up through the enterprise are migrating mission-critical applications to a cloud-native IT stack:

- » **Old apps:** Legacy, monolithic application architectures running in on-premises data centers
- » **New apps:** Applications running on modern microservices architectures using containers and Kubernetes, along with other cloud-native infrastructure technologies, and running on public, private, or hybrid cloud infrastructure

Learning new, fast-changing technologies for new application development and delivery, and moving selected legacy applications to the new platform, is hard work.

The data layer has often been an impediment to developing new applications and migrating existing ones.



TECHNICAL
STUFF

The data layer is stateful by necessity, because persistent data is the very definition of *state*. Aligning this stateful data layer with purposefully stateless application architectures based on containers and Kubernetes is uniquely challenging.

The data layer comprises a persistent database and an in-memory cache in front of it, accelerating writes and making the most frequently accessed data available much faster than if all data were confined to disk.

Accelerating the digital transformation process requires a new approach to the data layer. A modern data stack must match and serve the modern application stack.

Distributed SQL: The Modern Transactional Database

What does a modern transactional database serving the needs of cloud-native application architectures look like?

The functional characteristics of this new database should match those of modern application architectures, so the desired benefits from modernization should extend to every part of the application, all through application development — which, in the modern IT world, is continuous — and during application delivery as well.

This is a best-of-both-worlds arrangement: The application consumes data services that deliver familiar capabilities such as relational data modeling and RDBMS features. But the application and a distributed SQL data layer share all the best characteristics of modern architectures:

- » **Resilient:** A data layer based on distributed SQL is, by nature, resilient because it isn't dependent on any single server, and failover and other resiliency features are easily supported.
- » **Continuously available:** Both architecture and operations of the distributed SQL data layer are carried out to support continuous availability (for example, by always having two or more copies of data within the layer).
- » **Horizontally scalable:** The data layer scales by simply adding servers to the cluster, just as the application architecture does.
- » **Strongly consistent:** Core transactional applications require strongly consistent data that delivers full ACID properties.



TIP

Eventual consistency, the standard for NoSQL databases, complicates app development and has a major impact on data accuracy.

ACID stands for atomicity, consistency, isolation, and durability.

- » **Geographically distributed:** Resources are geographically distributed when needed for resiliency and located near each other when needed for performance, governance, or other reasons.
- » **SQL-compatible and RDBMS-feature compatible:** Unlike with NoSQL, developers and the organization no longer must give up on relational data modeling, the use of the SQL query language, and other long-established RDBMS features.
- » **Hybrid and multi-cloud ready:** The data layer allows for deployment on-premises, on a cloud of choice, and/or on multiple clouds, avoiding lock-in.

- » Describing traditional database problems
- » Addressing forced trade-offs and segmentation
- » Identifying challenges with NoSQL and NewSQL

Chapter 2

Where Legacy Transactional Databases Fall Short

Monolithic relational database management systems (RDBMSs) were designed for a world before data processing systems were distributed. Both computation and database operations took place on a single (typically proprietary) system as modern cloud and cloud-native architectures were far from being a reality.

If an application needed more transaction processing or data storage capacity from the database, the only way to scale was to move the database to a server with more resources (central processing unit [CPU], memory, storage).

The problem with this “scale-up” approach is that there are limits to the speed and capacity of any single computer or server.

These limits increase as technology advances, but data and application growth have recently exceeded the speed of these technological advances. Applications or services can go viral in a matter of days, and if and when that inflection point hits is impossible

to predict. In enterprise environments with hundreds of applications, it's hard to know which applications will require massive scale and which will not.

To tackle this uncertainty, distributed systems were designed and developed in part to deliver rapid scale and agility. This happened first for computation because the challenges involved are easier to solve. This shift accelerated the availability of inexpensive, virtually unlimited compute instances in public cloud services that could be quickly spun up on demand.

Supporting distributed database operations is much more challenging because of the need to maintain data consistency across different systems while still delivering acceptable performance. Creating a single, logical ACID-compliant database out of distributed storage and compute resources is a significant challenge (www.yugabyte.com/tech/distributed-acid-transactions). (Even keeping storage consistent on a single machine presents challenges; using multiple machines, only more so.)



REMEMBER

ACID stands for atomicity, consistency, isolation, and durability.

As database technology has advanced, partial solutions to the need for distributed databases have appeared. These partial solutions are called NoSQL databases, which are in widespread use, and NewSQL databases, which are more of a specialty category that adds limited distribution capabilities to existing legacy architectures.

This chapter describes all these database categories in some detail. I show why they fall short in meeting all the needs of modern applications and how those gaps inspired a new approach to databases, which became the groundwork for distributed SQL. This modern approach combines the benefits of both traditional relational database management systems (RDBMSs) and NoSQL databases.

Describing the Problems with Traditional Databases

Traditional databases fall into three categories, each of which has advantages and disadvantages. However, none of them meets the needs of today's organizations for a scalable cloud-native data

layer that matches up well with today's scalable cloud-native application software. (Cloud-native software is defined by several key characteristics, but horizontal scalability may be the most important.)



TECHNICAL
STUFF

Certain database updates are called *transactions*, and they involve adding or changing a record in a reliable manner so that all applications that access the database can count on the retrieved data as being 100 percent accurate and current. The most reliable transactions are atomic, consistent, isolated, and durable (referred to as the ACID properties).

Traditional Structured Query Language (SQL) databases, described next, support ACID transactions; other kinds may or may not.



