



EBOOK

# Modern Distributed Database Fundamentals

What Organizations Need to Know to Increase Scalability,  
Meet Ever-Increasing Data Requirements, and Streamline  
Tech Stacks



# Contents

- Introduction** ..... 3
  - The Evolution of Transactional Databases ..... 4
  - What is a Distributed SQL Database? ..... 5
  - What to Expect in this eBook ..... 7
- 1. Why Distributed SQL Databases Matter** ..... 8
  - Handling Ever-Increasing Data Requirements ..... 9
  - Increasing Modern Application Scalability and Availability ..... 11
  - Streamlining the Tech Stack Jungle ..... 14
- 2. Fundamentals of Distributed SQL Databases** ..... 17
  - Scalable by Design ..... 17
  - Versatile by Nature ..... 20
  - Reliable by Default ..... 22
- 3. Distributed SQL Use Cases** ..... 25
  - Database Modernization ..... 25
  - Tech Stack Unification ..... 27
  - Operational Data Management ..... 28
- 4. Choosing a Distributed SQL Database** ..... 30
  - Key Factors ..... 30
  - Evaluation Criteria ..... 31
  - Best Practices ..... 32
- 5. Evolving Transactional Data with TiDB** ..... 34
  - Introducing TiDB ..... 34
  - Origins of TiDB ..... 35
  - Inside TiDB's Distributed SQL Architecture ..... 35
  - The Advantages of TiDB ..... 37
- Conclusion** ..... 39



# Introduction

Transactional databases have been a fundamental component of modern computing for decades. They are designed to store and manage large volumes of data, while ensuring that the data remains accurate, consistent, and available. The evolution of transactional databases can be traced back to the early days of computing, with the emergence of the first computer systems.

# The Evolution of Transactional Databases

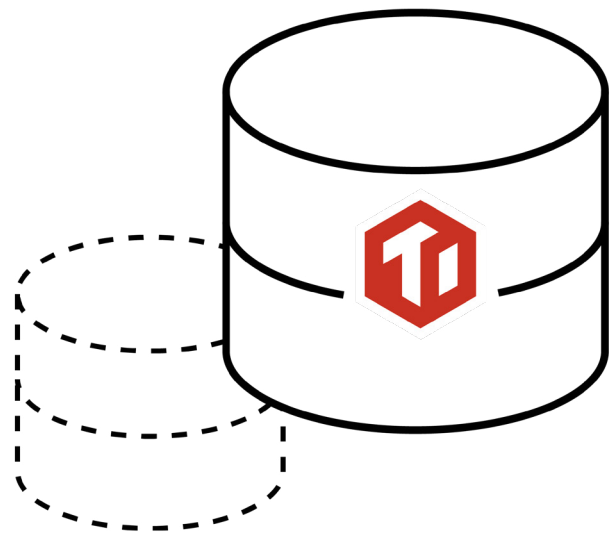
During the 1950s and 1960s, the first transactional databases were developed for mainframe computers. These early databases were typically built on top of hierarchical and network-based data models, which were common at the time. These databases were designed to handle simple transaction processing tasks, such as banking transactions.

In the 1970s, the first relational database management systems (RDBMS) were developed. These databases were built on the relational data model, which allowed data to be stored in tables with relationships between them. This made it easier to manage data, and allowed for more complex queries to be executed.

During the 1980s and 1990s, relational databases became increasingly popular, with the emergence of commercial database management systems, such as Oracle, IBM DB2, and Microsoft SQL Server. These databases were designed to handle large volumes of data and complex queries, and were used in a wide range of applications, including banking, retail, and healthcare.

By the early 2000s, the emergence of web-based applications and the increasing popularity of e-commerce led to the development of new types of transactional databases, such as NoSQL databases. These databases were designed to handle large volumes of unstructured data, and were used in applications such as social media and web analytics.

As organizations generate and process ever-increasing amounts of data, the need for scalable and efficient databases is becoming more important. Traditional SQL databases have long been used for managing and processing data with strong consistency, but they have scalability and performance limitations. NoSQL databases, on the other hand, are masters at data scalability and performance, but they tend to fall short when data consistency is a requirement.



What if there was a solution that combined the best of traditional SQL and NoSQL databases for modern applications?

In recent years, distributed SQL databases have emerged as a popular relational database

alternative. They offer the benefits of traditional SQL and NoSQL databases while also allowing for more efficient data processing of mixed workloads and storage across multiple nodes.

## What is a Distributed SQL Database?

Distributed SQL is a type of database architecture that distributes data across multiple nodes, allowing for elastic scalability, relentless reliability, and faster query processing of mixed workloads. Unlike traditional SQL databases that rely on a single-node server to store and process

data, distributed SQL databases distribute data across multiple servers, also known as nodes. Each node operates independently, but also communicates with other nodes to ensure that data is consistent and available for processing.

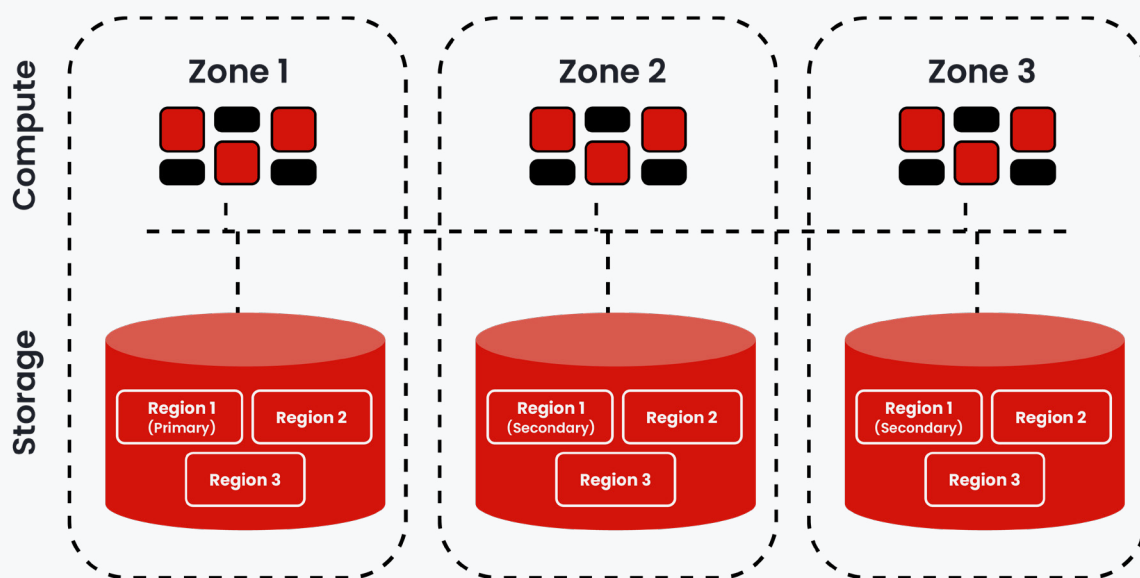



Figure 1. A typical distributed SQL data architecture



Distributed SQL databases work by partitioning data into smaller, more manageable subsets, known as shards. Each shard is stored on a separate node, and queries that involve data from multiple shards are executed across multiple nodes simultaneously. This allows for faster query processing and better performance, as each node can process queries in parallel.

Additionally, distributed SQL databases with mixed workload processing capabilities can combine row and column storage in a single database. This provides a single endpoint for mixed workloads while guaranteeing strong data consistency. Data can also be collected from multiple applications and aggregated instantly, allowing real-time queries to be performed on online operational data.

## Benefits of Distributed SQL

Distributed SQL databases offer several benefits over traditional SQL databases, including:

### 1. Scalability:

As data volumes grow, these databases can easily scale up or down to handle the load, by adding or removing nodes as needed. This makes it possible to handle large-scale data processing and storage without sacrificing performance.

### 2. Fault tolerance:

These databases are designed to be fault-tolerant, which means they can continue to operate even if one or more nodes fail. This is achieved by replicating data across multiple nodes, so that if one node fails, data can still be retrieved from other nodes.

### 3. High availability:

These databases can provide high availability, ensuring that data is always accessible, even in the event of a node failure. This is achieved through data replication, which ensures that data is always available on multiple nodes.

