DZone

TREND REPORT

NOV 2025

DATABASE SYSTEMS

Fusing Transactional Speed and Analytical Insight in Modern Data Ecosystems

BROUGHT TO YOU IN PARTNERSHIP WITH



Table of Contents

HIGHLIGHTS AND INTRODUCTION

03 Welcome Letter

Rob Bellenfant | *Founder and CEO at TechnologyAdvice/DZone*

DZONE RESEARCH

04 Key Research Findings

AN ANALYSIS OF RESULTS FROM DZONE'S 2025 DATABASE SYSTEMS SURVEY

G. Ryan Spain | *Freelance Software Engineer, Former Engineer & Editor at DZone*

FROM THE COMMUNITY

32 An Analysis of Modern Distributed SQL

Abhishek Gupta | *Product Manager at Microsoft*

39 Building Trusted, Performant, and Scalable Databases: A Practitioner's Checklist

Saurabh Dashora | *Software Architect at System Design Codex*

ADDITIONAL RESOURCES

44 Solutions Directory

Welcome Letter

By Rob Bellenfant, Founder and CEO at TechnologyAdvice/DZone

More than two decades ago, I began my journey in IT consulting and technology research with the goal of helping organizations solve critical business challenges through the strategic use of data, automation, and enterprise applications. That journey ultimately led to the creation of TechnologyAdvice. This online publication offering software reviews, advice, and guidance from a diverse community of technology experts has since grown into a global media company with more than 30 unique brands, including DZone, TechRepublic, eWeek, Channel Insider, The Neuron, eSecurity Planet, RT Insights, and more. Through these platforms, we've had the privilege of serving millions of IT professionals, software engineers, and business leaders around the world.

Throughout that time, I've had a front-row seat to the incredible evolution of enterprise IT and software development. From monolithic architectures to microservices, and from batch processing to real-time intelligence, one thing has remained constant: Progress in technology has always been driven by access to accurate, reliable, and actionable data.

Today, that's more true than ever.

Data is no longer just a byproduct of business systems. It's the fuel that powers innovation. Whether you are building enterprise applications, AI models, workflow automations, or autonomous agents, your ability to drive outcomes depends on the strength of your data foundation. And at the heart of that foundation? Modern database systems.

DZone's 2025 *Database Systems* Trend Report explores the next phase of database evolution, diving into distributed architectures, multi-model capabilities, and the critical importance of performance, scalability, and data governance in today's business environment. We've worked alongside some of the most experienced voices in the space to unpack how developers and data teams are solving real-world challenges and architecting for the future.

It's a topic that I'm truly passionate about, not just because of the technology itself but because of what it enables. We're seeing companies across every industry, from finance and healthcare to logistics and SaaS, transform the way they operate, serve customers, and grow, all by rethinking their approach to data systems.

Our goal at TechnologyAdvice has always been to help our communities stay ahead of the curve, and I'm proud of the role our brands continue to play in that mission. I hope you find the insights in this report valuable, thought-provoking, and energizing, and we look forward to hearing how your team is embracing the database technologies that will shape our digital future.

Thank you for reading,

Rob Bellenfant



Rob Bellenfant is the founder and CEO of TechnologyAdvice, a full-service B2B media company and leading provider of media, brand, and demand services. He started TechnologyAdvice in his freshman dorm room in 2006, and TechnologyAdvice now owns over 30 media brands, including DZone, TechRepublic, eWeek, Datamation, Channel Insider, Raconteur, and The Neuron. Rob is also involved in various philanthropic endeavors, including the support of organizations like Youth Villages and Best Buddies International. He lives in Nashville, Tennessee, with his wife and four children.

ORIGINAL RESEARCH

Key Research Findings

An Analysis of Results From DZone's 2025 Database Systems Survey

G. Ryan Spain, Freelance Software Engineer, former Engineer & Editor at DZone

Every organization is now in the business of data, but they must keep up as database capabilities and the purposes they serve continue to evolve. Systems once defined by rows and tables now span regions and clouds, requiring a balance between transactional speed and analytical depth, as well as integration of relational, document, and vector models into a single, multi-model design. At the same time, AI has become both a consumer and a partner that embeds meaning into queries while optimizing the very systems that execute them. These transformations blur the lines between transactional and analytical, centralized and distributed, human-driven and machine-assisted.

Amid all this change, databases must still meet what are now considered baseline expectations: scalability, flexibility, security and compliance, observability, and automation. With the stakes higher than ever, it is clear that for organizations to adapt and grow successfully, databases must be hardened for resilience, performance, and intelligence.

In September and October, DZone surveyed software developers, architects, and other IT professionals in order to gain insight into the current state of database systems in the software development space.

Methods

We created a survey and distributed it to a global audience of software professionals. Question formats included mainly single and multiple choice, with options for write-in responses in some instances. The survey was disseminated via email to DZone and TechnologyAdvice opt-in subscriber lists as well as promoted on DZone.com, in the DZone Core Slack workspace, and across various DZone social media channels.

The data for this report were gathered from responses submitted between September 24, 2025, and October 25, 2025; we collected 100 complete and partial responses. Our margin of error for the results of this survey is 10%, at a confidence level of 95%, and this report treats comparative results of 10% or less as insignificant. Segmented responses may have lower confidence levels due to smaller sample sizes.

Demographics

Before diving in, we noted certain key audience details in order to establish a more solid impression of the sample from which results are derived:

- 26% of respondents described their primary role in their organization as "Developer/engineer," and 23% described "Technical architect." No other role that we provided was selected by more than 10% of respondents.*
- 73% of respondents said they are currently developing "Web applications/Services (SaaS)," 56% said "Enterprise business applications," and 25% said "Native mobile apps."
- "Java" (67%) was the most popular language ecosystem used at respondents' companies, followed by "Python" (62%), "JavaScript (client-side)" (55%), "Node.js (server-side JavaScript)" (35%), and "TypeScript" (33%).
- Regarding responses on the primary language respondents use at work, the most popular was "Java" (39%), followed by "Python" (22%). No other language was selected by more than 10% of respondents.
- On average, respondents said they have 23.21 years of experience as an IT professional, with a median of 25 years.
- 26% of respondents work at organizations with < 100 employees or reported being self-employed, 27% of respondents work at organizations with 100-999 employees, and 43% of respondents work at organizations with 1,000+ employees.*

Note: For brevity, throughout the rest of these findings we will use the term "developer" or "dev" to refer to **anyone actively involved in the creation and release of software, regardless of role or title. Additionally, we will define "small" organizations as having < 100 employees, "mid-sized" organizations as having 100-999 employees, and "large" organizations as having 1,000+ employees.*

Major Research Targets

In our 2025 Database Systems survey, we aimed to gather data regarding various topics related to the following major research targets:

1. Database architecture and ecosystem trends
2. Performance, reliability, and data governance
3. Intelligent and emerging database paradigms

In this report, we review some of our key research findings. Many secondary findings of interest are not included here.

Research Target One: Database Architecture and Ecosystem Trends

Our research related to database architecture and ecosystem trends centered around three key topic areas:

1. Database adoption and developer workflows
2. Data architecture decisions and modernization
3. Distributed and multi-cloud design

Database Adoption and Developer Workflows

We asked the following questions:

- *What DBMS types have you worked with?*
- *How often do you write SQL manually?*
- *How often do you use SQL and Python together in the same workflow?*
- *What are your primary use cases for combining Python and SQL?**

**Note: This question was only asked to respondents who did not select "Never" as a response to the question, "How often do you use SQL and Python together in the same workflow?"*

Results:

Figure 1. DBMS type usage [*n*=100]

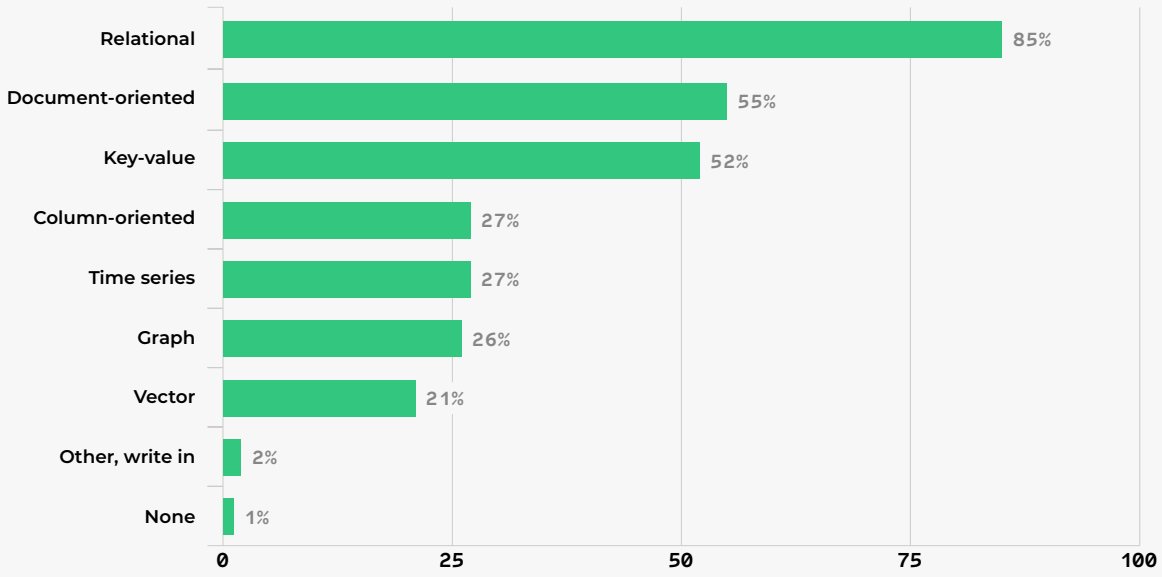


Figure 2. Frequency writing SQL manually [*n*=99]