REMEMBER

Here are the three traditional types of databases:

- » **Traditional SQL:** Traditional SQL databases are, by definition, relational. They fully support SQL and ACID transactions. Traditional SQL databases only scale vertically (a single powerful server), not horizontally (scaling by adding more servers). To scale a traditional SQL database beyond the limits of a single server, enterprise engineering teams must deploy a cache in front of the database or manually shard the database. (*Sharding* is breaking up a database table into smaller fragments stored in different databases and then logically stitched together.) Both these options are expensive and fragile. Traditional SQL databases aren't inherently resilient either. To achieve high availability, you need to deploy a bolt-on replication solution.
- » **Traditional NoSQL:** NoSQL databases gained horizontal scalability and the ability to support multiple data models by abandoning relational structure, SQL, and ACID transactions. They're inherently resilient to failures in the underlying infrastructure. They're useful for a subset of database-related problems, but they're weak for transactions, which tend to be inconsistent, and for queries, due to the lack of support for the speed and power that accompanies support for SQL.
- » **NewSQL:** NewSQL is an initial effort toward supporting horizontal scalability in a database that uses a relational data model and benefits from support for the speed and power of SQL. However, NewSQL does not support multiple

models, only relational; has low availability, even though it's horizontally scalable; and, like the other traditional databases, is not fully cloud native. Behind the scenes, NewSQL solutions are still based on the legacy relational database architecture.

Battling Forced Trade-offs and Fragmentation

The evolution of databases to include traditional SQL, traditional NoSQL, and NewSQL caused a lot of organizational pain. Each approach had some advantages for a certain subset of applications. But the result is that many organizations now support a plethora of databases, none of which are truly optimal.

For example, NoSQL databases share some attractive characteristics: horizontal scalability, high-speed write performance, flexible schemas, and high availability. This causes organizations to adopt different NoSQL databases that share these characteristics, each with its strengths for different use cases.

And of course, organizations must retain and expand their base of traditional SQL databases because they need reliable ACID transactions and fast queries that go with SQL support.

However, this means that organizations need to support a complex mix of SQL and NoSQL databases. Every new database an organization adopts has a learning curve; the need for organizational expertise in that database; organizational relationships with companies and standards bodies involved in that specific database; and hiring and retention of expert personnel, among other considerations.

When a problem arises with a complex application, and one or more databases are involved, finding which database type is involved and the developers and operations people who can fix the problem, can be a major headache. Agility is hindered for the development and delivery of applications.



TIP

When choosing which database to use, consider how you'll find and solve problems when they arise, including whether your staff have the required experience.

Identifying Why NoSQL and NewSQL Don't Solve the Problem

Both NoSQL and NewSQL have strong proponents who will claim that every problem associated with a specific database in their category, or with that category as a whole, can be solved. So, it's important to understand why this is not the case.

Challenges with NoSQL

NoSQL has been around longer than NewSQL and has a much wider and richer array of database options, each with its own fans. At the same time, there has been consolidation in this market, with some database types and the companies associated with them, falling by the wayside.

The core problem with NoSQL databases goes back to their origin in 2006. They were originally optimized for the scalability of writing data while sacrificing (a) up-to-the-moment consistency, (b) complex access patterns to support ACID transactions and the full range of SQL queries, or (c) both.

The most valuable data was committed to SQL databases; less valuable data that came in fast and in large volumes went to NoSQL databases.

Applications based on NoSQL databases could still be valuable; for instance, Google Search is undoubtedly a valuable application, and NoSQL was originally developed partly to power it.

But the most recent value of any specific data item used for Google Search is not indispensable, and there isn't a need for transactions or reliable updates. Because search is a well-established problem, a small number of search types can be optimized without the need for the power and flexibility of SQL queries.

However, there are only so many applications where individual data updates are not highly valued and that don't need to support a broad range of fast, reliable queries. There are ongoing efforts to add ACID transactions to NoSQL databases and approximate SQL on the query side. The value of relational databases is increasingly appreciated by all concerned. Even Google Search is increasingly relying on relational databases.

Challenges with NewSQL

NewSQL databases were an effort to meet the need for scalability of relational databases (that is, those that support SQL queries). There are two types of NewSQL databases:

- » **Sharded:** A SQL database can support limited scalability by adding a sharding layer to manage independent servers, each holding a portion of a larger database. The sharding layer handles some challenges well but is less agile than a truly distributed database.
- » **Distributed engine:** A SQL database can be created that has a fully distributed storage engine. These databases are closer to the goal of a scalable relational database. But they don't support the additional data models introduced by NewSQL, and availability is low because these databases don't achieve the global elasticity required by modern cloud infrastructure.

Each NewSQL database is weak in one or more areas that are desirable, or even required, by specific applications:

- » **Transactional capability:** Each NewSQL database chooses from MySQL compatibility, PostgreSQL compatibility, or a proprietary approach. Each is limited on one or more capabilities, such as distributed transactions, JOINS across shards, and so on.
- » **Geographic data distribution:** NewSQL databases are mostly limited in their support for data distribution across multiple geographies.
- » **High performance:** NewSQL databases tend to be limited in performance and don't scale well horizontally, with performance gain per server dropping as more servers are added.
- » **Kubernetes-native:** NewSQL databases were mainly created before the rise of Kubernetes in the late 2010s and tend not to work natively with Kubernetes.

The world has evolved beyond the capabilities of NewSQL databases. They were an effort to supercharge the existing relational database architecture with some horizontal scalability, but they're not as scalable as the application software running in today's (or tomorrow's) cloud and they don't offer the flexibility provided by NoSQL.

- » Understanding the appeal of distributed SQL
- » Looking under the architecture hood
- » Comparing distributed SQL to other databases

Chapter 3

Distributed SQL: Rethinking Transactional Databases

Modern cloud-native applications and data processing demands have exposed the limitations of traditional SQL, NoSQL, and NewSQL databases. But what's the answer?



TECHNICAL
STUFF

A nonrelational database doesn't use the table format, with defined rows and columns, used by relational database systems. A NoSQL database is literally a database that does not use Structured Query Language (SQL) for queries, but several formerly NoSQL databases have added SQL-like query languages. The term *NoSQL database* is often used to mean the same thing as *nonrelational database*.

As we've seen, no single solution meets every set of today's diverse requirements.

Business-critical applications need a flexible, scalable relational database that offers the best features of its predecessors but eliminates their shortcomings, a database that meets the requirements of cloud-native application code running at scale.



REMEMBER

A new category of databases has arrived, and it's called distributed SQL. A distributed SQL database matches the characteristics of a traditional SQL database but brings essential flexible capabilities previously found only in the cloud-native world.

Understanding the Appeal of Distributed SQL

Distributed SQL essentially combines the best of two worlds: the transactional capability and SQL support of relational databases with the scalability and resiliency offered by NoSQL and NewSQL.