### 4. Mixed workload processing:

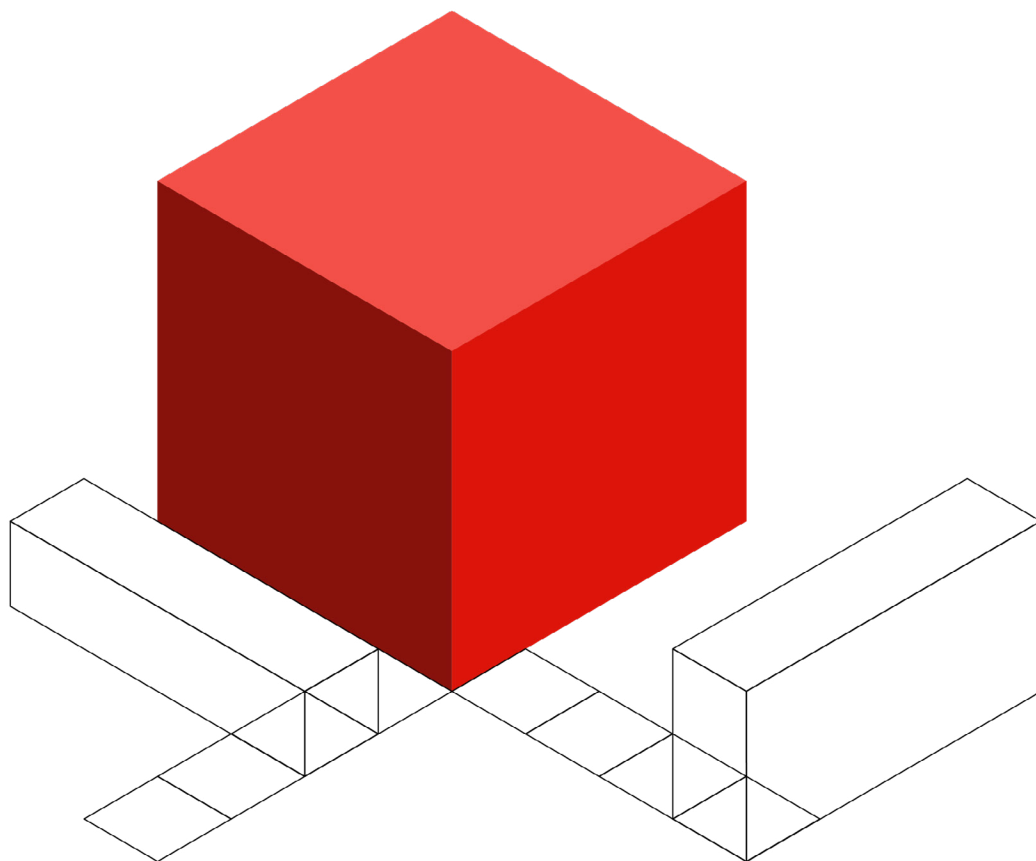
These databases can provide efficient complex query processing of mixed workloads, enabling greater developer productivity, a simplified architecture, and real-time data aggregations.

Distributed SQL databases are becoming increasingly popular as organizations look for ways to manage and process large volumes of data efficiently. As data continues to grow at an exponential rate, distributed SQL databases will become even more important for modern application development.

# What to Expect in this eBook

In this eBook, designed for application development, data architecture, and infrastructure leaders, we'll take a tour through the fundamentals of a modern distributed database. With distributed SQL databases as our lens, we'll uncover why these databases matter, how they're architected, as well as how they're used in real-world environments. We'll also explore what you need to know when choosing a distributed SQL database, as the proliferation of choices can be daunting to navigate.

By the end of this eBook, you'll have the knowledge and confidence to take the next step in your cloud-native journey. You'll also be able to pinpoint precisely what makes distributed SQL a unique modern distributed database solution for transactional data.





# 1

# Why Distributed SQL Databases Matter

Choosing the right database to power modern applications can be challenging. For starters, as data volumes grow when using a traditional relational database, performance and scalability radically degrade. These problems can only be remedied with additional data processing, aggregation, and integration tools. However, such solutions create greater technical complexity for developers, poor real-time performance, and higher data storage costs

Additionally, modern applications need to meet an increasingly mixed and complex set of requirements. When using a traditional relational database, teams end up adopting separate databases for transactional and

analytical workloads, adding even more technical complexity while opening up major challenges in data reliability and consistency.

With the explosive growth of data and the need for scalable and efficient systems, traditional relational and NoSQL databases have faced limitations. This has led to the emergence of distributed SQL databases, revolutionizing how organizations handle their data.



# Handling Ever-Increasing Database Requirements

In today's data-driven world, organizations are generating and collecting vast amounts of data at an unprecedented rate. From user interactions to IoT devices, the volume, velocity, and variety of data are continually expanding.

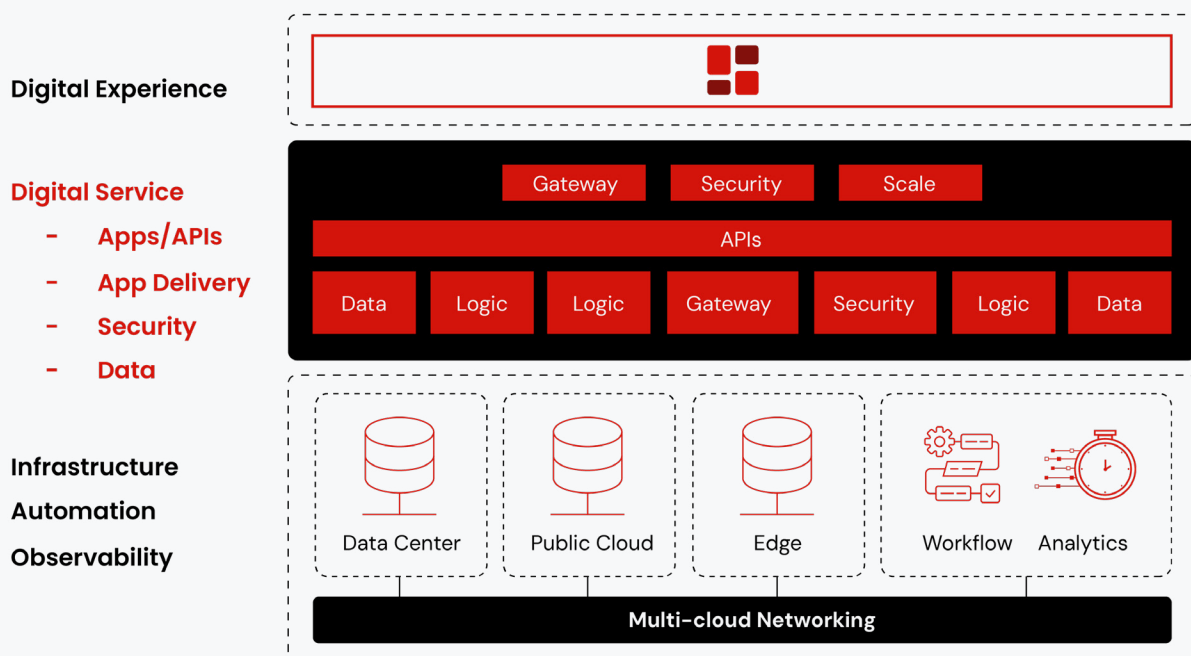


Figure 2. The acceleration of new customer experiences into digital channels is driving the creation of modern software applications as digital services

As a result, businesses face the significant challenge of effectively managing and processing this ever-increasing data. Distributed SQL databases have emerged as a robust solution to address these escalating data requirements.



## Scalable Data Storage

Traditional relational databases often struggle to accommodate the rapidly growing data volumes. Distributed SQL databases address this challenge by providing scalable data storage capabilities. Instead of relying on a single server, these databases distribute data across multiple nodes in a cluster. As data grows, organizations can seamlessly add new nodes to the cluster, allowing for horizontal scaling. This distributed nature of storage enables organizations to handle massive data growth while maintaining optimal performance and ensuring data availability.

## Elastic Computing Power

In addition to scaling storage, distributed SQL databases also offer elastic computing power. As data requirements increase, the demand for processing power grows. Traditional relational databases may experience performance bottlenecks when attempting to handle large volumes of concurrent queries and complex analytical workloads. Distributed SQL databases, on the other hand, leverage the distributed nature of their architecture to distribute query execution across multiple nodes. This parallel processing capability enables organizations to leverage the combined computing power of the cluster, resulting in faster query response times and improved overall system performance.

## Data Partitioning and Sharding


To efficiently handle ever-increasing data requirements, distributed SQL databases employ data partitioning and sharding techniques. Data partitioning involves dividing the data into smaller, more manageable subsets called partitions. Each partition is then distributed across different nodes in the cluster.

This approach allows organizations to distribute the data processing workload evenly and achieve better query performance. Furthermore, distributed SQL databases support sharding, which involves horizontally splitting the data across multiple nodes based on specific criteria such as a range of values or a hash function. Sharding ensures that data is evenly distributed across nodes, preventing hotspots and enabling efficient data retrieval.

## Data Compression and Optimization

With the exponential growth of data, storage costs and network bandwidth become critical considerations. Distributed SQL databases incorporate advanced data compression and optimization techniques to minimize storage requirements and improve data transfer efficiency.

By compressing data, these databases reduce the storage footprint, allowing organizations to store more data within the same infrastructure.



Additionally, optimized data transfer protocols and algorithms ensure efficient movement of data across the distributed cluster, reducing network latency and bandwidth consumption.

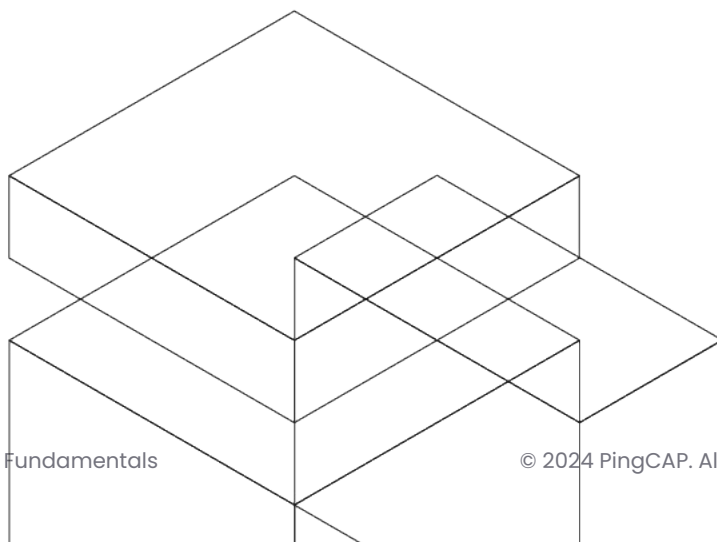
## Real-Time Data Processing

Ever-increasing data requirements often demand real-time or near-real-time data processing capabilities. Distributed SQL databases excel in this aspect by offering the ability to store data in real time when paired with distributed processing frameworks like Apache Kafka or Apache Flink. With this combined capability, organizations can handle continuous data streams, perform real-time analytics, and make data-driven decisions promptly.