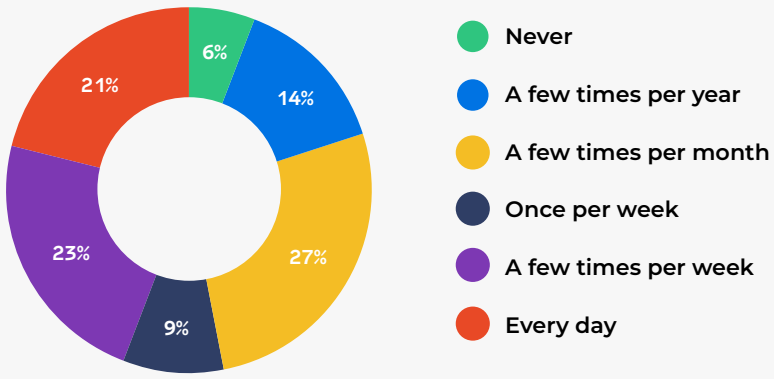


Figure 3. Frequency using SQL and Python together [*n*=100]

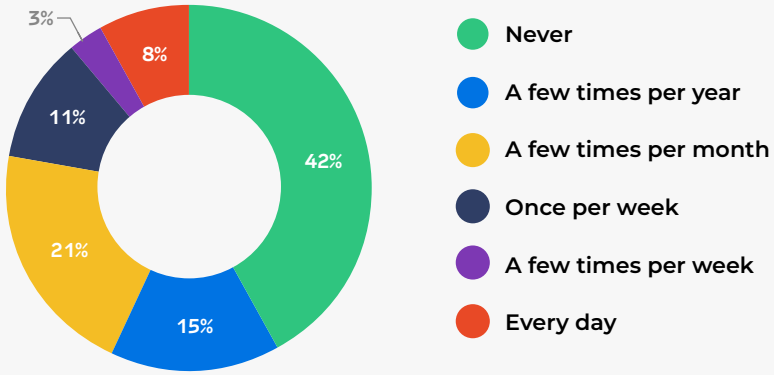
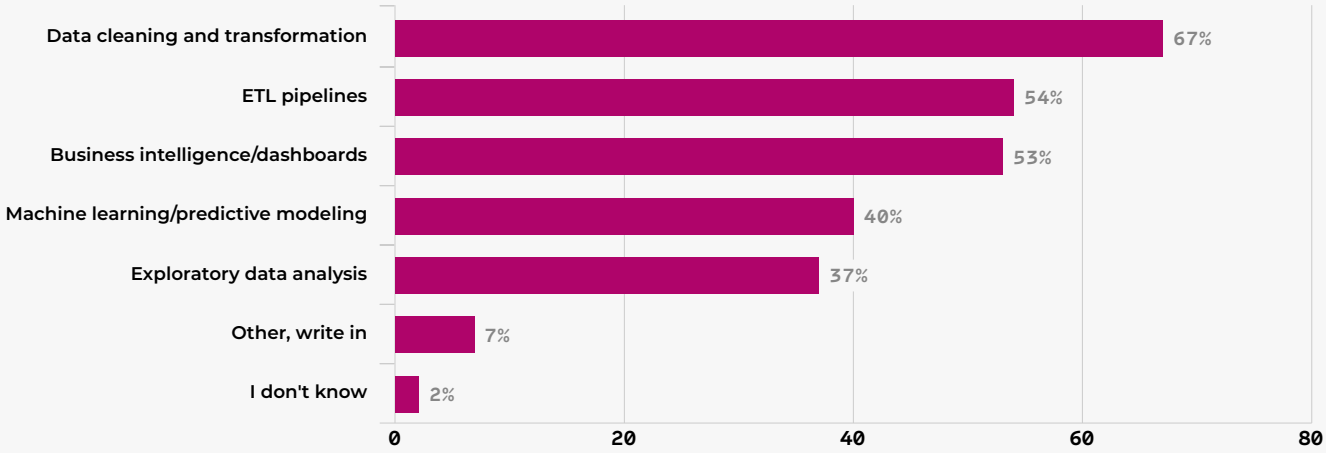


Figure 4. Use cases for combining SQL and Python [*n*=57]



OBSERVATIONS

About three-quarters of respondents (76%) said they have worked with at least two types of DBMSs, and over half (56%) said they have worked with three or more. "Relational" DBMS was the most commonly selected, followed distantly by "Document-oriented" and "Key-value" DBMS. Compared to our 2024 Data Engineering survey, response rates for "Key-value" and "Column-oriented" DBMS both saw significant decreases (see Table 1).

While we found that respondents at mid-sized organizations were much more likely than others to say they have used "Key-value" databases, there were no other significant correlations between respondents' organization size and experience with DBMS types (see Table 2).

Roughly half of respondents (53%) said they write SQL manually at least once per week, and very few respondents said they never have to write SQL manually. There were no significant changes in response rates between the results from 2024 compared to those from this year.

Respondents at mid-sized organizations were less likely than others to say they write SQL manually "A few times per year," and respondents at small organizations were less likely than others to say they write SQL manually "A few times per month" (see Table 3).

Over half of respondents (58%) said they use SQL and Python together to some extent. However, there were significantly more respondents saying they "Never" use SQL and Python together compared to the results of our Data Engineering survey from July of this year (see Table 4).

Respondents at large organizations were more likely than others to report that they "Never" use SQL and Python together (see Table 5).

Three-quarters of respondents (75%) who said they use SQL and Python together selected at least two use cases for doing so, and nearly half (46%) selected three or more. "Data cleaning and transformation," "ETL pipelines," and "Business intelligence/dashboards" were each selected by at least half of respondents as use cases for combining SQL and Python in the same workflow. Compared to our 2025 Data Engineering survey from July, we saw a significant drop in response rates for "Exploratory data analysis" (see Table 6).

Segmenting the results by respondents' organization size, we found the following (see Table 7):

- Respondents at large organizations were more likely than others to say they combined SQL and Python for "Business intelligence/dashboards."
- Respondents at mid-sized organizations were less likely than others to say they combined SQL and Python for "Data cleaning and transformation" and "Exploratory data analysis."
- Respondents at small organizations were less likely than others to say they combined SQL and Python for "ETL pipelines."

CONCLUSIONS

Most devs continue to work across multiple database models, though the relative decline in key-value and column-oriented DBMS usage may indicate a consolidation around relational systems for core workloads. This could be due to the growing maturity and flexibility of modern relational platforms or to the rise of cloud-managed relational services that reduce operational overhead. The consistency in manual SQL use suggests that, despite automation and ORM adoption, SQL remains an indispensable skill across organization sizes.

We were surprised by the overall decrease in the use of SQL and Python together, and will revisit this data in future surveys to determine if that change might reflect an unidentified bias in the results. One such significant bias we found that could have affected these results: a significant increase in respondents identifying as "Technical architects" from our Data Engineering survey in July.

Based on the data we have currently, we think the lower rates of SQL/Python co-use among respondents from large organizations may reflect greater specialization of roles or reliance on dedicated data engineering pipelines that abstract query logic away from analysts and developers. Meanwhile, smaller organizations' higher engagement with mixed SQL/Python workflows may point to more hands-on, cross-functional data practices in those environments.

Finally, the overall decline in "Exploratory data analysis" as a use case for combining SQL and Python could suggest a shift toward more automated or pipeline-oriented usage patterns rather than ad hoc analysis.

ADDITIONAL TABLES

Table 1. DBMS type usage: 2024–2025

DBMS Type	2024	2025	Δ*
Relational	90%	85%	-5%
Document-oriented	53%	55%	+2%
Key-value	64%	52%	-12%
Column-oriented	42%	27%	-15%
Time series	36%	27%	-9%
Graph	32%	26%	-6%
Vector	16%	21%	+5%
Other, write in	5%	2%	-3%
None	1%	1%	0%
n=	107	100	-

**Change shown as percentage points*

Table 2. DBMS type usage by organization size*

	Organization Size			
DBMS Type	1-99	100-999	1,000+	Overall
Relational	84%	85%	88%	85%
Document-oriented	56%	65%	52%	55%
Key-value	36%	73%	45%	52%
Column-oriented	24%	31%	26%	27%
Time series	20%	27%	33%	27%
Graph	28%	23%	31%	26%
Vector	20%	23%	24%	21%
n=	25	26	42	100

**% of columns; "Other, write-in" and "None" rows omitted*

Table 3. Frequency writing SQL manually by organization size*

	Organization Size			
Frequency	1-99	100-999	1,000+	Overall
Every day	24%	23%	19%	21%
A few times per week	20%	23%	21%	22%
Once per week	12%	12%	5%	9%
A few times per month	16%	35%	31%	27%
A few times per year	24%	4%	17%	14%
Never	4%	4%	7%	6%
n=	25	26	42	99

**% of columns*

Table 4. Frequency using SQL and Python together: July–October 2025

Frequency	Jul	Oct	Δ*
Every day	14%	8%	-6%
A few times per week	9%	3%	-6%
Once per week	7%	11%	+4%
A few times per month	22%	21%	-1%
A few times per year	21%	15%	-6%
Never	27%	42%	+15%
n=	106	100	-

**Change shown as percentage points*

Table 5. Frequency using SQL and Python together by organization size*

	Organization Size			
Frequency	1-99	100-999	1,000+	Overall
Every day	8%	4%	10%	8%
A few times per week	4%	0%	5%	3%
Once per week	12%	15%	7%	11%
A few times per month	20%	27%	19%	21%
A few times per year	20%	15%	10%	15%
Never	36%	38%	50%	42%
n=	25	26	42	100

**% of columns*

Table 6. Use cases for combining SQL and Python: July–October 2025

Use Case	Jul	Oct	Δ*
Data cleaning and transformation	59%	67%	+8%
ETL pipelines	51%	54%	+3%
Business intelligence/dashboards	46%	53%	+7%
Machine learning/predictive modeling	46%	40%	-6%
Exploratory data analysis	47%	37%	-10%
Other, write in	4%	7%	+3%
I don't know	0%	2%	+2%
n=	76	57	-