Distributed SQL database management systems present a single logical database for both traditional and containerized applications. Behind the scenes, they're powered by a distributed, multi-cloud data storage layer that can span across public, private, and hybrid clouds. This storage layer automatically replicates the data for resilience. All query processing is distributed across multiple servers for scale.

This new architectural approach offers advantages that aren't available elsewhere:

- » **Best of multiple worlds:** Developers get attributes usually found with traditional SQL — low latency; relational data modeling and capabilities; atomic, consistent, isolated, and durable (ACID) transactions; high performance; familiar interfaces; support for existing infrastructure. They also get those associated with NoSQL — geographic distribution, ease of deployment in a cloud-native environment, high availability, scalability, and simplified management.
- » **Simplicity of operations:** Because the data layer offers built-in scalability and resilience, database operations teams do not need to engineer solutions with caches and third-party replication solutions. A multi-cloud data layer based on distributed SQL is easy to deploy and quick to adopt due to relative simplicity at the point of interaction and the familiarity of its interfaces and capabilities. By working with apps across all cloud-native environments, organizations can start anywhere and scale everywhere, while maintaining or even improving performance, uptime, and data integrity.

» **Architectural resiliency:** Resiliency is offered at the data layer level, independently of how it's supported within specific applications. Because resiliency is built into the database rather than just implemented in the application layer, it allows capabilities to be updated and improved over time without affecting how application developers interact with the data layer. This offers a distinct advantage compared to more traditional databases, which focus solely on the application layer.

Distributed SQL is useful for a wide range of workloads. Some specific characteristics to look for when creating a new application, moving an existing application to the cloud, or upgrading an application that is failing to meet requirements, include:

- » **Transactional consistency:** Data updates that must be reflected identically no matter how, where, or when the data service is accessed benefit from a distributed SQL approach. The data layer delivers ACID compliance against a set of database assets spanning many servers that are carefully managed behind the scenes.
- » **Low latency:** What are the service level agreement (SLA) parameters that an application operates under? A time-critical application programming interface (API) call may need to run as quickly as required by the most demanding application. Low-latency applications benefit from speed and support for distributed — that is, appropriately local — data found in distributed SQL, where data can be moved around in the cloud or even placed on the edge close to users as needed.
- » **Ingest and query speed:** These may seem to be two different things, with high write speeds seen as a core NoSQL benefit and fast and reliable queries belonging to the traditional SQL camp, but they often go together. Distributed SQL delivers rapid ingest by leveraging its distributed nature and fast and reliable inquiries via its support for the relational model and SQL.
- » **Data access volume:** When vast amounts of data need to be accessed to reply to a query, any database approach will be hard-pressed to meet requirements. For instance, credit card approvals will examine as much data as possible for as long as the application can afford to give them. Distributed

SQL is a reliable and flexible model to meet such challenging queries.

- » **Data complexity:** Distributed SQL allows data to be structured and accessed, ensuring that complex data can be processed at high speeds. Relational data modeling has stood the test of time, remaining the most popular approach to manage data for business applications.

Looking Under the Distributed Architecture Hood

An advantage of a distributed SQL data layer is that most users shouldn't need to concern themselves with the implementation details of the underlying database architecture. However, understanding the detailed system setup can be useful when needed to support critical applications. For example, it may be possible to optimize data placement to help meet critical performance metrics such as latency and throughput.

So, how does a distributed architecture actually work?

The database management system presents a single logical SQL database to the user. The data layer is deployed on multiple data nodes (or servers) that, in current implementations, store parts of a database, called *shards*, on automatically created keys. Each key may consist of one or more data column names in the database schema.



TECHNICAL
STUFF

This new data stack is distributed SQL. It has a two-layer architecture, as shown in Figure 3-1. Logically, it functions as a single relational database. But functionally, it's deployed on a cluster of servers, ensuring resilience, scalability, and more.

Each data node has two parts:

- » **API/query layer:** This is the coordinating layer of the data node. All distributed SQL databases support a SQL API for applications to access when modeling relational data and to perform queries across these relations. The overall data layer automatically distributes data updates, API calls, and queries to the appropriate nodes for processing. Some

distributed SQL databases offer additional APIs at the query layer, like a Cassandra-compatible API, to look like a NoSQL environment.

» **Distributed data storage layer:** Indexes and data are automatically sharded by the data layer to multiple nodes, with data copied in a way that provides resiliency against failure. By distributing data, no single node becomes a bottleneck to performance or a single point of failure. Writes are synchronously committed across multiple nodes to support resiliency during the failure of a single node, and ACID transactions are distributed for single or multiple-row updates. Many solutions have built-in intelligence to seamlessly rebalance data as nodes come online or go offline.

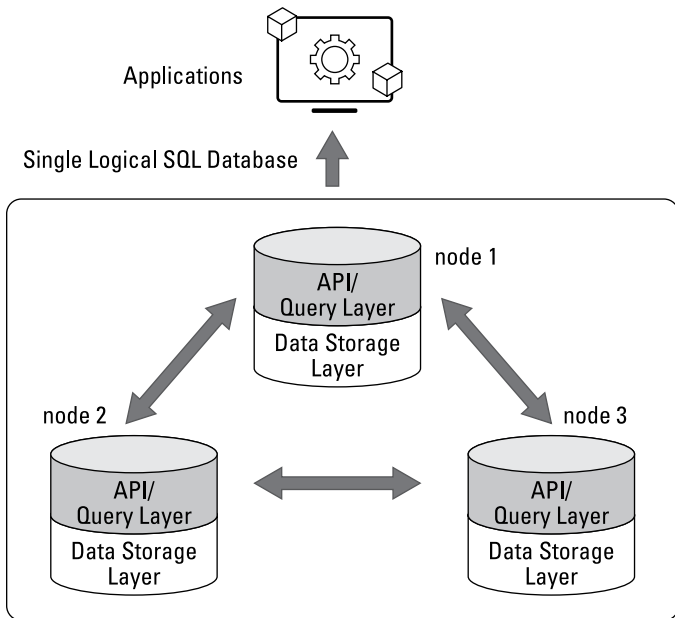


FIGURE 3-1: Distributed SQL has a two-layer architecture as part of a single logical SQL database.