As data requirements continue to grow, distributed SQL databases have proven their effectiveness in handling the challenges posed by this rapid data expansion. Through scalable data storage, elastic computing power, data partitioning and sharding, data compression and optimization, and real-time data processing capabilities, these databases empower organizations to efficiently scale and manage ever-increasing data volumes. By leveraging the distributed nature of their architecture, distributed SQL databases provide the scalability, availability, and flexibility required to meet the demands of modern data-driven applications.

## Increasing Modern Application Scalability and Availability

Organizations strive to deliver highly-scalable and always-on applications that provide an exceptional user experience. However, traditional database systems often struggle to keep up with the scalability and availability demands of modern applications, which require real-time responsiveness.



## Application Services

Complex Shard Access Logic

Complex Sharding Orchestration

Sharded for Scale

Source /  
Replica Model

Figure 3. An example of a traditional database system that implements sharding, adding technical complexity

Distributed SQL databases have emerged as a powerful solution to address these challenges and significantly improve application scalability and availability.

## Distributed Query Execution

Distributed SQL databases leverage their distributed architecture to execute queries in parallel across multiple nodes. This parallel processing capability allows for faster query execution times, resulting in improved application performance. By dividing the query workload across the cluster, distributed SQL databases can harness the collective computational power of the nodes, effectively reducing the response times for complex queries. This distributed query execution ensures that modern applications can deliver real-time results to users, enabling them to interact seamlessly with the application.



## Intelligent Data Placement

Efficient data placement is crucial for maximizing application availability. Distributed SQL databases can intelligently distribute and replicate data across data nodes in multiple availability zones (AZs), offering high availability and fault tolerance. This means if a single node or less than half of the nodes fail, the system can continue to function, a characteristic traditional monolithic databases can never achieve. This intelligent data placement ensures that data is located closer to the nodes that require it, optimizing application availability.

## Disaggregated Storage and Compute Architecture

To further enhance application scalability and availability, distributed SQL databases utilize a disaggregated storage and compute architecture. This architecture separates computing from storage, so each layer can be deployed separately and scaled independently.

In a disaggregated storage and compute architecture, different functionalities are divided and allocated to two types of nodes: the Write Node and the Compute Node. This means you can decide the number of Write Nodes and Compute Nodes to be deployed as needed. Additionally, you can scale out or scale in the computing or storage capacity online as needed. The scaling process is transparent to application operations and maintenance staff.

## Integration with Modern Application Frameworks

Distributed SQL databases seamlessly integrate with modern application frameworks, enabling developers to leverage their performance-enhancing features. These databases support popular frameworks and libraries for application development, such as Spring Boot, Django, or Ruby on Rails.

By integrating with these frameworks, distributed SQL databases provide a familiar development environment and enable developers to take advantage of performance optimizations specific to the database. This integration ensures that modern applications can harness the full potential of distributed SQL databases and deliver exceptional performance to end users.

# Streamlining the Tech Stack Jungle

In a rapidly-evolving technological landscape, companies often find themselves navigating through a complex jungle of technologies, frameworks, and tools. Managing multiple components and integrating them seamlessly can be a daunting task. Distributed SQL databases offer a valuable solution by streamlining the tech stack jungle, simplifying the architecture, and reducing the complexity associated with data management.

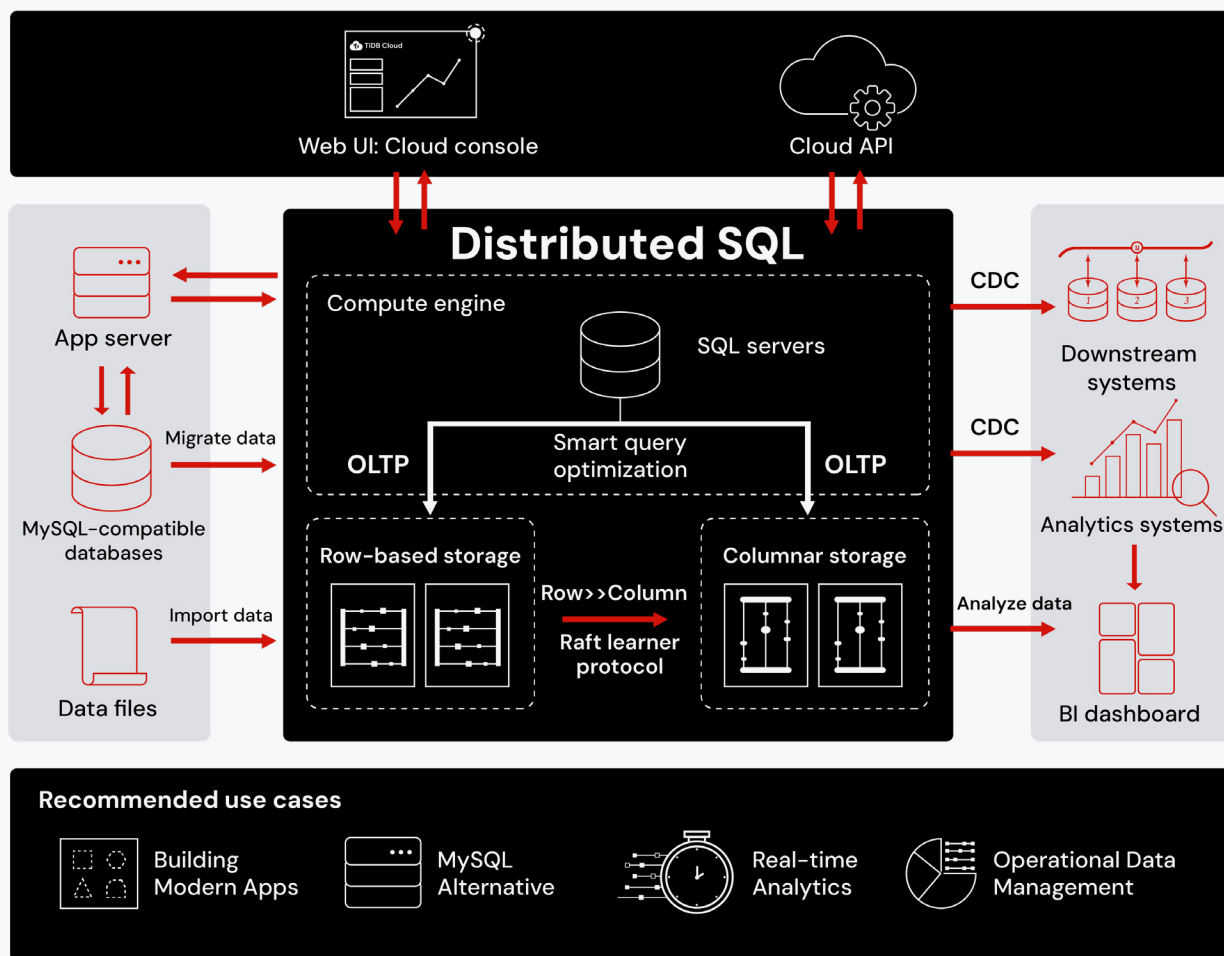


Figure 4. An example of a distributed SQL architecture with scalability and reliability for modern transactional apps coupled with real-time analytics on transactional data



## Consolidated Data Management

A significant challenge in the tech stack jungle is dealing with multiple data management systems. Traditional architectures often involve separate databases for different purposes, such as relational databases, NoSQL databases, caching systems, and message brokers. This fragmentation introduces complexities in data modeling, data synchronization, and maintaining consistency across systems.

Distributed SQL databases consolidate these different data management needs into a single, unified system. By consolidating data management, organizations can simplify their tech stack, reduce integration challenges, and streamline their operations.

### Integration with Ecosystem Tools and Frameworks

Navigating the tech stack jungle often involves integrating various tools and frameworks to build end-to-end solutions. Distributed SQL databases are designed to seamlessly integrate with popular ecosystem tools and frameworks. They provide connectors and APIs for integration with programming languages, frameworks, and data processing platforms.