**Change shown as percentage points*

Table 7. Use cases for combining SQL and Python by organization size*

Use Case	Organization Size			Overall
	1-99	100-999	1,000+	
Data cleaning and transformation	73%	50%	71%	67%
Machine learning/predictive modeling	47%	38%	48%	40%
Business intelligence/dashboards	47%	50%	62%	53%
ETL pipelines	40%	56%	62%	54%
Exploratory data analysis	40%	25%	43%	37%
n=	15	16	21	57

**% of columns; "Other, write-in" and "None" rows omitted*

Data Architecture Decisions and Modernization

We asked the following questions:

- *Which query and interaction languages do you use?*
- *What factors most influence your choice of data processing architecture?*
- *In your opinion, what was, or would be, most challenging with migrating to a cloud DBMS?*

Results:

Figure 5. Query/interaction languages used [*n*=100]

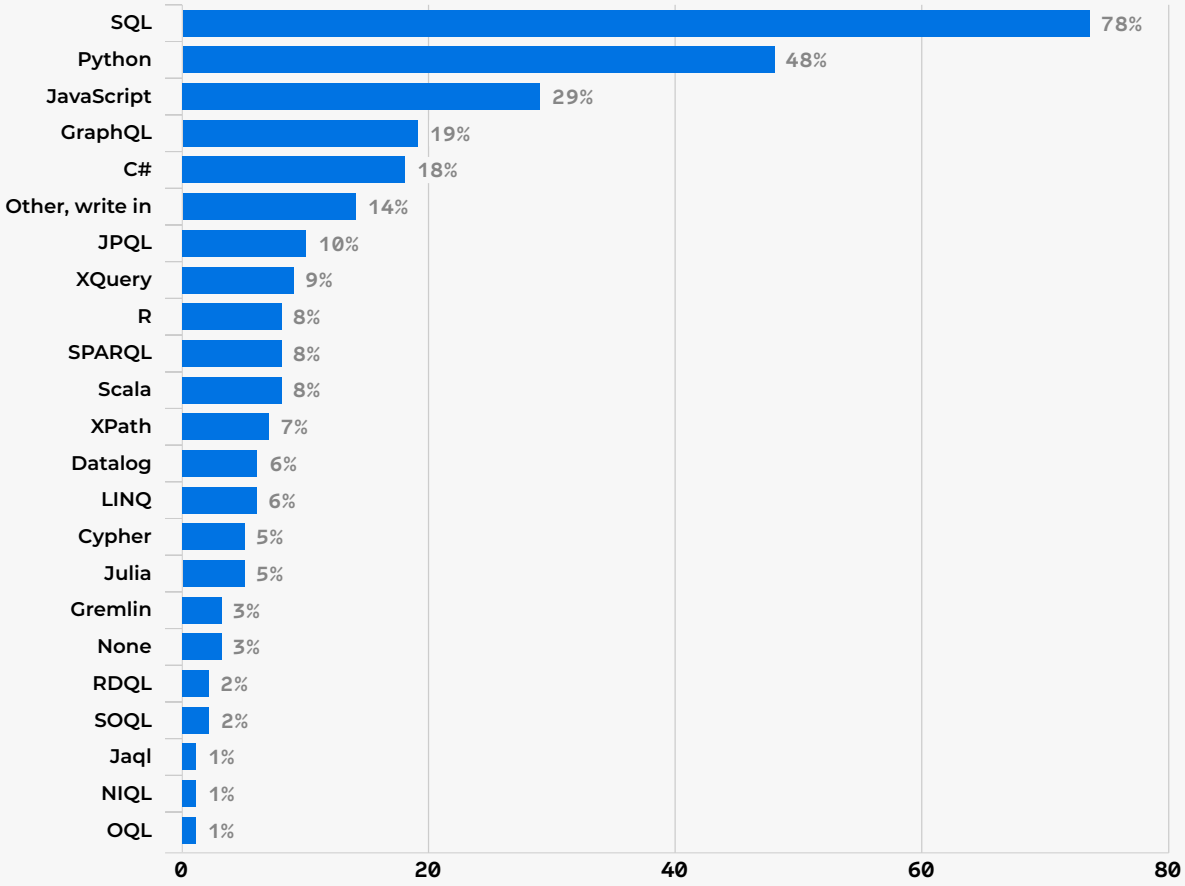
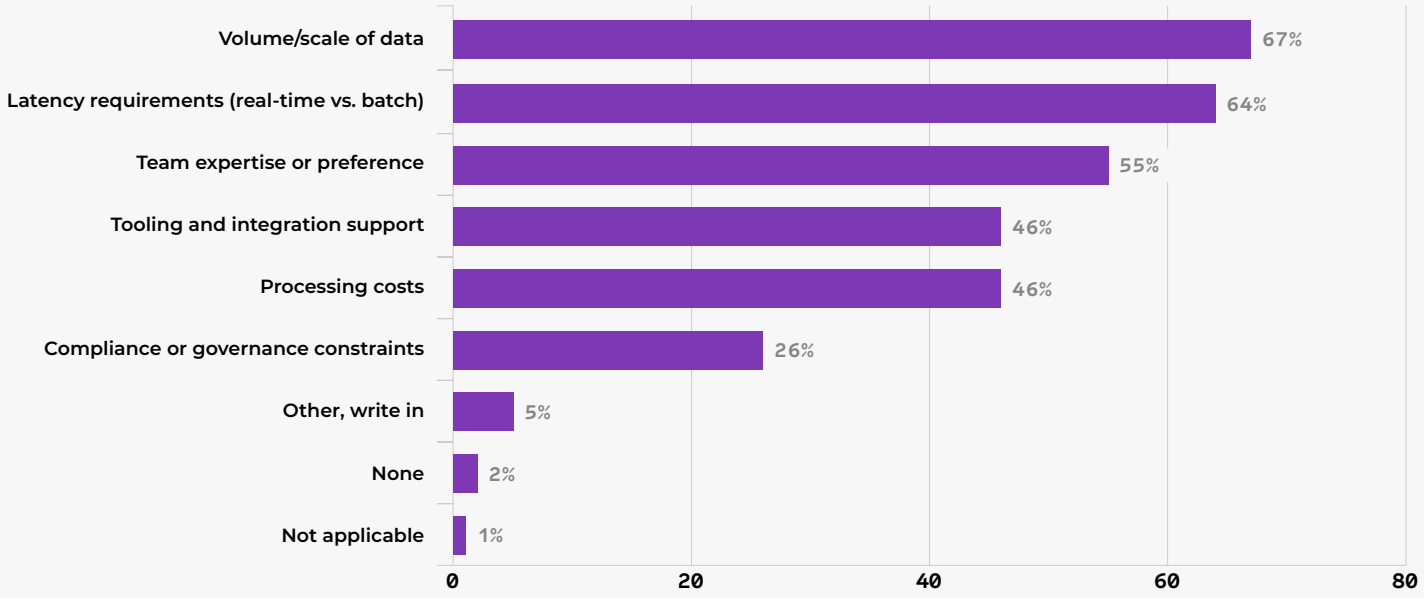


Figure 6. Factors influencing choice of data processing architecture [*n*=98]



OBSERVATIONS

Nearly three-quarters of respondents (72%) said they use two or more data query/interaction languages, and almost half (47%) said they use three or more. SQL was the most commonly selected query language by a wide margin, and 58% of respondents said they use Python and/or JavaScript to interact with data.

Segmented by organization size, we noted the following (see Table 8):

- Respondents at large organizations were more likely than others to say they use "GraphQL" and "LINQ."
- Respondents at mid-sized organizations were less likely than others to say they use "C#."
- Respondents at small organizations were less likely than others to say they use "SQL."

A large majority of respondents (89%) selected at least two of the six provided factors influencing their choice of data processing architecture, and over half (59%) selected three or more. "Volume/scale of data," "Latency requirements (real-time vs. batch)," and "Team expertise or preference" all had response rates over 50%. Compared to the Data Engineering survey results from earlier this year, we saw an increase in the response rate for "Team expertise or preference" and a decrease in the response rate for "Compliance or governance constraints" (see Table 9).

Segmented by organization size, we observed the following (see Table 10):

- Respondents at large organizations were more likely than others to say they factor "Volume/scale of data," "Processing costs," and "Compliance or governance constraints."
- Respondents at mid-sized organizations were more likely than others to say they look at "Team expertise or preference."
- Respondents at small organizations were less likely than others to say they factor "Volume/scale of data," "Team expertise or preference," "Processing costs," and "Compliance or governance constraints."

CONCLUSIONS

Most developers continue to operate in polyglot environments, leveraging multiple query and interaction languages to meet diverse application and analytics needs. SQL remains the universal foundation for database querying, but the widespread use of Python and JavaScript reflects the increasing convergence of traditional database querying and modern data application development, as well as the utility of scripting languages for quickly interacting with data needs that query languages are less suited to handle. Larger organizations' higher adoption of GraphQL and LINQ likely stems from more complex, API-driven architectures and enterprise ecosystems that emphasize typed query interfaces. On the other hand, smaller organizations' lower reported use of SQL may reflect reliance on abstraction layers or lightweight ORM-based development frameworks that obscure direct database interaction.

When it comes to architectural decision-making, the data suggests that performance, scalability, and practical team capabilities are the dominant factors guiding design. The growing emphasis on "Team expertise or preference" may indicate a shift toward developer productivity and maintainability over rigid governance requirements, perhaps due to the normalization of cloud-native and managed database services. Large organizations' continued prioritization of data volume, cost, and compliance considerations aligns with their broader operational and regulatory complexity, whereas smaller organizations appear to optimize primarily for agility and ease of use.