Comparing Distributed SQL to Other Database Types

The capabilities and architecture of distributed SQL make it the ideal choice for cloud-native application development. If you're using microservices, containers, or Kubernetes to develop and deliver applications in a cloud-native environment, distributed SQL may be the best starting point for the data services required by your applications.

The advantages of standardization

There are often database requirements that need a specific database or legacy applications that don't need any additional features or capabilities. For example, you may decide not to migrate away from traditional SQL databases for legacy applications. Most companies will find themselves, often not because of a specific strategy, having to support many different databases.

However, even in a multi-database scenario, identifying a single database type as your organization's default first choice has significant advantages, which only increase as the company grows:

- » **Hiring and retention:** Developers and operations personnel will be able to use, learn, and stay up to date on fewer database types, with distributed SQL acting as the core.
- » **Faster application development:** Repeated use of a single database type will give developers "muscle memory" for that database.
- » **Reduced operational issues:** Distributed SQL is ideally suited for cloud-native environments, most application types, and use cases. So, you're less likely to incur operational issues.
- » **Faster resolution of operational issues:** As with application developers, operations people will develop pattern recognition for solving problems that arise.
- » **Higher application performance:** Distributed SQL is highly performant across various scenarios, so making distributed SQL the first choice is likely to improve application performance across the board.

- » **Increased agility:** Your organization's ability to respond to challenges and take advantage of opportunities will improve due to the benefits of standardization and the advantages of distributed SQL.



TIP

Selecting a widely used and familiar type of database can have significant benefits when hiring and retaining IT staff. Consider this alongside technical issues when determining your overall strategy.

Overcoming the challenges of traditional SQL

The advantages of using distributed SQL over traditional SQL in a cloud-native application development and delivery environment are clear. Distributed SQL has all the capabilities of traditional SQL, and more. And it's a far better fit for an agile, dynamic cloud environment.

Sticking with traditional SQL means continuing to suffer three significant disadvantages:

- » **Manual sharding:** When a traditional SQL database grows large, you're forced to shard it manually. This leads to difficult design choices, such as choosing or creating a shard key, and enduring operational difficulties as the sharded database sections grow and shrink at different rates.
- » **Bolt-on replication:** Growing databases are usually important enough to require high availability, but traditional SQL databases don't include this feature. You'll need to source and deploy a bolt-on replication solution, requiring design work and adding operational complexity.
- » **Lack of geo-distribution:** A traditional SQL database exists in one geographic location, which may be far from your customers. It may not meet performance needs or regulatory requirements.

Challenges with NoSQL

NoSQL presents three distinct challenges compared to distributed SQL:

- » **Operational complexity:** NoSQL databases represent the first generation of cloud-native databases. They emerged early in the cloud evolution and are not architected to optimally use the cloud as it is today. This makes development and operations harder. Performance is inconsistent and often only improved by committing extensive resources to data standardization or query optimization. Finally, the proliferation of NoSQL (and nonrelational) database types introduces complexities of its own.
- » **Frustrating application development:** Getting data into a NoSQL database is easy, but everything after that can be harder than it is with a SQL database. NoSQL requires more specific planning up front, which can make new queries a tangle of complexity, inconsistency, and difficulty. NoSQL can also bog down application development and operations.
- » **Inconsistent customer experiences:** A database used only for specific use cases makes it hard for your team to deliver its best work. This issue is exacerbated by the fact that NoSQL databases are eventually — not necessarily currently — consistent. Every level of your organization can expect some frustration when using a NoSQL database for all but the most straightforward use cases.

Data and database types are very sticky, and you're likely to have a complex data operating environment for a long time yet. Standardizing on distributed SQL, where possible, can reduce many of the disadvantages that arise when using NoSQL.

- » Realizing database modernization rewards
- » Unlocking cloud-native applications
- » Powering edge and streaming solutions
- » Solving problems for major industries

Chapter 4

Distributed SQL in Action: Exploring Top Use Cases

Every company has unique infrastructure and application needs, but many face similar challenges in their evolution to a data-centric company. These shared issues boil down to a few common themes.

The first area to cover is common problems faced by many, or even most, organizations today. Distributed SQL is particularly well suited for database modernization initiatives and for use with cloud-native “born in the cloud” applications and edge and streaming applications.

The second area to examine is industry-specific use cases. Many industries find that distributed SQL simplifies the data-centric issues they face. These industries include financial services, retail, telecommunications, manufacturing and automotive, software as a service (SaaS), and Internet applications.

This chapter outlines these potential solutions and industry use cases, so you can identify similar issues in your own organization that distributed SQL may help you fix.

Realizing the Rewards of Database Modernization

Increasingly, organizations are prioritizing database infrastructure modernization because their existing database solutions result in:

- » **Wasted time:** When an existing database is unsuited to application needs, or not powerful enough to meet them, developers spend extra time in the application trying to work around database gaps, instead of developing cool and differentiating features.
- » **Increased risk:** Organizations add capability in the form of bolt-on tools for data transformation, caching layers for improved performance, and so on. The result is a more complex architecture with additional items to maintain and added points of failure.
- » **Unexpected costs:** The high cost of most legacy solutions and the additional expense of bolt-on tools combine to make the data estate expensive and inadequate for the organization's changing needs. Figure 4-1 contrasts the rapidly rising costs of scaling up, as required by legacy solutions, with the steady and predictable cost increases of scaling out, as enabled by distributed SQL.

Moving to a modern data layer built on distributed SQL allows organizations to free their developers from developing complex applications due to limitations in the underlying database.

By automating key data operations, like automatic sharding, intelligent load balancing, and simplified multi-region deployments, developers can innovate faster and focus on delivering value-added services.