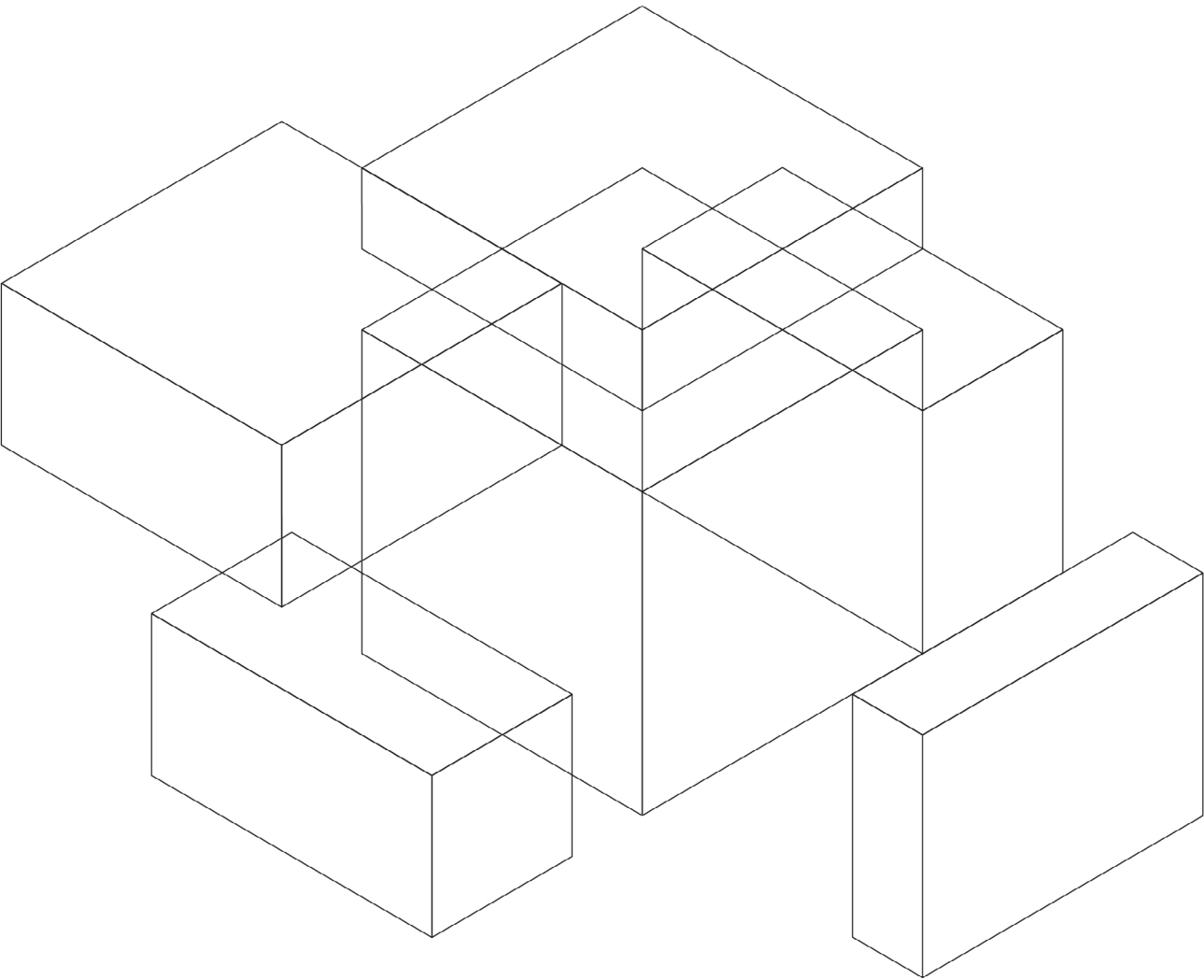
Whether it's integrating with data streaming frameworks like Apache Kafka or connecting with analytics tools like Apache Spark, distributed SQL databases—when designed as an open-source product—offer robust integrations that simplify the process of building comprehensive data-driven solutions, which require less effort for adoption. This integration capability reduces complexity in the tech stack and ensures a smoother development and deployment experience.

### Simplified Data Operations


Effective data operations are essential for managing the tech stack jungle efficiently. Traditional databases often require specialized knowledge and expertise for administration, monitoring, and scaling. These monolithic systems may also require system downtime, as you're upgrading a single node. Distributed SQL databases address these challenges by providing built-in automation and management tools. Even better, these databases utilize automatic rolling upgrades since they upgrade nodes one-by-one, minimizing impact to the running cluster. These tools and built-in processes simplify tasks such as database deployment, configuration, monitoring, and scaling, allowing organizations to streamline their data operations.



Additionally, distributed SQL databases often offer intuitive web-based interfaces or command-line tools that provide a unified view and control over the entire distributed database cluster. This simplification of data operations minimizes the complexity associated with managing the tech stack jungle.







# Fundamentals of Distributed SQL Databases

# 2

In today's data-driven world, organizations face the daunting challenge of managing ever-growing volumes of data, ensuring reliable access, and accommodating dynamic workloads. To overcome these challenges, distributed SQL databases have emerged as a powerful solution, offering scalability, reliability, and versatility. This chapter will explore the fundamental principles that underpin distributed SQL databases and their significance in modern data management.

## Scalable by Design

Scalability is a key advantage of distributed SQL databases, enabling organizations to efficiently handle growing data volumes, user demands, and transactional workloads. In this section, we'll explore how distributed SQL databases are designed to be inherently scalable. We will delve into their horizontal scalability, automatic sharding capabilities, distributed transactions, and concurrency control mechanisms.



## Horizontal Scalability

Distributed SQL databases excel at horizontal scalability, allowing organizations to seamlessly expand their data management capabilities as their needs evolve. Horizontal scalability refers to the ability to add more nodes to a distributed system, thereby increasing its capacity and performance. Key features contributing to horizontal scalability include:

### 1. Node Addition and Removal:

Distributed SQL databases allow for easy addition and removal of nodes in the cluster. When the workload increases, organizations can add more nodes to distribute the data and processing load, effectively scaling the system. Conversely, if the workload decreases, nodes can be removed to optimize resource utilization. This elasticity in node management ensures that distributed SQL databases can scale up or down based on demand.

### 2. Load Balancing:

Distributed SQL databases incorporate load balancing mechanisms that evenly distribute data and query processing across the available nodes. Load balancing ensures that the workload is efficiently distributed, preventing any single node from becoming a performance bottleneck. By evenly distributing the data and queries, distributed SQL databases optimize resource utilization and maximize system throughput.

## Automatic Sharding


Automatic sharding is a vital capability of distributed SQL databases that allows them to partition data across multiple nodes transparently. Sharding ensures that data is distributed evenly and managed efficiently in a distributed environment. Key features of automatic sharding include:

### 1. Data Partitioning:

Distributed SQL databases automatically partition data into smaller, manageable chunks known as shards. Each shard contains a subset of the data and resides on different nodes within the cluster. Data partitioning allows for parallel processing of queries across multiple shards, enabling high-performance query execution.

### 2. Transparent Data Distribution:

Automatic sharding ensures that data distribution is transparent to applications. Applications can interact with the distributed SQL database as if it were a single logical database, without needing to handle the complexity of data distribution.



This transparent data distribution simplifies application development and maintenance while allowing the database to scale seamlessly.

## Distributed Transactions

Distributed SQL databases provide support for distributed transactions, allowing organizations to maintain transactional integrity across multiple nodes. Distributed transactions ensure that a group of database operations is treated as a single unit, guaranteeing consistency and durability. Key features of distributed transactions include:

### 1. Atomicity and Consistency:

Distributed transactions in distributed SQL databases adhere to ACID (Atomicity, Consistency, Isolation, Durability) properties. ACID properties ensure that a distributed transaction either commits entirely or rolls back entirely, preserving data consistency across multiple nodes. Distributed SQL databases employ consensus protocols to coordinate the commit or rollback of distributed transactions, maintaining transactional integrity.

### 2. Two-Phase Commit Protocol:

Distributed SQL databases employ the two-phase commit protocol to ensure the atomicity of distributed transactions. The two-phase commit protocol coordinates the commit decision across all participating nodes, ensuring that all nodes either commit or rollback the transaction consistently. This protocol enables distributed SQL databases to maintain data consistency and transactional integrity in a distributed environment.

Distributed SQL databases are designed to be inherently scalable, allowing organizations to handle increasing data volumes, user demands, and transactional workloads effectively. Through horizontal scalability, automatic sharding, distributed transactions, and concurrency control mechanisms, distributed SQL databases empower organizations to scale their data management systems seamlessly while ensuring consistent performance, data integrity, and transactional reliability. By leveraging these scalability features, organizations can confidently meet the challenges of modern data-intensive applications and achieve optimal resource utilization in distributed environments.



# Versatile by Nature

Distributed SQL databases form the backbone of modern data management systems, offering versatility that enables organizations to handle a wide range of data-intensive tasks. In this section, we'll explore the inherent versatility of distributed SQL databases, focusing on their high-performance architecture and their ability to handle mixed workloads efficiently.

## High-Performance Architecture

The high-performance architecture of distributed SQL databases is one of their key strengths, enabling them to deliver exceptional performance for a variety of data processing tasks. The following aspects contribute to their high-performance capabilities:

### 1. Distributed Query Execution:

Distributed SQL databases leverage their distributed architecture to execute queries in parallel across multiple nodes. This parallel processing capability allows for faster query execution times, enabling organizations to obtain real-time insights from large datasets. By dividing the query workload across the cluster, distributed SQL databases harness the collective computational power of the nodes, resulting in improved application performance and responsiveness.

### 2. Data Partitioning and Replication:

Distributed SQL databases employ intelligent data partitioning and replication techniques to ensure efficient data distribution and availability. Data is divided into smaller, manageable partitions that are distributed across the nodes in the cluster. This distribution enables the database to process queries in parallel, as each node operates on a subset of the data. Additionally, data replication ensures fault tolerance and high availability by storing copies of data across multiple nodes. In case of node failures, the database can seamlessly retrieve data from other replicas, minimizing downtime and ensuring continuous operations.