ADDITIONAL TABLES

Table 8. Query/interaction languages used by organization size*

Language	Organization Size			Overall
	1-99	100-999	1,000+	
SQL	60%	81%	88%	78%
Python	44%	54%	48%	48%
JavaScript	20%	38%	29%	29%
GraphQL	16%	15%	26%	19%
C#	24%	8%	24%	18%
Other, write in	16%	15%	12%	14%
JPQL	8%	8%	12%	10%
XQuery	0%	19%	10%	9%
R	12%	8%	2%	8%
SPARQL	8%	8%	10%	8%
Scala	4%	8%	12%	8%
XPath	0%	12%	10%	7%
Datalog	8%	4%	7%	6%
LINQ	0%	0%	14%	6%
n=	25	26	42	100

**% of columns; rows with < 5% "Overall" rate omitted*

Table 9. Factors influencing choice of data processing architecture: July–October 2025

Factor	Jul	Oct	Δ*
Volume/scale of data	59%	67%	+8%
Latency requirements (real-time vs. batch)	55%	64%	+9%
Team expertise or preference	43%	55%	+12%
Tooling and integration support	40%	46%	+6%
Processing costs	54%	46%	-8%
Compliance or governance constraints	36%	26%	-10%
Other, write in	1%	5%	+4%
None	7%	2%	-5%
Not applicable	-	1%	-
n=	109	98	-

**Change shown as percentage points*

Table 10. Factors influencing choice of data processing architecture by organization size*

Factor	Organization Size			Overall
	1-99	100-999	1,000+	
Volume/scale of data	39%	69%	88%	67%
Latency requirements (real-time vs. batch)	57%	65%	74%	64%
Team expertise or preference	35%	69%	57%	55%
Tooling and integration support	43%	50%	45%	46%
Processing costs	26%	42%	62%	46%
Compliance or governance constraints	13%	23%	36%	26%
n=	23	26	42	98

**% of columns; "Other, write-in," "Not applicable," and "None" rows omitted*

Distributed and Multi-Cloud Design

We asked the following questions:

- *Which of the following distributed database architectures has your organization adopted?*
- *How important is multi-cloud or hybrid-cloud portability for your organization's databases?*

Results:

Figure 7. Data storage prevalence: cloud vs. hybrid vs. on-prem [*n*=99]

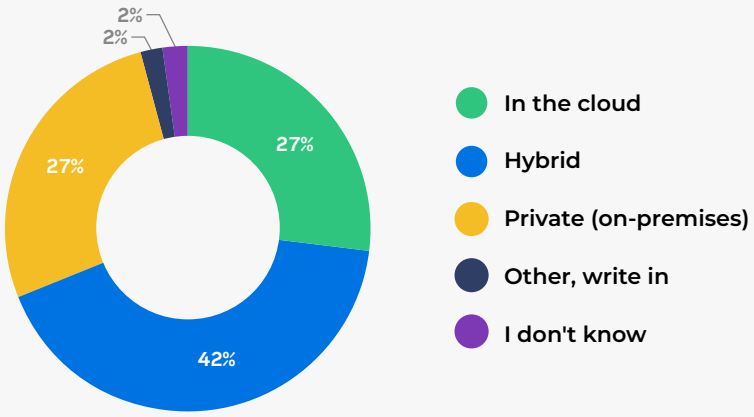


Figure 8. Adoption of distributed database architectures [*n*=97]

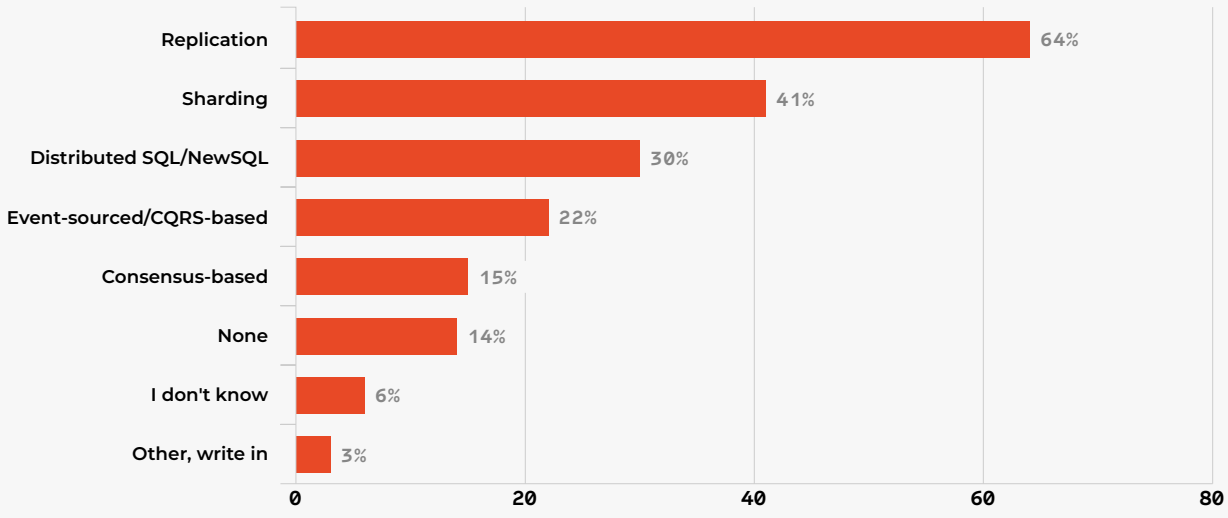
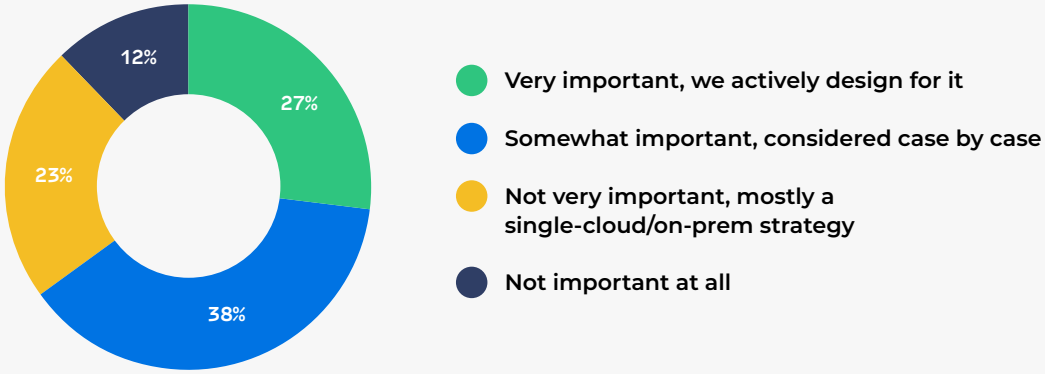


Figure 9. Importance of multi-/hybrid-cloud portability for organizations' databases [*n*=98]



OBSERVATIONS

69% of respondents said that their organization uses some sort of cloud data storage. Compared to our 2025 Data Engineering survey, we saw a significant decrease in the response rate for "In the cloud" data solutions and a significant increase in the response rate for "Private (on-premises)." Response rates for this survey were much more closely aligned with the rates we saw in our 2024 Data Engineering survey (see Table 11).

Segmenting responses by organization size, we noted the following (see Table 12):

- Respondents at large organizations were less likely than others to say their organization uses "Private (on-premises)" data storage.
- Respondents at mid-sized organizations were less likely than others to say their organization stores its data "In the cloud."
- Respondents at small organizations were less likely than others to say their organization uses "Hybrid" data storage.

80% of respondents said their organization has adopted at least one of the five provided distributed database architectures, and about half of respondents (53%) said their organization has adopted two or more. "Replication" was the most commonly selected distributed architecture, followed by "Sharding" and "Distributed SQL/NewSQL."

Segmented by organization size, we found the following:

- Respondents at large organizations were more likely than others to say their organization has adopted "Replication" and "Event-sourced/CQRS-based" architectures.
- Respondents at mid-sized organizations were more likely than others to say their organization has adopted "Distributed SQL/NewSQL."
- Respondents at small organizations were less likely than others to say their organization has adopted "Replication" and "Sharding" architectures. They were more likely than others to say their organization has not adopted any distributed database architectures.

Furthermore, segmented by organizations' cloud data storage, we found:

- Respondents at organizations storing data "In the cloud" were less likely than others to say their organization has adopted "Distributed SQL/NewSQL."
- Respondents at organizations using "Hybrid" cloud data storage were more likely than others to say their organization has adopted "Distributed SQL/NewSQL."
- Respondents at organizations using "Private (on-premises)" cloud data storage were less likely than others to say their organization has adopted "Sharding" and "Event-sourced/CQRS-based."

See Tables 13 and 14, respectively.

Nearly two-thirds of respondents (65%) said that multi-cloud or hybrid-cloud portability is at least somewhat important for their organization's databases.

Respondents at large organizations were more likely than others to say multi-/hybrid-cloud portability is "Somewhat important." On the other hand, respondents at small organizations were less likely to say multi-/hybrid-cloud portability is "Somewhat important" and more likely to say it is "Not very important" or "Not important" (see Table 15).

Furthermore, respondents at organizations using "Private (on-premises)" data storage/database solutions were less likely to say multi-/hybrid-cloud portability is "Very important" and more likely to say it is "Not important." Respondents at organizations with "Hybrid" cloud data storage were more likely to say multi-/hybrid-cloud portability is "Very important" and less likely to say it is "Not very important" (see Table 16).

CONCLUSIONS

While cloud or hybrid storage remains the dominant model for most organizations, the pendulum may be swinging slightly back toward private data environments. The decrease in fully cloud-based adoption compared to earlier this year could reflect growing concerns about cost control, data sovereignty, or security — factors that often prompt renewed investment in on-premises or hybrid solutions. Larger organizations' heavier reliance on cloud storage likely stems from established multi-region infrastructure and managed service ecosystems, whereas smaller and mid-sized organizations may be rebalancing toward simpler, self-managed models.