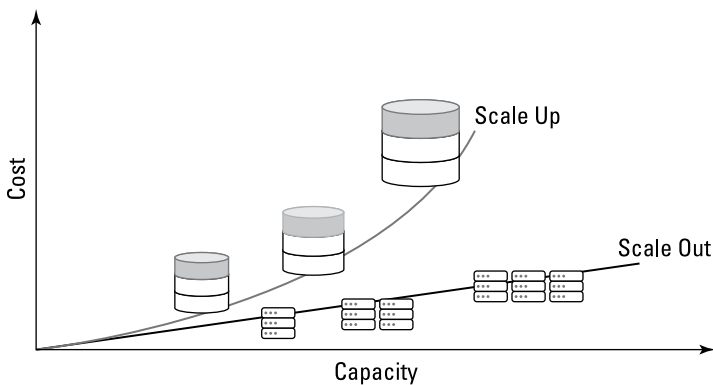


FIGURE 4-1: Scaling out avoids the rapid increases in cost associated with scaling up.

A huge benefit of embracing a modern database is around risk reduction. Relational database modeling and the full suite of relational database management system (RDBMS) capabilities allow organizations to capitalize on the resilience, scale, and geo-distribution required to achieve the results they need while maintaining a secure, always-on business.

A modern database will increase operational efficiency, reducing costs. The hassle factor of sticking with a traditional database — which requires you to manage sharded copies of your data, add a bolt-on solution for replication, and grapple with the lack of geo-distribution — keeps your best people busy with operational issues and increases your costs.

As you examine the needs of your organization and the current state of your database or databases (because you're likely to be juggling a number of different solutions), consider the impact that a modern database could have on your business, developers, and customers today and in the years to come.

By shifting to a distributed SQL solution, trade-offs are eliminated, so you can deliver critical business outcomes by accelerating productivity, reducing costs, and lowering risk.



REMEMBER

Distributed SQL provides a modern future-proof data layer that allows you to scale confidently and enhance developer productivity. This enables you to maximize your investment in people and processes by using familiar tools and reducing complexity. Distributed SQL databases address your organization's needs today while preparing you for what the future holds.

Unlocking Cloud-Native Application Capabilities

Cloud-native applications are built on modern software development and design approaches such as microservices, continuous integration/continuous delivery (CI/CD), containers, and Kubernetes (for container orchestration).

These modern “born in the cloud” applications demand a cloud-native, distributed database that complements the rest of the tech stack.

Using monolithic legacy database architectures with cloud-native applications causes problems — the very problems that cloud-native application development and delivery were created to avoid:

- » **Forced trade-offs:** Developers must choose between an RDBMS that has native support for schema and SQL but doesn't scale, and a nonrelational database that scales but doesn't fit organizational needs for structure and consistency.
- » **Complex code:** With legacy databases, developers must “add on” database scaling and resilience manually. This is expensive and results in fragile solutions, as well as the need for re-architecting each time an application needs to scale further.
- » **Operational headaches:** Complex, inconsistent solutions make automation harder. They present proprietary application programming interfaces (APIs) that have to be mastered and maintained, and cause fresh problems each time scaling thresholds are exceeded.

With a cloud-native, distributed SQL database, app development proceeds on a familiar, flexible, modern stack. The data layer matches the distributed, clustered architecture of Kubernetes along with the productivity, agility, performance, and simplicity of cloud-native application code.

Developers work faster, customers (internal and external) are happier, and operations are smoother. Your organization benefits from faster time to value, easier scaling, and access to a vibrant community that supports the open-source solutions powering distributed SQL.



Kubernetes is open-source orchestration software for containers — a key technology for cloud-native solutions. The term *cloud native* can also be applied to software running on-premises, as long as it's managed using cloud-friendly software like Kubernetes.

Powering Demanding Edge and Streaming Solutions

Streaming devices such as mobile phones are now spread all over the world, in great numbers. These devices generate and consume reams of data that greatly benefit from data that can be processed close to them. (Think of someone using a mapping or ride-sharing app — as they travel, they need up-to-date information at their current location.) It becomes ever more important to process data as close as possible to the data producers and/or consumers, and move data efficiently when needed.

Distributed SQL provides a consistent database architecture for streaming data and at the edge, where the people who need application access live and work. Figure 4-2 shows how the edge relates to the cloud.

Distributed SQL allows you to avoid a host of problems found with legacy solutions, including

- » **Slow response times:** You may re-architect your entire application to deliver responsive solutions at a global scale, but a legacy database solution will frustrate all your hard work. You'll have to move data to where the processing is, even if that's on the other side of the globe, and suffer slow ingest speeds, inconsistent processing capabilities, and many other disadvantages that a mixed architecture based on legacy technologies brings.
- » **Data silos:** Having some databases at the edge and others in the cloud makes it inevitable that you'll have data silos and high data costs. You'll also need to make databases do things they weren't designed for as you try to avoid large, expensive, slow data transfers between systems.
- » **Complex management:** Data replication, the need to scale, and outages plague complex systems that are inconsistent

at the edge and in the center. The same complexity can cause problems and make them tricky to solve.

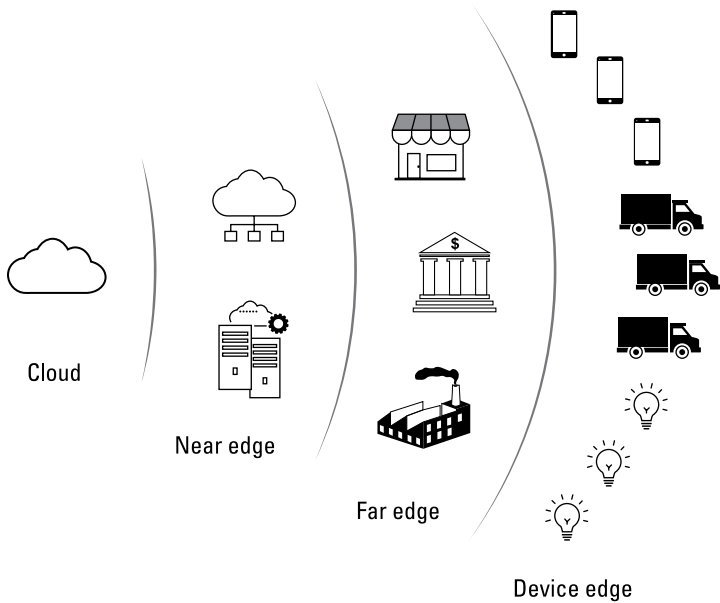


FIGURE 4-2: The edge in edge computing is divided into several tiers.