## Mixed Workload Processing

Another aspect of the versatility of distributed SQL databases is their ability to handle mixed workloads efficiently. Whether it involves processing analytical queries, transactional operations, or a combination of both, distributed SQL databases excel in accommodating diverse workloads. Here's how they achieve this:

### 1. Hybrid Transactional/Analytical Processing (HTAP):

Distributed SQL databases support HTAP, allowing organizations to perform both transactional and analytical operations within a single database system. This eliminates the need for separate systems for different workloads, simplifying the architecture and reducing operational complexities. Organizations can run complex analytical queries while simultaneously serving transactional requests, enabling real-time decision-making and faster time-to-insights.

### 2. Automatic Query Optimization:


Distributed SQL databases employ advanced query optimization techniques to efficiently handle mixed workloads. These techniques analyze query patterns, data distribution, and available resources within the cluster to generate optimized execution plans. By automatically

optimizing queries, distributed SQL databases ensure that analytical and transactional workloads coexist harmoniously without impacting each other's performance.

### 3. Workload Isolation and Prioritization:

Distributed SQL databases allow for workload isolation and prioritization, ensuring that critical transactional operations are not affected by resource-intensive analytical queries. Workload management features enable organizations to allocate resources based on predefined priorities and dynamically adjust resource allocation as per changing workload demands. This ensures that mission-critical transactions receive the necessary resources and performance, even in the presence of heavy analytical workloads.

The versatility of distributed SQL databases makes them a fundamental component of modern data management systems. Their high-performance architecture, characterized by distributed query execution and intelligent data partitioning, enables organizations to achieve optimal performance for a



wide range of data processing tasks. Additionally, their ability to efficiently handle mixed workloads, including HTAP scenarios, ensures flexibility and agility in meeting diverse data processing requirements. By harnessing the power of distributed SQL databases, organizations can unlock the potential of their data and drive innovation in today's data-driven world.

## Reliable by Default

Reliability is a foundational characteristic of distributed SQL databases, ensuring consistent and dependable data management in distributed environments. In this section, we will explore how distributed SQL databases are designed to be reliable by default. We will delve into their strong consistency guarantees, high availability features, fault tolerance mechanisms, and disaster recovery capabilities.

### Strong Consistency

Distributed SQL databases provide strong consistency guarantees, ensuring that data remains consistent across all nodes in the distributed system, even in the presence of concurrent operations. Strong consistency is essential for applications requiring accurate and reliable data access. Key features contributing to strong consistency include:

#### 1. Distributed ACID Transactions:

Distributed SQL databases support ACID (Atomicity, Consistency, Isolation, Durability) properties across distributed transactions. ACID transactions ensure that multiple operations within a transaction either complete successfully or fail entirely, maintaining data consistency across distributed nodes. These transactions guarantee that the database is always in a consistent state, regardless of concurrent modifications.

#### 2. Consensus Protocols:

Distributed SQL databases employ consensus protocols, such as Raft, to achieve strong consistency. These protocols enable nodes to agree on the order of operations and reach consensus on the state of the distributed database. Through consensus, distributed SQL databases ensure that data modifications are applied uniformly across all nodes, guaranteeing strong consistency across the system.



## High Availability

High availability is a crucial aspect of distributed SQL databases, ensuring that applications remain accessible and responsive even in the face of node failures or network interruptions. Distributed SQL databases achieve high availability through various features, including:

### 1. Automatic Data Replication:

Distributed SQL databases automatically replicate data across multiple nodes in the cluster. This replication ensures that copies of data are readily available, even if a node fails. By replicating data, distributed SQL databases maintain data availability and enable seamless failover in case of node outages.

### 2. Failover Mechanisms:

In the event of a node failure, distributed SQL databases employ failover mechanisms to ensure uninterrupted service. Failover mechanisms detect failed nodes and automatically promote replica nodes to take over the responsibilities of the failed nodes. This automatic failover process ensures high availability by quickly restoring operations and minimizing downtime.

## Fault Tolerance

Fault tolerance is a critical capability of distributed SQL databases, enabling them to withstand hardware failures, network issues, or other system failures. Distributed SQL databases implement fault tolerance through the following mechanisms:

### 1. Data Replication:

Distributed SQL databases replicate data across multiple nodes, providing redundancy and safeguarding against data loss. If a node fails, data can be retrieved from replicas, ensuring that data remains accessible and preserving system functionality.

### 2. Self-Healing Architecture:

Distributed SQL databases employ self-healing architectures that automatically detect and recover from failures. When a node fails, the system identifies the failure and redistributes its responsibilities to other available nodes, ensuring fault tolerance and maintaining system integrity.



## Disaster Recovery

Disaster recovery (DR) is a critical aspect of distributed SQL databases, ensuring data integrity and business continuity in the face of catastrophic events. Distributed SQL databases provide robust DR capabilities through the following mechanisms:

### 1. Backup and Restore:

Distributed SQL databases can realize high availability of clusters based on the Raft consensus protocol and a reasonable deployment topology. That means when a few nodes in a cluster fail, the cluster can still be available. On this basis, to further ensure data safety, a distributed SQL database provides backup and restore as the last resort to recover data from natural disasters and operational failures.

Backup and restore in a distributed SQL database satisfies the following requirements:

- a. Backs up cluster data to a DR system with a Recovery Point Objective (RPO) as short as 5 minutes, reducing data loss in disaster scenarios.
- b. Handles operational failures from applications by rolling back data to a time before the error event.
- c. Performs history data auditing to meet the requirements of judicial supervision.
- d. Clones the production environment, which is convenient for troubleshooting, performance tuning, and simulation testing.

### 2. Replication Across Geographical Regions:

Distributed SQL databases offer the ability to replicate data across multiple geographical regions or data centers. By replicating data in different locations, organizations can achieve geographical redundancy, ensuring that data remains available even if an entire region or data center is affected by a disaster.

Distributed SQL databases prioritize reliability by default, enabling organizations to have confidence in their data management systems. Through strong consistency guarantees, high availability features, fault tolerance mechanisms, and disaster recovery capabilities, distributed SQL databases ensure consistent and dependable operations in distributed environments. By leveraging these inherent reliability features, organizations can build robust and resilient applications, meeting the demands of modern data-intensive scenarios while maintaining data integrity and availability.



# 3

## Distributed SQL Use Cases

As we've demonstrated so far throughout this guide, distributed SQL databases have emerged as a transformative technology. They're revolutionizing the way organizations manage their data infrastructure while unlocking new levels of scalability, reliability, and versatility. In this chapter, we'll explore several use cases that demonstrate the practical applications and benefits of distributed SQL databases across different domains and industries. We'll also provide real-world examples to support these use cases.

### Database Modernization

Organizations are constantly seeking ways to modernize their data infrastructure to meet the evolving needs of their applications and users. Database modernization has become a critical initiative, as legacy systems often struggle to cope with the demands of scalability, performance, and agility required by modern use cases. Distributed SQL databases have emerged as a powerful solution to drive this database modernization journey.

## Legacy Database Challenges

Legacy databases often present several challenges and limitations that hinder organizations from achieving optimal performance and scalability. Some common challenges include:

- **Scalability:**

Legacy databases may struggle to scale horizontally, limiting their ability to handle increasing data volumes and high concurrency demands.

- **Performance Bottlenecks:**

As data sizes and query complexity grow, legacy databases can experience performance bottlenecks, resulting in slow response times and degraded user experiences.

- **Limited Agility:**

Legacy systems often have rigid data models and schemas, making it challenging to adapt to changing business requirements and evolve alongside modern applications.

- **Data Silos:**

Legacy databases may suffer from data silos, where different applications maintain their own isolated data stores, leading to data duplication, inconsistencies, and difficulties in data sharing.

## Industry Use Case:

# Financial Services

Distributed SQL databases offer a compelling solution for financial services organizations seeking to modernize their data infrastructure.

A global financial services company needed a solution with the combination of scale, simplicity, and strong consistency. Its architecture was made up of sharded MySQL on Vitess. This solution required major application refactoring, complicated sharding orchestration, and lacked strong transactional consistency.

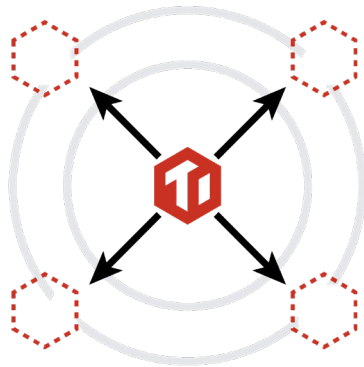
After extensive research and testing, the company chose a distributed SQL database with MySQL compatibility. Being MySQL compatible made the transition simple, while the distributed SQL architecture eliminated manual sharding, significantly simplifying its tech stack.

By eliminating sharding, the company enabled simple, no downtime scale in and scale out with no application refactoring. Its new distributed SQL architecture also provided strong consistency through full ACID compliance.

With a distributed SQL database, this company achieved better performance results—with more data—and no sharding, including:

- 4,000 queries per second (QPS)
- 5ms response times on 2TBs of data

# Tech Stack Unification



Successfully meeting project requirements on one data platform meant that the company could turn its attention to the innovation needs of the business at-large.

Companies often find themselves managing a complex and disparate tech stack comprising multiple databases, data processing frameworks, and data integration tools. This fragmented infrastructure can lead to inefficiencies, increased maintenance costs, and challenges in data management. Distributed SQL databases offer a compelling solution for tech stack unification, enabling organizations to streamline their data infrastructure, simplify operations, and achieve greater efficiency.