The widespread adoption of distributed database architectures underscores the ongoing need for scalability, redundancy, and performance in data-intensive applications. Replication remains the foundational strategy, but rising interest in distributed SQL and event-sourced models points to more sophisticated designs aimed at global consistency and real-time responsiveness. Interestingly, the relationship between deployment model and architecture adoption — especially the stronger alignment between hybrid storage and distributed SQL — suggests that hybrid environments are becoming a proving ground for modern, resilient data systems.

Finally, the strong support for multi-cloud and hybrid-cloud portability indicates that even as organizations diversify their storage and architectural approaches, flexibility remains a high priority. Large enterprises' moderate emphasis on portability may reflect pragmatic trade-offs between standardization and complexity, while smaller organizations' lower prioritization could stem from vendor lock-in being less of a concern at their scale. The stronger preference for portability among hybrid adopters reinforces the idea that these organizations view multi-cloud as a strategic advantage, not just an operational contingency.

ADDITIONAL TABLES

Table 11. Data storage prevalence: 2024–2025

Response	2024	Jul 2025	Oct 2025	Jul–Oct Δ*
In the cloud	30%	49%	27%	-22%
Hybrid	48%	37%	42%	+5%
Private (on-premises)	20%	10%	27%	+17%
I don't know	2%	4%	2%	-2%
n=	109	110	99	-

**Change shown as percentage points*

Table 12. Data storage prevalence by organization size*

Storage Type	Organization Size			Overall
	1-99	100-999	1,000+	
In the cloud	32%	15%	33%	28%
Hybrid	32%	42%	50%	44%
Private (on-premises)	36%	42%	17%	28%
n=	22	26	42	93

**% of columns; "I don't know" row omitted*

Table 13. Adoption of distributed database architectures by organization size*

Architecture	Organization Size			Overall
	1-99	100-999	1,000+	
Replication	46%	72%	83%	65%
Sharding	25%	48%	55%	42%
Distributed SQL/NewSQL	33%	44%	25%	31%
Event-sourced/CQRS-based	13%	20%	33%	22%
Consensus-based	13%	20%	18%	16%
None	29%	8%	3%	15%
n=	24	25	40	95

**% of columns; "Other, write-in," and "I don't know" rows omitted*

Table 14. Adoption of distributed database architectures by data storage type*

Architecture	Storage Type			Overall
	Cloud	Hybrid	Private	
Replication	46%	72%	83%	65%
Sharding	25%	48%	55%	42%
Distributed SQL/NewSQL	33%	44%	25%	31%
Event-sourced/CQRS-based	13%	20%	33%	22%
Consensus-based	13%	20%	18%	16%
None	29%	8%	3%	15%
n=	24	25	40	95

**% of columns; "Other, write-in" and "I don't know" rows omitted*

Table 15. Importance of multi-/hybrid-cloud portability by organization size*

Importance	Organization Size			Overall
	1-99	100-999	1,000+	
Very important, we actively design for it	25%	31%	26%	27%
Somewhat important, considered case by case	21%	38%	50%	38%
Not very important, mostly a single-cloud/on-prem strategy	33%	23%	17%	23%
Not important at all	21%	8%	7%	12%
n=	24	26	42	98

**% of columns*

Table 16. Importance of multi-/hybrid-cloud portability by data storage type*

Importance	Storage Type			Overall
	Cloud	Hybrid	Private	
Very important	23%	44%	4%	27%
Somewhat important	46%	39%	31%	38%
Not very important	27%	15%	35%	23%
Not important at all	4%	2%	31%	12%
n=	26	41	26	98

**% of columns*

Research Target Two: Observability, Performance, and Reliability

Our research related to database observability, performance, and reliability trends centered around two key topic areas:

1. Database monitoring and observability
2. Data resilience and performance management

Database Monitoring and Observability

We asked the following questions:

- *Which of the following tools does your organization use for data observability?*
- *Which of the following types of database metrics do you find most important to monitor?*

Results:

Figure 10. Data observability tool usage [*n*=96]

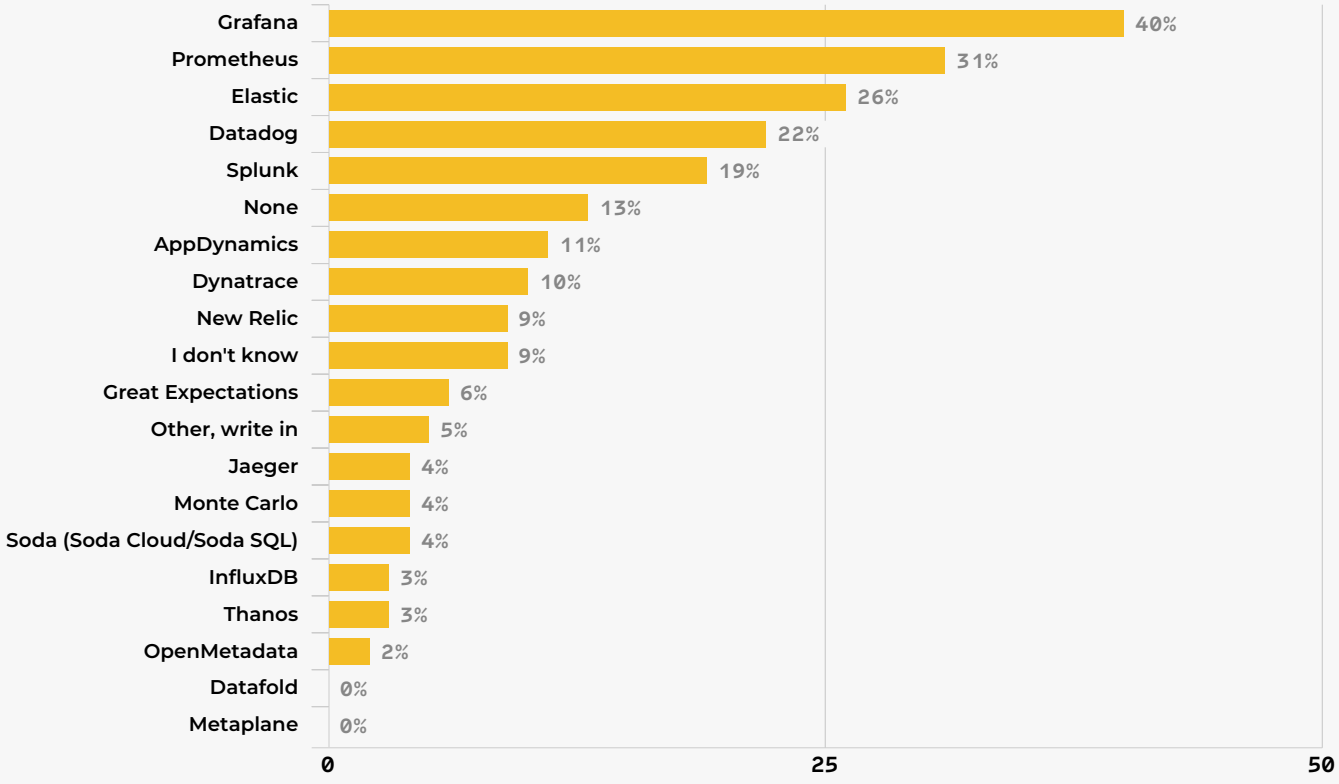
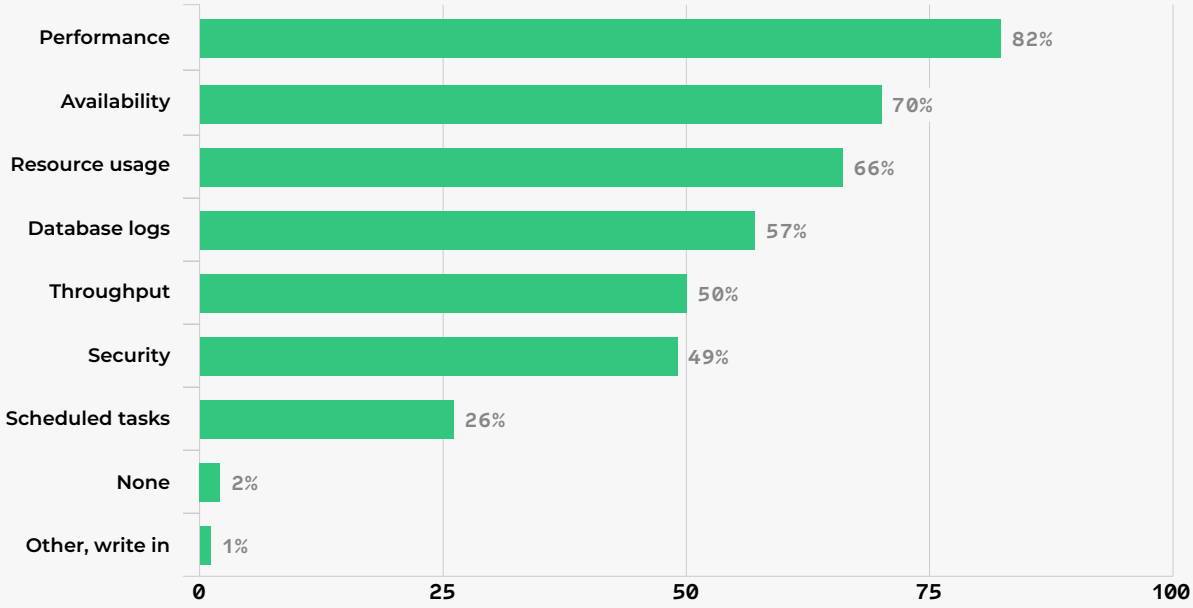


Figure 11. Important database metrics to monitor [*n*=96]



OBSERVATIONS

Over three-quarters of respondents (77%) said their organization uses at least one of the data observability tools we listed, and over half (58%) said their organization uses two or more. "Grafana" was the most commonly selected tool, followed by "Prometheus" and "Elastic." Compared to July's Data Engineering survey, there was a significant increase in the response rate for "Grafana," which returned to the rate we saw in 2024 (41%).