A cloud-native database powered by distributed SQL is purpose-built for edge and streaming applications. Unlike legacy solutions, the core distributed SQL architecture is built to automatically distribute data across nodes and regions to seamlessly address many common issues with streaming and edge applications. In addition, the cloud and platform-agnostic nature of these databases simplifies the deployment in different locations, allowing them to match whatever infrastructure or cloud is preferred in different regions.

You can create powerful and architecturally simple solutions to challenges that previously seemed insurmountable.



REMEMBER

Distributed SQL enables you to deploy anywhere on a consistent underlying infrastructure, steadily improve and future-proof that infrastructure over time, and deliver reliably high performance.

It provides low-latency data streaming and computing at the edge, support for large data sets with high throughput, support for multiple current connections, and horizontal scalability without added latency. Leading distributed SQL databases offer built-in replication between databases at the edge and in the data center or cloud.

Solving Problems across Industries

The solutions above are needed by nearly every organization of any size, but different industries have additional demands unique to them that require a mix of solutions. As I mention earlier, there are also some commonalities among the following organizations.

Financial services

Financial services organizations combine the need to support a wide range of legacy systems with the need to compete aggressively in a fast-changing world. Distributed SQL allows these organizations to be more productive and to create future-proof systems.

Regulatory requirements are especially stringent for financial services organizations. They only get tighter and more complex as regulators raise the stakes, seeking to avoid incidents like the financial crisis of 2008. Scalability requirements are uniquely pressing for financial services organizations that operate at a regional or global scale.

Financial services also face unique challenges:

- » Running financial ledgers on a distributed basis and at scale
- » Unifying customer data across services while meeting data protection regulations across markets
- » Accelerating the processing and data transfers that power financial services

Only distributed SQL allows financial service organizations to meet these obligations and take advantage of new opportunities, with the speed required by today's fast-paced world.

Retail and e-commerce

Retail and e-commerce organizations share many of the same requirements as financial services organizations, along with their own unique concerns. These include a strong need to differentiate from competitors and deliver modern, omnichannel services. Distributed SQL lends tremendous agility to these organizations, allowing them to innovate seamlessly at scale.

The need for differentiated experiences is particularly strong in retail and e-commerce. Entire retail locations, whether physical stores or online shopping portals, need to distinguish themselves from competitors. Experiences delivered online may need to be backed up by retail locations and vice versa, with a consistent customer experience throughout.

Imagine a brick-and-mortar location with a curated and locally aware selection of products, backed by a global catalog including hundreds or thousands of additional products, responsive to the customer's every need. Distributed SQL allows retail and e-commerce organizations to innovate nimbly across a range of touch-points and delivery platforms.

Telecommunications

Telecommunication requirements are extremely demanding, with interactive audio and video, and large data transfers competing for bandwidth. The move to 5G is exponentially increasing data volumes and the need for real-time data processing that delivers immediate value from the flood of data. Distributed SQL provides the fast, flexible, and scalable platform for delivering on these escalating requirements.

The need to continually modernize applications is especially strong in telecommunications, as rapid infrastructure improvements keep raising user expectations. Developers and architects need horizontal scalability to stay ahead of spikes in demand and maintain compatibility with existing infrastructure to make delivering solutions fast, easy, and reliable.

Like financial services, telecommunications providers need to meet stringent regulatory requirements. These often include local storage of data generated in a specific regulatory footprint,

even as users move rapidly across geographical and regulatory boundaries. Only a simplified data layer that scales across multiple clouds and hybrid cloud can support these complex requirements.

Manufacturing and automotive

The industrial requirements of the manufacturing, automotive, and energy sectors are demanding. They include specialized equipment that must run safely, meet detailed regulatory requirements, and provide a consistent stream of data to operators to ensure continuous, safe, and efficient operation.

Many industrial companies require global scalability, high availability, and a high degree of data integrity, similar to other sectors, but with even higher stakes, given the safety concerns involved. In addition, industrial systems must operate with a minimum of operational overhead and complexity, so operators can focus on the most important aspects of the systems they're running.

A distributed SQL database can scale as needed, operate across multiple cloud availability zones to meet geographic extensibility requirements, and run on any cloud or on-premises platform to provide flexibility and cost-effective operations. Industrial companies can use distributed SQL to innovate in a way that has never been available before.

SaaS and Internet

SaaS and Internet companies provide the digital infrastructure used by businesses and consumers to run their lives and organizations efficiently and cost-effectively. All the sectors I mention earlier depend on SaaS and Internet companies for new and improved solutions to meet customer needs, competitive pressures, and regulatory requirements.

A distributed SQL data layer provides these companies with the best of both worlds: the stability and reliability of SQL solutions that have been around for decades, and the flexibility and scalability they expect in the digital era. Distributed SQL provides these capabilities to SaaS and Internet companies, and allows them to pass them on to their own customers.

SaaS and Internet companies are especially aligned with the most flexible method for delivering a distributed SQL data layer: a fully managed solution. Often managed service providers themselves, these companies appreciate the time to market and flexibility benefits that a fully managed offering gives them.

IN THIS CHAPTER

- » Driving innovation more productively
- » Experiencing the resiliency of distributed SQL
- » Seeing the benefits of efficiency
- » Finding savings through scalability
- » Realizing benefits from enhanced security

Chapter 5

Measuring the Business Impact of Database Modernization

Being effective in business means making the right choices, and there are many opportunities to make a difference with IT initiatives. So, what makes using a distributed SQL database particularly valuable?

Part of the answer is technical simplicity, even a kind of beauty. Distributed SQL takes a familiar construct, the relational database, and supercharges it by bringing it into the cloud era.

The cloud is all about easy and predictable availability of resources. Distributed SQL uses that strength to rearchitect the relational database for the cloud-native era.