## Challenges of a Disparate Tech Stack

A disparate tech stack often arises as organizations adopt various technologies to meet different data processing requirements. However, this fragmentation can create several challenges:

- **Data Inconsistency:**  
Managing data across multiple systems can result in data inconsistencies, as each database may have its own data model and storage format. Synchronizing and reconciling data becomes complex and error-prone.
- **Increased Complexity:**  
Maintaining and operating a diverse tech stack requires specialized skills and resources, leading to increased complexity and overhead costs.
- **Integration Complexity:**  
Integrating data from different systems often requires custom solutions or ETL processes, making data integration complex and time-consuming.

- **Data Bottlenecks:**

Moving data between different systems can introduce latency and performance bottlenecks, negatively impacting the overall system performance.

## Operational Data Management

Efficient and effective management of operational data is a crucial aspect of any organization's success. It involves handling real-time data ingestion, processing, storage, and retrieval to support critical business operations and decision-making processes. Distributed SQL databases offer a powerful solution for operational data management, enabling organizations to handle large-scale data workloads, ensure data integrity, and achieve high performance.

### Industry Use Case:

## SaaS Providers

Distributed SQL databases offer a unified solution for SaaS providers to streamline their tech stack and address the challenges posed by a disparate infrastructure.

A popular Customer Success Platform (CSP) provider used PostgreSQL to handle all the data it collected externally. However, as its business grew and data sources rapidly expanded, PostgreSQL wasn't able to keep up with its needs. Additionally, due to the increased amount of data being stored, costs skyrocketed.

To handle these increasing demands, this company redesigned its entire data processing and storage system from the ground-up. A distributed SQL database was chosen to power this new architecture's data serving layer for pre-processing data for real-time customer queries.

By adopting a distributed SQL database, the company's CSP now provides:

- A better customer experience with 60x faster query responses
- A more resilient system
- Amplified data storage and processing
- Real-time analytical capabilities

The company also reduced its overall storage, operation, and maintenance costs.



## The Importance of Operational Data Management

Operational data serves as the lifeblood of organizations, fueling various business processes, applications, and analytics. Effective operational data management is essential for the following reasons:

- **Real-Time Decision Making:**  
Operational data provides valuable insights and supports real-time decision-making processes, allowing organizations to respond swiftly to changing market conditions and customer needs.
- **Business Process Optimization:**  
Proper management of operational data enables organizations to streamline their business processes, identify bottlenecks, and improve operational efficiency.
- **Customer Experience:**  
Access to accurate and up-to-date operational data enables organizations to provide personalized and exceptional customer experiences.

## Industry Use Case:

# Enterprise Software

Distributed SQL databases excel in managing operational data by providing robust features and capabilities. Here's how a distributed SQL database enhanced operational data management for a leading enterprise software company.

This company was using MySQL to manage its control plane containing multiple cloud services in AWS, GCP, and Azure. This control plane supports users, cluster management, and web applications. However, as the company's Azure cloud usage grew by 3x, Azure MySQL was unable to handle the increased load: queries became slow or even unresponsive for large customers.

The company evaluated a distributed SQL database against Azure MySQL and found the former outperformed the latter in latency and QPS with comparable hardware resources. Development teams were also impressed by distributed SQL's horizontal scalability, no manual sharding, and zero downtime for scale-in and scale-out.

With a distributed SQL database, this company has seen better performance with no MySQL scalability issues.

Additional benefits include:

- Lower average P99 and p999 latencies
- More than 10x QPS compared to Azure MySQL
- Reduced hardware costs and maintenance burden as the company can now host several control plane services in a single cluster



# Choosing a Distributed SQL Database

# 4

As organizations embrace the benefits of distributed SQL databases, selecting the right database becomes crucial for successful implementation. Choosing a distributed SQL database involves considering several key factors, evaluating various criteria, and following best practices to ensure the selected database aligns with your organization's requirements.

In this chapter, we will explore the process of choosing a distributed SQL database, covering key factors to consider, evaluation criteria to assess, and best practices to follow during the selection process.

## Key Factors

When choosing a distributed SQL database, it is essential to consider the following key factors:

### 1. Scalability:

Evaluate the database's scalability capabilities to ensure it can handle the anticipated growth in data volume and user traffic. Consider factors such as horizontal scaling, automatic data partitioning, and the ability to add or remove nodes from the cluster seamlessly.

## 2. Fault Tolerance and High Availability:

Consider the database's fault tolerance and high availability features to ensure data durability and minimize downtime. Look for features such as automatic data replication, failover mechanisms, and efficient disaster recovery options.

## 3. Data Consistency and Integrity:

Examine the database's data consistency and integrity mechanisms to ensure accurate and reliable data management. Features like distributed ACID transactions, strong consistency models, and conflict resolution strategies are crucial considerations.

## 4. Performance:

Assess the performance capabilities of the database, including query execution times, throughput, and latency. Look for features such as distributed query processing to optimize performance.

# Evaluation Criteria

To effectively evaluate distributed SQL databases, consider the following criteria:

## 1. Architecture and Design:

Assess the database's underlying architecture, distribution model, and data partitioning strategies. Evaluate how well the architecture aligns with your organization's requirements, such as data distribution, fault tolerance, and scalability.

## 2. Performance Benchmarking:

Conduct performance benchmarking tests to evaluate the database's performance under realistic workloads. Compare query execution times, throughput, and latency across different databases to identify the one that meets your performance expectations.

## 3. Data Management Features:

Evaluate the database's data management features, such as indexing capabilities, data compression, backup and restore mechanisms, and data security features. Consider how well these features align with your organization's data management needs.

## 4. Ecosystem Integration:

Assess the database's compatibility and integration with your existing tech stack and ecosystem tools. Consider its support for programming languages, frameworks, data processing platforms, and data streaming frameworks.

## 5. Open-Source Community and Support:

Evaluate the strength and responsiveness of the database's community and support channels.

Consider factors such as active user communities, available documentation, the type of licenses available, and the responsiveness of the vendor or open-source community in addressing issues or providing support. Databases with deep open-source roots have established trust with users, which is critical when considering a newer database paradigm such as distributed SQL.

# Best Practices

Follow these best practices when choosing a distributed SQL database:

## 1. Clearly-Defined Requirements:

Clearly define your organization's requirements, including scalability needs, performance expectations, data model support, and high availability requirements. This will help narrow down the list of suitable distributed SQL databases.

## 2. Pilot Testing in Dev/Test Environment:

Conduct a pilot test phase where you can evaluate the shortlisted databases in a controlled development or test environment. Use representative workloads and scenarios to assess the performance, scalability, and ease of use of each database.

## 3. Consider Total Cost of Ownership (TCO):

Evaluate the TCO of each database, including licensing costs, hardware requirements, operational overhead, and ongoing maintenance costs. Consider both the initial investment and the long-term costs associated with the selected distributed SQL database.

## 4. Vendor or Community Support:

Consider the availability and quality of support options provided by the vendor or the open-source community. Ensure that you have access to reliable support channels and resources to address any issues or challenges that may arise during implementation and maintenance.

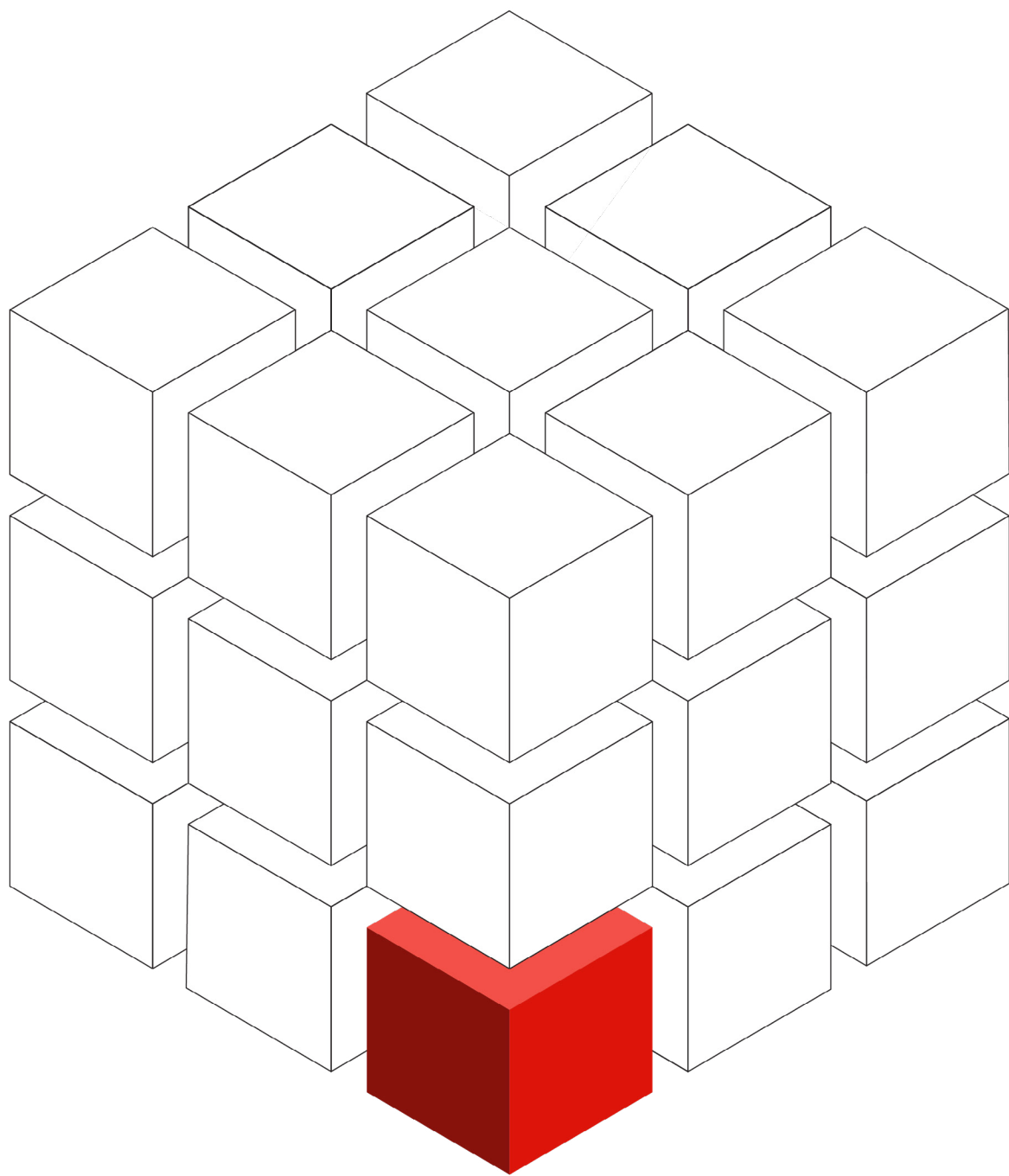
## 5. Scalability and Flexibility:

Choose a distributed SQL database that can scale with your organization's growth and adapt to evolving needs. Consider the flexibility to accommodate changes in data volumes, user traffic, and future data model requirements.





Choosing the right distributed SQL database is a critical decision that significantly impacts the success of your data management initiatives. By considering key factors, evaluating various criteria, and following best practices, you can navigate the selection process effectively. Remember to align your requirements, conduct thorough evaluations, and consider long-term implications to select a distributed SQL database that meets your organization’s needs and supports its future growth and success.



# 5

# Evolving Transactional Data with TiDB

As we've shown throughout this eBook, selecting the right database to power modern applications can be challenging. However, there's a better option that can evolve alongside your organization.

## Introducing TiDB

[TiDB](#), powered by PingCAP, unlocks limitless scale for data-intensive businesses. Our advanced distributed SQL database enables leading enterprises, SaaS, and digital native companies to build petabyte-grade clusters while managing millions of tables, concurrent connections, frequent schema changes, and zero-downtime scaling. With these advanced capabilities, growing companies like yours can focus on the future without worrying about complex data infrastructure management or tedious application development cycles.



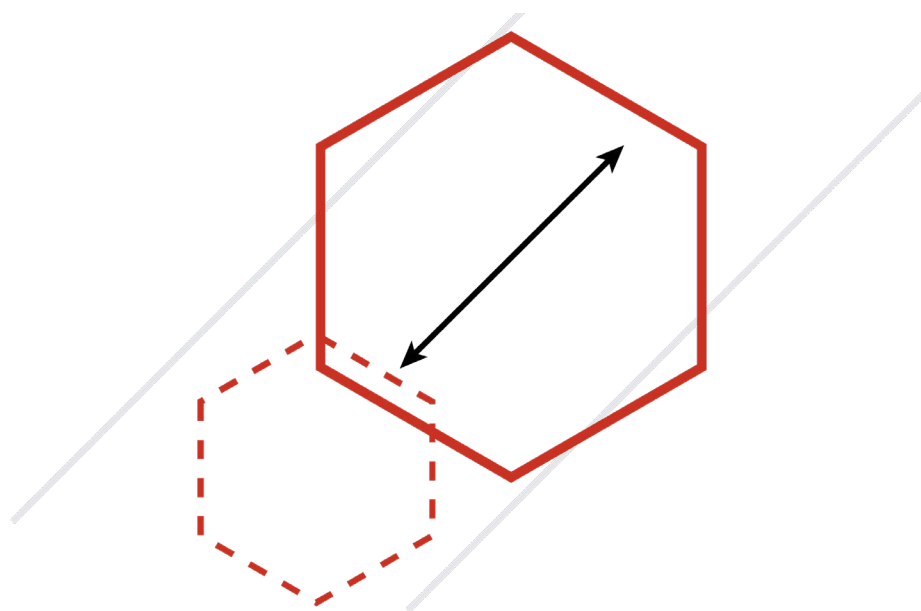
# Origins of TiDB

Created in 2015, TiDB was born when three talented software engineers decided to build a new database system that was more scalable and reliable than MySQL. At the time, they had grown frustrated with how MySQL was managed and maintained. They also wanted a true open-source solution that would maintain MySQL compatibility, as they enjoyed the database's rich ecosystem of tools and frameworks. What they didn't know then is just how popular their creation would become.

With [over 37,000 GitHub stars and a growing community of contributors](#), TiDB is also one of the world's most adopted open-source distributed SQL databases. The "Ti" in TiDB represents the symbol for Titanium from the Periodic Table of Elements. Found in nature as an oxide, Titanium is a powerful chemical element that can produce highly elastic, versatile, and reliable Titanium metal. With TiDB, some of the world's largest companies across technology, financial services, travel, Web3, and gaming are building modern applications as relentlessly powerful as Titanium.

## Inside TiDB's Distributed SQL Architecture

What makes TiDB so advanced? At its core, its distributed SQL architecture provides horizontal scalability, high availability, ACID transactions, and MySQL compatibility, while its unique mixed workload processing layer enables real-time analytics.



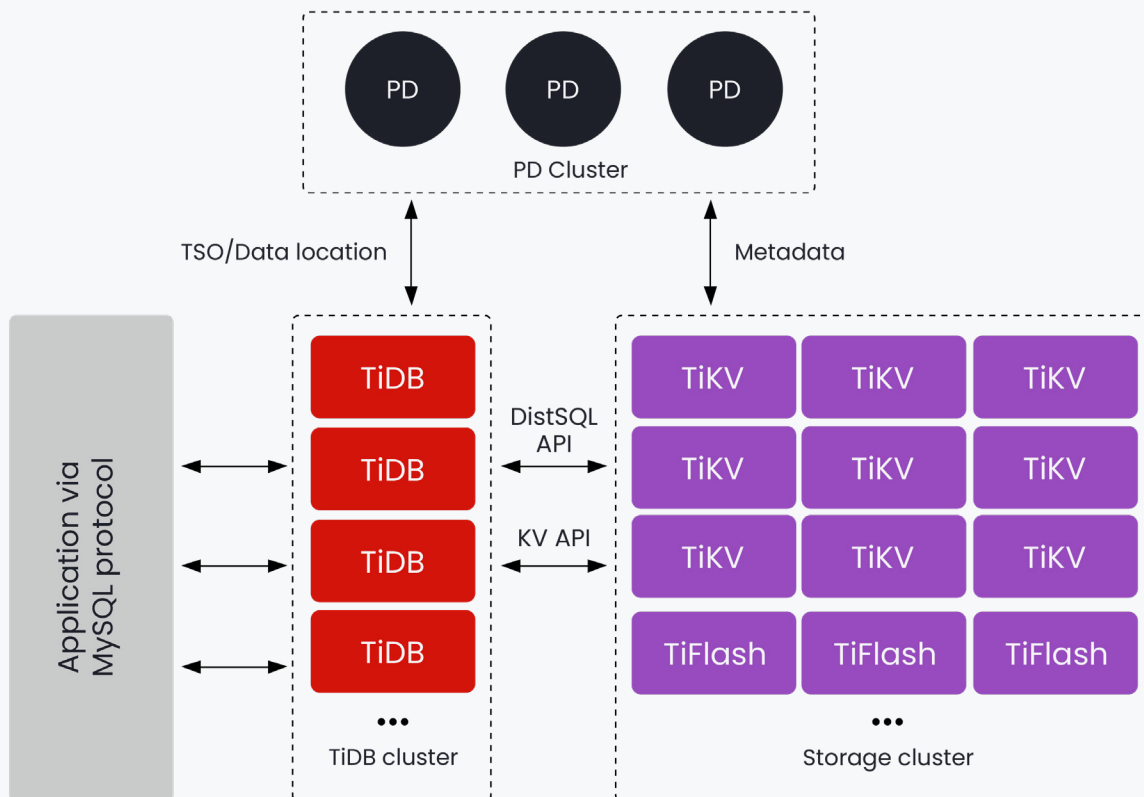


Figure 5. A visual representation of TiDB's architecture.

## TiDB SQL Layer

The TiDB SQL Layer separates compute from storage to make scaling simpler, delivering a true cloud-native architecture.

This stateless MySQL-compatible layer provides uniformity into data access without regard for sharding, or any other underlying technical implementation, making applications easier to develop and easier to use.

Additionally, the TiDB SQL Layer functions as a smart optimizer and parallel planning engine.

## Placement Driver Layer

The Placement Driver (PD) Layer functions just like a full-time DBA, monitoring millions of shards and performing hundreds of operations per minute. It also handles the scale-in and scale-out of clusters to meet demand. Additionally, this layer dynamically balances the data load in real time, mitigates hotspots, and provides for the implementation of customized scheduling policies.

## TiDB Storage Layer

Consisting of row and column-based storage engines, the TiDB Storage Layer offers built-in high availability and strong consistency that can auto-scale to hundreds of nodes and petabytes of data.

This layer also offers a modern replication mechanism based on the Raft consensus protocol.