Segmenting results by respondents' organization size, we noted the following (see Table 17):

- Respondents at large organizations were more likely than others to say their organization uses "Datadog," "Splunk," and "Dynatrace" for data observability.
- Respondents at mid-sized organizations were more likely than others to say their organization uses "Prometheus" and "Elastic."
- Respondents at small organizations were less likely than others to say their organization uses "Grafana" and "Prometheus." They were more likely than others to say their organization does not use any tools for data observability.

Additionally, respondents at organizations using "Hybrid" cloud data storage were more likely than others to say their organization uses "Splunk" and "AppDynamics" for data observability. Respondents at organizations using "Private (on-premises)" cloud data storage were less likely than others to say their organization uses "Prometheus" for data observability (see Table 18).

Almost all respondents (97%) found at least two of the seven provided metrics important for database monitoring, and over half (56%) selected four or more. "Performance," "Availability," and "Resource usage" were the most commonly selected metrics.

Segmenting responses by organization size, we found the following (see Table 19):

- Respondents at large organizations were more likely than others to find "Availability" as an important database monitoring metric.
- Respondents at mid-sized organizations were more likely than others to view "Scheduled tasks" as an important metric.
- Respondents at small organizations were less likely than others to find "Performance" and "Throughput" as important metrics.

CONCLUSIONS

Data observability has become a well-established component of database management across organizations, though adoption patterns still vary by size and infrastructure model. Larger organizations' stronger reliance on full-featured commercial platforms like Datadog, Splunk, and Dynatrace is likely due to their broader operational complexity and compliance needs, while mid-sized organizations' preference for Prometheus and Elastic reflects a balance between flexibility, cost, and self-managed control.

Smaller organizations' lower adoption rates for observability tools — and their higher likelihood of having no formal observability stack — may point to resource constraints, a lower bar for the organization's perceived needs, or a reliance on built-in database monitoring features rather than dedicated systems. The stronger correlation between hybrid cloud users and adoption of enterprise observability tools such as Splunk and AppDynamics suggests that hybrid infrastructures, with their distributed complexity, are driving demand for more integrated monitoring solutions.

The nearly universal agreement on the importance of multiple monitoring metrics reinforces that observability is viewed as a multi-dimensional practice rather than a simple uptime check. Emphasis on performance, availability, and resource usage highlights enduring operational priorities, while the attention mid-sized organizations give to scheduled tasks may reflect more automated or pipeline-driven workloads. In contrast, smaller organizations' relatively lower focus on performance and throughput could indicate simpler data environments or fewer high-scale workloads where such metrics become critical.

ADDITIONAL TABLES

Table 17. Data observability tool usage by organization size*

Tool	Organization Size			Overall
	1-99	100-999	1,000+	
Grafana	21%	50%	49%	40%
Prometheus	8%	54%	34%	31%
Elastic	17%	38%	24%	26%
Datadog	13%	19%	32%	22%
Splunk	13%	12%	29%	19%
AppDynamics	4%	12%	17%	11%
Dynatrace	0%	4%	22%	10%
None	33%	0%	0%	13%
n=	24	26	41	96

**% of columns; rows with < 10% "Overall" rate omitted*

Table 18. Data observability tool usage by data storage type*

Tool	Storage Type			Overall
	Cloud	Hybrid	Private	
Grafana	38%	45%	38%	40%
Prometheus	23%	38%	35%	31%
Elastic	23%	30%	27%	26%
Datadog	27%	23%	19%	22%
Splunk	8%	33%	12%	19%
AppDynamics	4%	20%	8%	11%

CONTINUES ON NEXT PAGE

Tool	Storage Type			Overall
	Cloud	Hybrid	Private	
Dynatrace	4%	15%	12%	10%
None	8%	5%	15%	13%
n=	26	40	26	96

**% of columns; rows with < 10% "Overall" rate omitted*

Table 19. Important database metrics to monitor by organization size*

Metric	Organization Size			Overall
	1-99	100-999	1,000+	
Performance	70%	96%	88%	82%
Availability	61%	65%	81%	70%
Resource usage	70%	62%	67%	66%
Database logs	65%	62%	55%	57%
Throughput	30%	54%	60%	50%
Security	57%	46%	52%	49%
Scheduled tasks	17%	42%	21%	26%
n=	23	26	42	96

**% of columns; "Other, write-in" and "None" rows omitted*

Data Resilience and Performance Management

We asked the following questions:

- *What strategies does your organization use to ensure database high availability and fault tolerance?*
- *What strategies or tools does your organization use to manage data duplication, fragmentation, or sprawl in its data architecture?*
- *Which of the following challenges related to data sprawl have you encountered in your current or past projects?*

Results:

Figure 12. Strategies to ensure database high availability and fault tolerance [*n*=94]

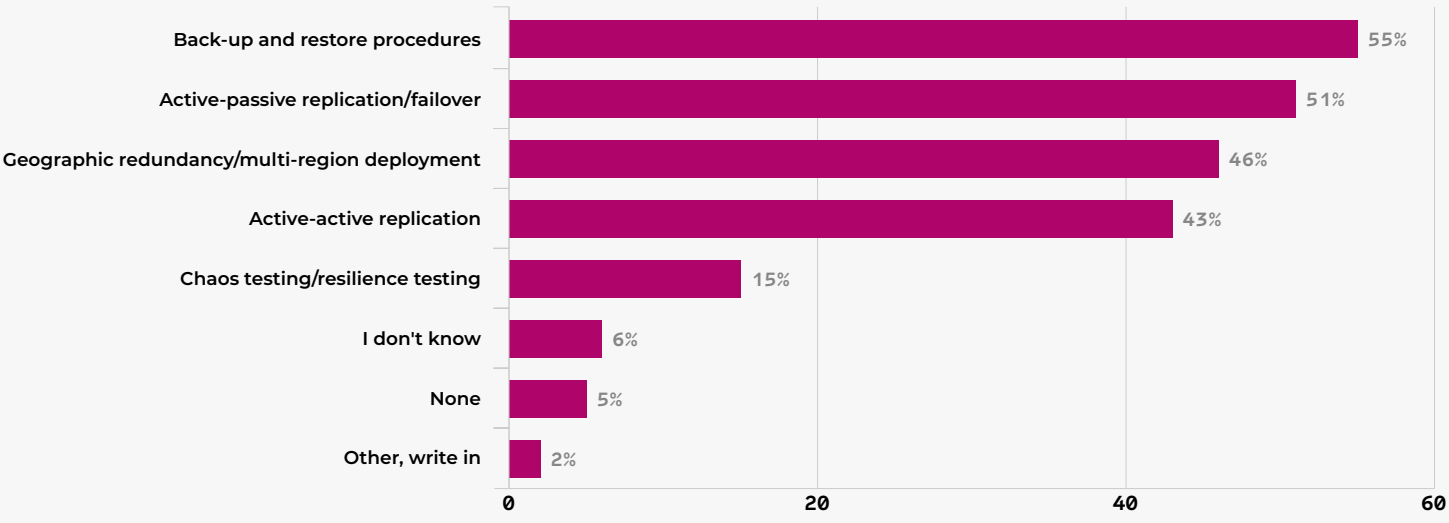


Figure 13. Strategies/tools for managing data [*n*=97]