But technical simplicity and harnessing the power of the cloud are only part of the story. Distributed SQL also has direct and immediate business benefits. These include helping organizations increase productivity, resiliency, efficiency, security, and savings — PRESS for short.

In this chapter, I show you how distributed SQL, as delivered by Yugabyte with YugabyteDB, delivers these important benefits to your organization.

Accelerating Innovation with Greater Productivity

Improved automation and unlimited scaling are two key benefits of distributed SQL. They free up developers to focus on building value-add features.

The modern data layer automates data scaling and distribution. Distributed SQL can support new ideas in production in hours or days rather than weeks or months, increasing the rate of innovation, responsiveness, and competitiveness.

Key business metrics that you can realize with distributed SQL include

- » Time to first value (TTFV) is reduced by the availability of familiar interfaces and ways of working.
- » Application scaling time is reduced as database instances can be provisioned in hours (not days or weeks), speeding responsiveness.
- » Time taken to scale up or out the database is reduced, as clusters can easily be expanded with new or bigger nodes, without requiring downtime.
- » Performance improvements over existing database platforms can be realized, delivering tremendous value to the business — especially those with fast-growing apps and/or a widely distributed customer base.

YugabyteDB Managed, the fully managed database as a service (DBaaS) offering of YugabyteDB, eliminates the need for server purchasing, setup, provisioning, and so on. Developers become more agile and productive with instant access to the needed resources.



Support for industry-standard application programming interfaces (APIs) reduces TTFV. Yugabyte Structured Query Language (YSQL) is PostgreSQL-compatible, and Yugabyte Cloud Query Language (YCQL) is Cassandra-compatible. This makes YugabyteDB a drop-in replacement for less-flexible databases, while leveraging existing tooling and skill sets.

Because distributed SQL is a true relational database, it's easier to migrate existing apps to YugabyteDB and start realizing business value immediately. The availability of YSQL and YCQL further speeds migration.

Prioritizing Resiliency as a Key Database Requirement

Resiliency is one of the most important benefits of the cloud. It's important to maintain multiple copies of data, for instance, and to have the ability to seamlessly and non-disruptively support an application from a different, fully functioning server if one server goes down.

But traditional SQL fails to take full advantage of that efficiency. Because a single server controls each database, there's operational delay and higher operation costs when a server crashes.

Distributed SQL inherently avoids this problem. With distributed servers, true operational redundancy is automatically achieved for data storage just as it is for computation. The advanced features of a distributed SQL database ensure that distributed data across servers remains strongly consistent.



Yugabyte sets standards for data retention and performance when the cloud provider whose service YugabyteDB is running on suffers a zone outage. The recovery point objective (RPO), which measures any data loss, is set to 0, meaning no data loss will occur. The recovery time objective (RTO) to recover and resume operations from the new zone is 3 seconds.

Business metrics that are improved, often dramatically, by the resilient nature of distributed SQL include

- » Failure impact and mean time to recovery (MTTR) are measured in seconds or minutes, not hours.

- » Fewer database-related incidents that interfere with production.
- » Failover time can be instant, happening as soon as a load balancer is switched to point to a new database server — no data movement or data reconciliation is required.
- » Downtime for users when a cloud zone or data center goes down can be as little as zero because data can be synchronously replicated across zones and data centers in a true active/active configuration.
- » Data backups can be taken numerous times per day without impacting performance, allowing recovery from even the most serious crashes in minutes.



TIP

With distributed SQL, the database gains the same advantage that the cloud famously lends to compute: There is no single point of failure. The approach and learnings that operations use to make compute capabilities robust also now apply to data.

Uncovering the Hidden Effect of Efficiency

With distributed SQL, operations become much more efficient. DBAs, developers, and infrastructure teams need to do much less work to carry out routine tasks and respond to problems, even potentially serious ones, that would historically have led to days of data recovery activities.

This means important business metrics move in the right direction:

- » A database can be reconfigured or moved to a more current version, with zero downtime.
- » Database deployments can be scaled out in a few days after a business decision is made, and in minutes or a few hours after the infrastructure is ready.
- » Database performance bottlenecks, which may have previously taken hours or days to resolve (or not been solvable), can now be removed instantly.

- » Application and data migration time is often a limiting factor for many cloud initiatives and can be reduced drastically thanks to intelligent migration and analysis tools.

YugabyteDB is easily managed in any event, but when consumed as a service from the cloud, many management tasks are removed from operations' list of things to do. Even for on-premises environments, where closer data control is still required, solutions like YugabyteDB Anywhere greatly simplify the deployment and management of a self-managed DBaaS solution in your preferred data center or cloud.

And success is no longer a problem. With traditional SQL, increased traffic, usually a good thing for the business, can break infrastructure. With distributed SQL, which scales up and out, performance stays the same even when traffic increases.

Sensing the Safety in Security

IT security often takes the form of a race between hackers and operators. Upgrades and patches that take time to implement result in vulnerabilities that be exploited, often at significant cost.

The flexibility of distributed SQL means that upgrades and patches can be deployed very quickly and without downtime. This means that metrics around security are significantly improved:

- » Patches to fix common vulnerabilities and exposure (CVE) take no time to deploy.
- » Credentials are rotated every few months, not every year (or more).
- » Operating system software version upgrades can be carried out without downtime.
- » Database software upgrades are carried out monthly and without downtime.
- » YugabyteDB Anywhere allows operators to exercise modern policy controls and limit access privileges per user and group, making it far easier to comply with recent standards such as the General Data Protection Regulation (GDPR).

Seeing the Sense in Savings

One of the advantages of the cloud is that you can pay for and provision only the resources you need, when you need them. Starting new projects is fast and cheap. And when an application is up and running, the ability to scale up also means the ability to scale down, reducing waste.