# The Advantages of TiDB

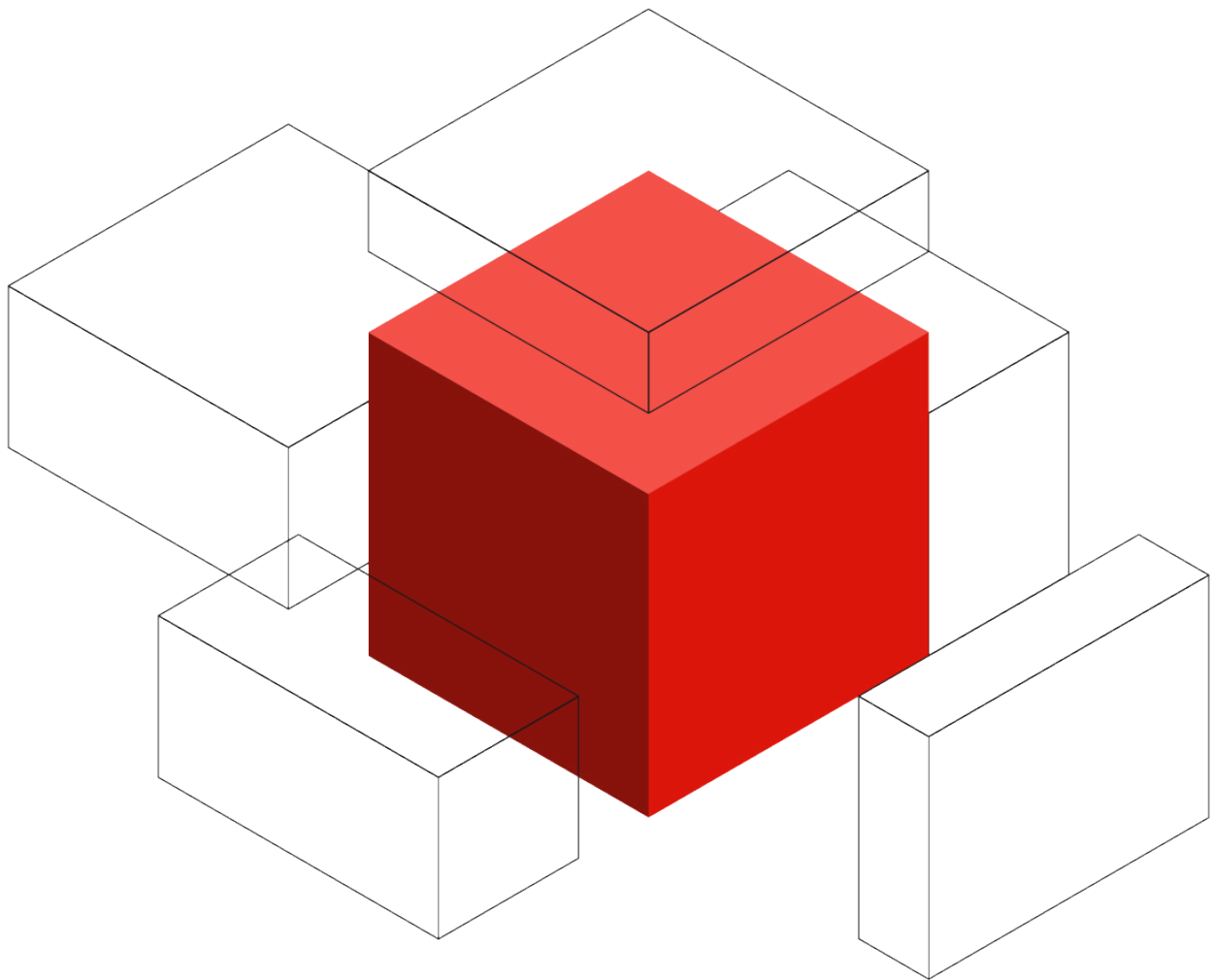
TiDB can power all of your modern applications with elastic scaling, real-time analytics, and continuous access to data. Companies using TiDB for their scale-out RDBMS and internet-scale OLTP workloads benefit from a distributed SQL database that is:

- **MySQL compatible:**  
Enjoy the most MySQL compatible distributed SQL database on the planet. TiDB is wire compatible with MySQL 5.7, so developers can continue to enjoy the database's rich ecosystem of tools and frameworks.
- **Horizontally scalable:**  
TiDB grants total transparency into data workloads without manual sharding. The database's architecture separates compute from storage to instantly scale data workloads out or in as needed.
- **Highly available:**  
TiDB guarantees auto-failover and self-healing for continuous access to data during system outages or network failures.
- **Strongly consistent:**  
TiDB maintains ACID transactions when distributing data globally.
- **Mixed workload capable:**  
A streamlined tech stack makes it easier to produce real-time analytics. TiDB's Smart Query optimizer chooses the most efficient query execution plan, which consists of a series of operators.
- **Hybrid and multi-cloud enabled:**  
With TiDB, IT teams can deploy database clusters anywhere in the world, in public, private, and hybrid cloud environments on VMs, containers, or bare metal.
- **Open source:**  
Unlock business innovation with a distributed database that's 100% open source under an Apache 2.0 license.

- **Secure:**


TiDB protects data with enterprise-grade encryption both in-flight and at-rest.

Global organizations, [such as those referenced in Chapter 3](#), are using TiDB for diverse database solutions from large-scale transactional workloads to recommendation engines, data-intensive applications, and more.



# Conclusion

We began by understanding the fundamental challenges faced by organizations dealing with transactional data in today's rapidly changing landscape. The exponential growth in data volume, the need for high-performance applications, and the complexity of modern tech stacks necessitate a scalable, reliable, and versatile solution. Distributed SQL databases provide precisely that.

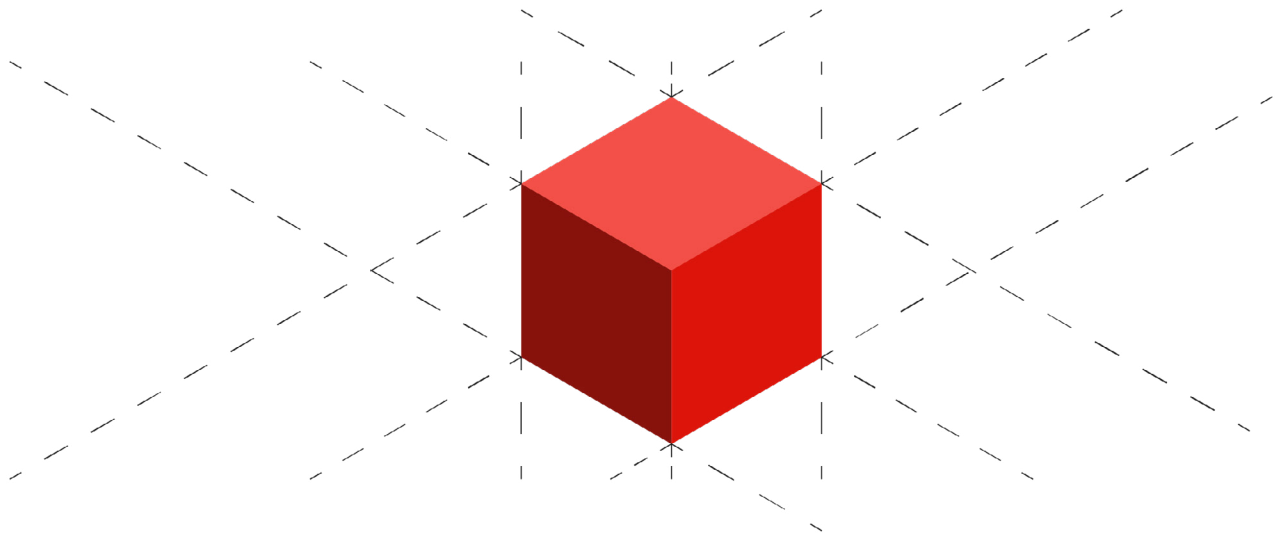


Next, we explored the key factors that make distributed SQL databases essential for managing transactional data in a modern context. From handling ever-increasing data requirements to improving application performance and streamlining the tech stack, distributed SQL databases address critical pain points faced by organizations across industries.

We then uncovered real-world use cases that demonstrate the practical applications of distributed SQL databases across various industries. These examples showcased how organizations have successfully leveraged distributed SQL to modernize their databases, unify their tech stacks, and efficiently manage operational data management.

Distributed SQL databases such as TiDB provide a solid foundation for organizations who want to evolve their transactional data management strategies. By embracing the modern distributed database principles mentioned in this guide, organizations can unlock the full potential of their data, leverage the scalability and reliability of distributed SQL architecture, and propel their businesses forward in the digital economy.

If you want to learn more about TiDB, find out how it's a more scalable alternative to MySQL in our [latest comparison guide](#).





## About TiDB

TiDB, powered by PingCAP, unlocks limitless scale for data-intensive businesses. Our advanced distributed SQL database enables leading enterprises, SaaS, and digital native companies to build petabyte-grade clusters while managing millions of tables, concurrent connections, frequent schema changes, and zero-downtime scaling. Large organizations who have adopted TiDB, such as Databricks, Pinterest, and Plaid, are finally free to focus on their future growth instead of complex data infrastructure management. With AI-driven innovations and multi-cloud flexibility, TiDB offers unmatched agility, resilience, and security. For more information, please visit [TiDB.io](https://tidb.io).