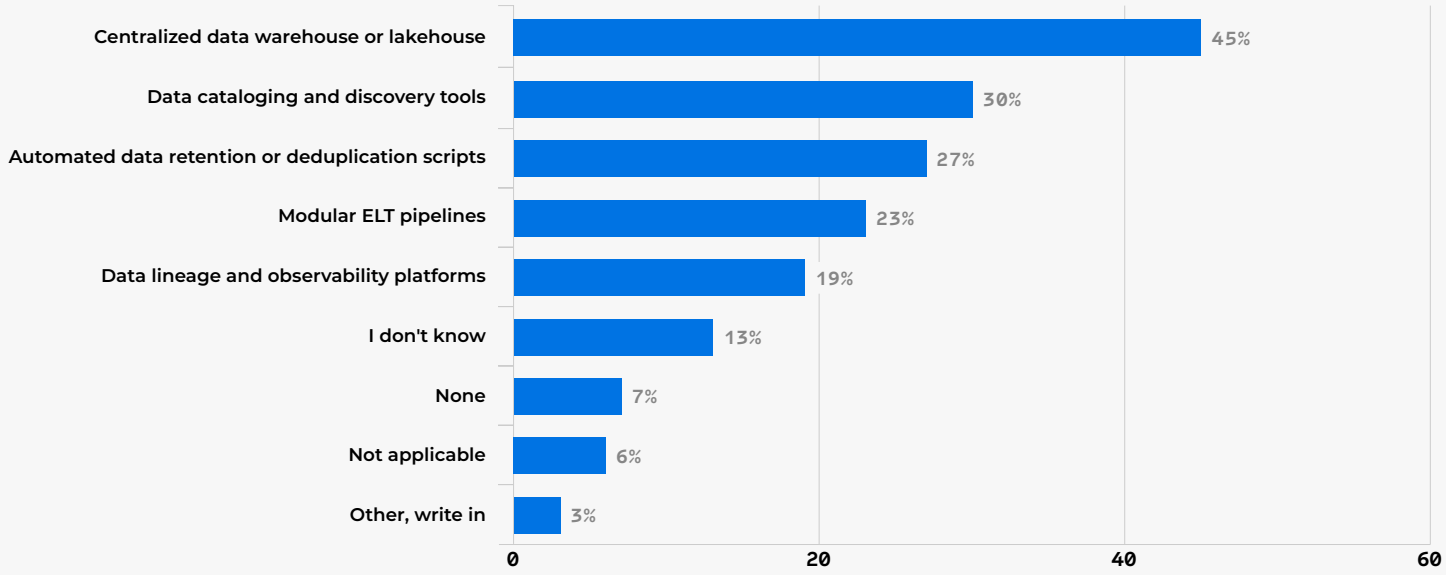
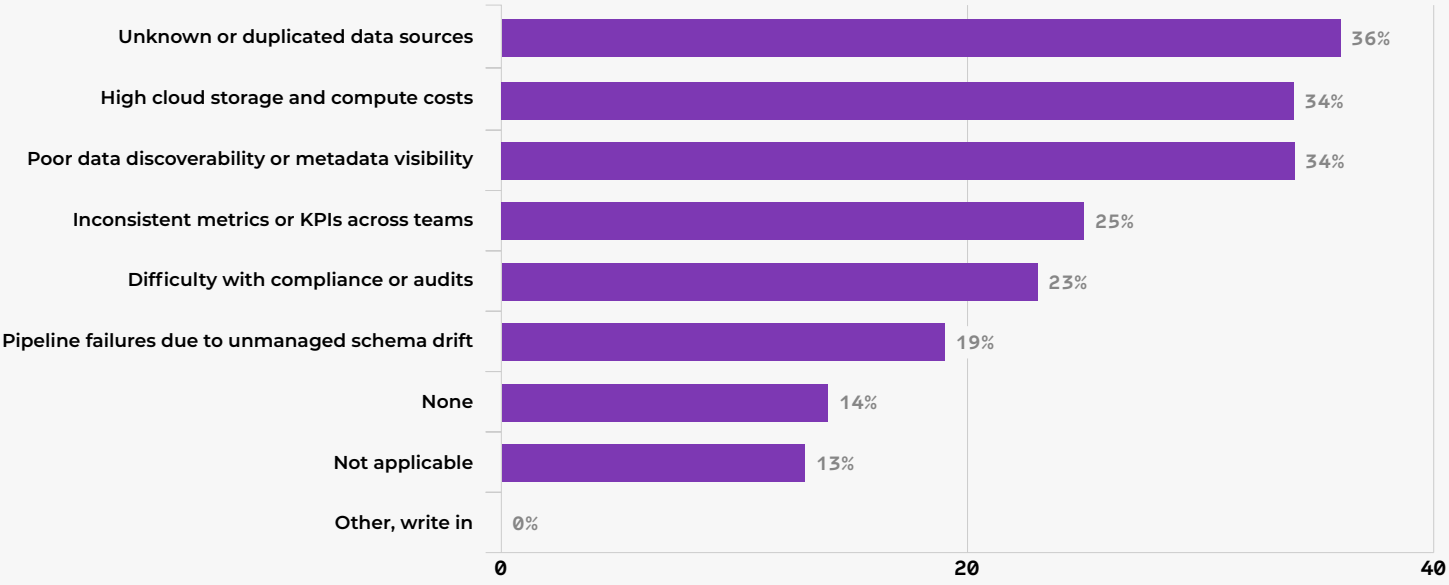


Figure 14. Data sprawl challenges encountered [*n*=96]



OBSERVATIONS

86% of respondents said their organization uses at least one of the five provided strategies for ensuring high availability/fault tolerance, and about two-thirds (65%) said they use two or more. Both "Back-up and restore procedures" and "Active-passive replication/failover" were selected by over half of respondents, while "Chaos testing/resilience testing" had a much lower response rate than the other four strategies.

We found that respondents at large organizations were more likely than others to say their organization uses "Active-passive replication/failover" and "Active-active replication." Respondents at small organizations were less likely than others to say their organization uses "Geographic redundancy/multi-region deployment" and "Active-active replication" (see Table 20).

Furthermore, respondents at organizations using "Hybrid" data storage were more likely than others to say their organization uses "Active-passive replication/failover" and less likely to say their organization uses "Back-up and

restore procedures." Respondents at organizations using "Private (on-premises)" data storage were less likely than others to say their organization uses "Geographic redundancy/multi-region deployment" and "Active-active replication" (see Table 21).

- 71% of respondents said their organization uses at least one of the five provided strategies for managing data duplication, fragmentation, or sprawl, and nearly half (45%) said their organization uses two or more. Using a "Centralized data warehouse or lakehouse" was the most commonly selected strategy. There were no significant changes in results from our July Data Engineering survey.

Respondents at large organizations were more likely than others to say their organization uses "Data cataloging and discovery tools," and respondents at small organizations were less likely than others to say their organization uses "Data lineage and observability platforms" (see Table 22).

Additionally, respondents at organizations using "Private (on-premises)" data storage were less likely than others to say their organization uses a "Centralized data warehouse or lakehouse" and "Data cataloging and discovery tools." They were more likely than others to say their organization does not use strategies or tools to manage duplication, fragmentation, or sprawl in their data architecture (see Table 23).

- About three-quarters of respondents (74%) said they have encountered at least one challenge related to data sprawl, and over half (56%) said they have encountered two or more. "Unknown or duplicated data sources," "High cloud storage and compute costs," and "Poor data discoverability or metadata visibility" were the most commonly selected challenges.

Compared to our Data Engineering survey from July, we saw significant decreases in response rates for "Unknown or duplicated data sources" and "Inconsistent metrics or KPIs across teams," and an increase in the percentage of respondents that said they have not experienced any data sprawl challenges (see Table 24).

Segmented by organization size, we found the following (see Table 25):

- Respondents at large organizations were more likely than others to say they have encountered "Inconsistent metrics or KPIs across teams."
- Respondents at small organizations were less likely than others to say they have encountered "Unknown or duplicated data sources," "High cloud storage and compute costs," "Poor data discoverability or metadata visibility," and "Pipeline failures due to unmanaged schema drift." They were more likely than others to select "Not applicable."

We also found that respondents at organizations using "Hybrid" data storage were more likely than others to say they encountered challenges with "Inconsistent metrics or KPIs across teams," while respondents at organizations storing data "In the cloud" were more likely than others to say they found "High cloud storage and compute costs" challenging.

CONCLUSIONS

While most organizations have adopted core strategies for ensuring database resilience and data integrity, there remains a notable divide in sophistication and implementation depth across organization sizes and infrastructure types. The widespread reliance on traditional approaches such as back-up/restore and active-passive replication likely reflects their accessibility and familiarity, while the low adoption of chaos or resilience testing indicates that proactive fault-tolerance validation remains a niche practice.

Larger organizations' preference for replication-based strategies aligns with their need for continuous uptime and distributed systems resilience, whereas smaller organizations' lower use of multi-region or active-active architectures may stem from cost and complexity barriers.

Hybrid storage environments appear to encourage more dynamic high-availability configurations, possibly because managing across cloud and private systems requires automated failover and replication capabilities. On the other hand, for organizations with on-premises data solutions, lower use of geographically redundant architectures reinforces the limits of self-hosted environments when scaling for resiliency.

In terms of managing data duplication, fragmentation, and sprawl, the continued dominance of centralized data warehouses and lakehouses points to an industry preference for consolidation over distributed governance approaches. However, the lag in adoption of cataloging and lineage tools — especially among smaller or on-premises organizations — suggests that data visibility and accountability remain uneven across the ecosystem.

The persistence of sprawl-related challenges, particularly unknown or duplicated data sources and high cloud costs, highlights ongoing struggles to balance scalability with control. The decline in several challenge categories compared to July's results may indicate incremental improvements in governance or cost optimization practices. Still, the continued prevalence of these issues at larger and hybrid-cloud organizations suggests that as data ecosystems grow more interconnected, fragmentation risks are evolving faster than most teams' ability to manage them.

ADDITIONAL TABLES

Table 20. Strategies to ensure database high availability and fault tolerance by organization size*

Strategy	Organization Size			Overall
	1-99	100-999	1,000+	
Back-up and restore procedures	57%	64%	56%	55%
Active-passive replication/failover	43%	40%	66%	51%
Geographic redundancy/multi-region deployment	30%	56%	51%	46%
Active-active replication	22%	40%	61%	43%
Chaos testing/resilience testing	13%	16%	17%	15%
n=	23	25	41	94

**% of columns; "Other, write-in," "I don't know," and "None" rows omitted*

Table 21. Strategies to ensure database high availability and fault tolerance by data storage type*

Strategy	Storage Type			Overall
	Cloud	Hybrid	Private	
Back-up and restore procedures	63%	49%	64%	55%
Active-passive replication/failover	42%	63%	44%	51%
Geographic redundancy/multi-region deployment	54%	54%	32%	46%
Active-active replication	46%	54%	24%	43%
Chaos testing/resilience testing	21%	12%	12%	15%
n=	24	41	25	94

**% of columns; "Other, write-in," "I don't know," and "None" rows omitted*

Table 22. Strategies/tools for managing data by organization size*

Strategy/Tool	Organization Size			Overall
	1-99	100-999	1,000+	
Centralized data warehouse or lakehouse	38%	46%	55%	45%
Data cataloging and discovery tools	25%	19%	43%	30%
Automated data retention or deduplication scripts	33%	27%	24%	27%
Modular ELT pipelines	17%		29%	23%
Data lineage and observability platforms	8%	23%	24%	19%
I don't know	17%	12%	14%	13%
n=	24	26	42	97

**% of columns; "Other, write-in," "Not applicable," and "None" rows omitted*

Table 23. Strategies/tools for managing data by data storage type*

Strategy/Tool	Storage Type			Overall
	Cloud	Hybrid	Private	
Centralized data warehouse or lakehouse	54%	51%	31%	45%
Data cataloging and discovery tools	35%	41%	12%	30%
Automated data retention or deduplication scripts	27%	24%	31%	27%
Modular ELT pipelines	31%	22%	19%	23%
Data lineage and observability platforms	15%	24%	15%	19%
I don't know	15%	17%	8%	13%
n=	26	41	26	97