A modern database architecture delivers a simplified architecture that delivers advanced features managed by simplified tools, built-in automation, and native intelligence. These features have a strong, and in some cases transformative, effect on key business metrics:

- »» The ratio of operations personnel to developers is very low, sometimes on the order of 1 operator to 100 developers.
- »» Infrastructure costs are far less due to the flexible use of commodity hardware versus the inflexible use of high-performance servers or specialized hardware usually required for databases like Oracle and IBM DB2. In addition, data density can greatly reduce hardware footprint, especially versus common NoSQL solutions.
- »» Even though distributed SQL is far more capable, software licensing costs for YugabyteDB are far less than for legacy databases.
- »» With distributed SQL, revenue loss due to downtime can be eliminated; the top line (revenue) is preserved, so the bottom line (profit) stays high.
- »» Consolidation of existing SQL and NoSQL workloads into a distributed SQL implementation saves money with reduced database sprawl and lower operational hassles.

Legacy databases often require extra fees for enterprise-grade features like monitoring and data replication. YugabyteDB is 100 percent open source, so these capabilities are included for free. Beyond the license costs, it's also important to analyze the overall value returned to the business from greater productivity and reduced risk.

In addition, concrete savings on hardware due to greater efficiencies and data densities, plus lower operational costs thanks to built-in automation and intuitive user interfaces can have a huge impact on reducing your total cost of ownership (TCO).



REMEMBER

Distributed SQL is run on a cluster of inexpensive commodity hardware, like compute. This takes full advantage of the promise of the cloud and eliminates the need to scale up into expensive custom servers or cloud instances.

Chapter 6

Ten Reasons to Pick Distributed SQL

In many ways, distributed SQL is a simple upgrade to traditional relational databases. Yet it profoundly affects your application development and finally aligns the data layer with the apps and infrastructure modernization you've invested in over the past decade.

In this chapter, I summarize the key advantages of distributed SQL, so you can quickly review them and see which features may be most important to your environment.

The first four advantages — strong consistency, continuous availability and resiliency, horizontal scalability, and geo-distribution — are inherent to well-engineered distributed SQL solutions, including YugabyteDB.

The remaining six advantages — dynamic workload optimization, hybrid cloud and multi-cloud deployment, the use of an open-source licensing approach, security and compliance, ease of migration, and ease of operations — are additional capabilities found in some distributed SQL offerings, including YugabyteDB, but not in all.

Strong Consistency

Cloud databases tend to force a trade-off between the scalability of cloud compute capabilities and the consistency of traditional SQL databases. Distributed SQL offers both scalability and fully ACID-compliant transactions.



REMEMBER

ACID stands for atomicity, consistency, isolation, and durability.

Continuous Availability and Resiliency

Traditional SQL databases are fragile because they depend on a single server or complex replication solutions. A distributed SQL solution is natively resilient and can easily meet requirements for continuous availability.

Horizontal Scalability

Cloud compute is horizontally scalable; you can quickly access the number of compute servers needed to get your work done — no more, no less. With distributed SQL, your database gains those same advantages, while maintaining the strong consistency previously only found with traditional SQL.

Geo-Distribution

Traditional SQL databases lack the scalability needed to take advantage of the geographic distribution that is usually an inherent feature of the cloud. Distributed SQL makes geo-distribution instantly available.

Dynamic Workload Optimization

A key feature of the cloud is the ability to put each workload on just the right resources to run efficiently, without waste. The flexibility of distributed SQL makes it an excellent fit for a wide and ever-changing set of applications and application needs.

Hybrid Cloud and Multi-Cloud Deployment

The gulf between the functionality of most clouds compared to on-premises databases forces awkward workarounds and compromises, creating operational hassles, security exposure, and added costs.

In addition, cloud lock-in can lead to unacceptable concentration risks. Distributed SQL behaves as a single logical database spanning a cluster of servers in your choice of public, private, and hybrid cloud environments.

Open Source/Open Standards

Open-source licensing and the use of open standards gives users control and flexibility. A distributed SQL database that uses open-source licensing and open standards give users the powerful features inherent to distributed SQL — strong consistency, continuous availability and resiliency, horizontal scalability, and geographic distribution — along with the advantage of openness.

Security and Compliance

Security is threatened by the fragility of single-server traditional SQL solutions and the lack of established compliance standards for weakly consistent NoSQL databases. Distributed SQL offers the best of both worlds, including advanced security features as a core part of the underlying design from Day 1.

Ease of Migration

Distributed SQL supports established approaches and specific standards used by traditional SQL databases, making it easy to migrate apps and data from traditional SQL to distributed SQL — a common reason why many cloud adoption initiatives fail or take longer than planned.

Ease of Operations

Look for a distributed SQL database built on a fully modern architecture that seamlessly delivers high levels of redundancy and resiliency, dynamic zero-downtime scaling, and flexible database-as-a-service consumption models along with advanced automation, powerful application programming interfaces (APIs), and familiar tools.

Such a database will make operations much easier, reducing toil for operators and wait times for everyone else.

Discover YugabyteDB Today!

It's like PostgreSQL for your business-critical cloud-native applications.

Starting with YugabyteDB is as easy as 1-2-3!

1. **Sign up** to YugabyteDB Managed for free
2. **Join** our active YugabyteDB Slack Community
3. **Follow us** on LinkedIn and tag @Yugabyte to share how you found us

Complete all three steps, and we'll send you an awesome **swag bag!** Plus, you'll finally be ready to bring your database into the cloud native world with consistency, scale, and high availability.

www.yugabyte.com

Get the most out of distributed SQL

Distributed SQL improves on a foundational technology: relational databases. Traditionally, such databases only run on a single computer or server. This makes it difficult to fully leverage the cloud: to handle truly large data sets, geographically distributed data, and even meet basic requirements such as resiliency, redundancy, and data backup. With distributed SQL, you get full scalability, along with the strong transactional capabilities usually only available with traditional SQL.

Inside...

- Ensure scalability
- Establish and improve redundancy
- Increase security
- Optimize processing for large data sets
- Make backups easier
- Handle transactions effectively
- Consolidate database sprawl



Floyd Smith is an experienced author and techie with experience in data-oriented startups such as Unravel Data and OtterTune. Books that he has written and coauthored include half a dozen *For Dummies* titles, including the bestselling *Creating Web Pages For Dummies*.

Go to **Dummies.com™**
for videos, step-by-step photos,
how-to articles, or to shop!

ISBN: 978-1-394-16889-7

Not For Resale



for
dummies
A Wiley Brand

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley's ebook EULA.