**% of columns; "Other, write-in," "Not applicable," and "None" rows omitted*

Table 24. Data sprawl challenges encountered: July–October 2025

Challenge	Jul	Oct	Δ*
Unknown or duplicated data sources	56%	36%	-19%
High cloud storage and compute costs	42%	34%	-7%
Poor data discoverability or metadata visibility	39%	34%	-5%
Inconsistent metrics or KPIs across teams	48%	25%	-23%
Difficulty with compliance or audits	25%	23%	-2%
Pipeline failures due to unmanaged schema drift	22%	19%	-3%
Other, write in	0%	0%	0%
None	1%	14%	+12%
I don't know	3%	-	-
Not applicable	-	13%	-
n=	77	96	-

**Change shown as percentage points*

Table 25. Data sprawl challenges encountered by organization size*

Challenge	Organization Size			Overall
	1-99	100-999	1,000+	
Unknown or duplicated data sources	21%	40%	43%	36%
High cloud storage and compute costs	17%	44%	43%	34%
Poor data discoverability or metadata visibility	21%	44%	40%	34%
Inconsistent metrics or KPIs across teams	13%	20%	38%	25%
Difficulty with compliance or audits	25%	20%	26%	23%
Pipeline failures due to unmanaged schema drift	8%	20%	26%	19%
None	17%	12%	10%	14%
Not applicable	29%	4%	7%	13%
n=	24	25	42	96

**% of columns; "Other, write-in" row omitted*

Table 26. Data sprawl challenges encountered by data storage type*

Challenge	Storage Type			Overall
	Cloud	Hybrid	Private	
Unknown or duplicated data sources	31%	38%	46%	36%
High cloud storage and compute costs	54%	33%	23%	34%
Poor data discoverability or metadata visibility	35%	38%	35%	34%
Inconsistent metrics or KPIs across teams	19%	40%	12%	25%
Difficulty with compliance or audits	19%	28%	19%	23%
Pipeline failures due to unmanaged schema drift	23%	18%	19%	19%
None	15%	8%	19%	14%
Not applicable	4%	10%	19%	13%
n=	26	40	26	96

**% of columns*

Research Target Three: Intelligent and Emerging Database Paradigms

Our research related to intelligent and emerging database paradigms centered around two key topic areas:

1. Databases for AI/ML
2. New and emerging paradigms

Databases for AI/ML

We asked the following questions:

- *How do you currently supply data or context to support generative or agentic AI applications?*
- *Which of the following practices do you use to ensure data quality when preparing datasets for AI or ML models?*

Results:

Figure 15. Methods for supplying data to generative/agentic AI apps [*n*=96]

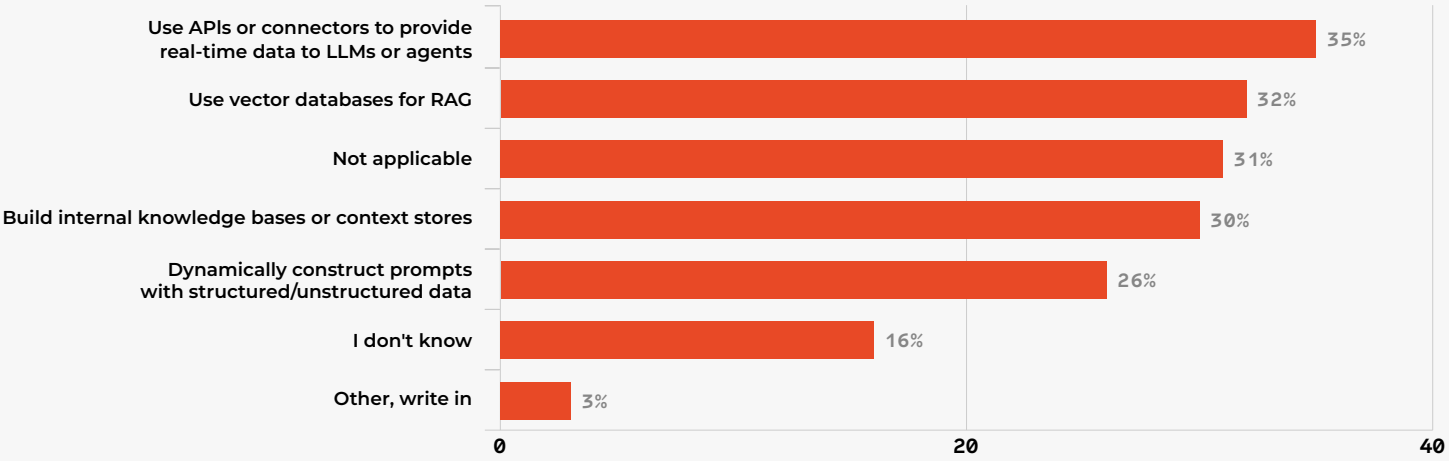
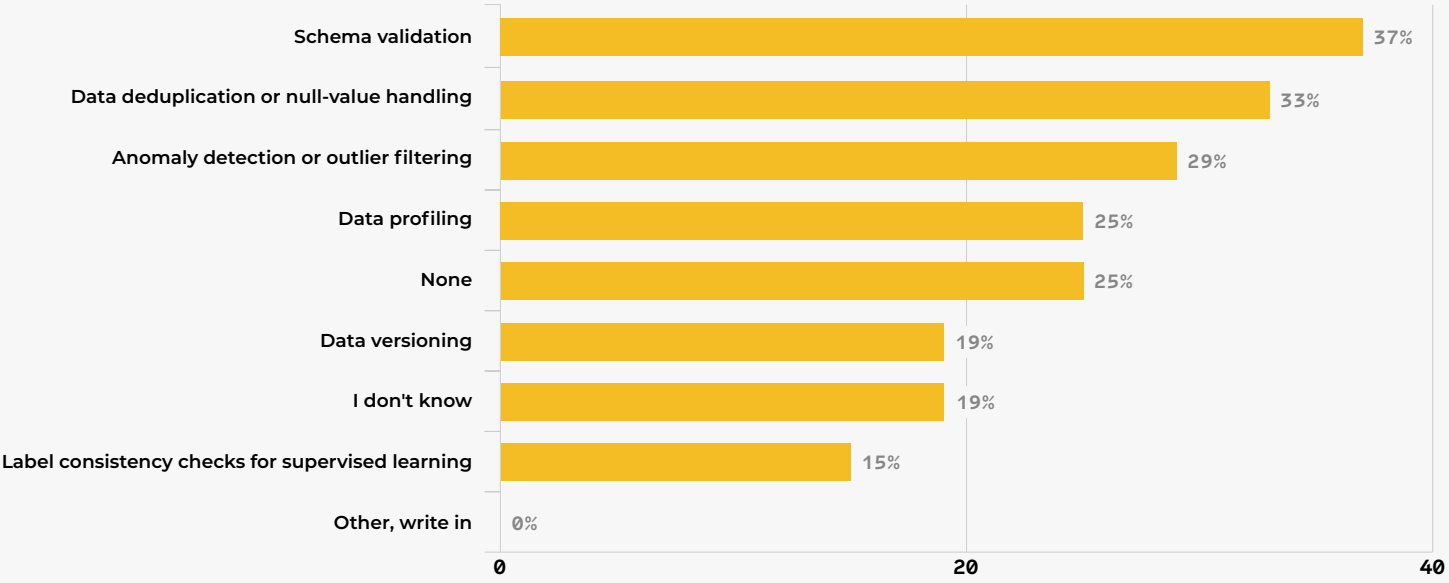


Figure 16. Practices for ensuring data quality for AI/ML models [*n*=97]



OBSERVATIONS

About half of respondents (52%) said they use at least one of the four provided methods for supplying data to generative/agentic AI applications, the most commonly selected of which was the "Use [of] APIs or connectors to provide real-time data." Compared to July's Data Engineering findings, the only significant change we observed was an increase in the percentage of respondents who selected "Not applicable" (see Table 27).

Segmenting results by respondents' organization size, we observed the following (see Table 28):

- Respondents at large organizations were more likely than others to say they "Build internal knowledge bases or context stores."
- Respondents at small organizations were less likely than others to say they "Use APIs/connectors to provide real-time data." They were more likely than others to select "Not applicable."

Segmented by data storage type, we found the following (see Table 29):

- Respondents at organizations storing data "In the cloud" were less likely than others to say they did not know how they [their organization] supplies data context for AI.
- Respondents at organizations using "Hybrid" cloud data storage were more likely than others to say they "Use APIs/connectors to provide real-time data" and "Dynamically construct prompts." They were less likely than others to select "Not applicable."
- Respondents at organizations using "Private (on-premises)" cloud data storage were less likely than others to say they "Use vector databases for RAG."

Over half of respondents (57%) said they use at least one of the six AI data quality practices we suggested, and nearly half (48%) said they use two or more. Compared to the data we presented in our July Data Engineering report, almost every practice for ensuring data quality for AI/ML models saw decreased response rates, excepting "Data profiling" (see Table 30).

Segmenting results by organization size, we noted the following (see Table 31):

- Respondents at large organizations were more likely than others to say they use "Schema validation" and "Data deduplication or null-value handling."
- Respondents at mid-sized organizations were more likely than others to say they did not know what practices they [their organization] uses for ensuring AI/ML data quality.
- Respondents at small organizations were less likely than others to say they use "Schema validation." They were more likely than others to select "None."

Segmenting results by data storage type, we observed the following (see Table 32):

- Respondents at organizations using “Hybrid” cloud data storage were more likely than others to say “Anomaly detection or outlier filtering” and “Data versioning.” They were less likely than others to select “None."
- Respondents at organizations using “Private (on-premises)” cloud data storage were less likely than others to say “Schema validation” and “Anomaly detection or outlier filtering."

CONCLUSIONS

Generative and agentic AI integration is beginning to take hold, but adoption patterns remain uneven and somewhat tentative across organizations. The fact that only about half of respondents reported using any formal method to supply data to AI applications indicates that many teams may still be in early exploration or experimentation stages. The rise in “Not applicable” responses compared to July’s results supports the idea that many organizations have yet to formally make AI workflows that require real-time or contextual data feeding operational. Larger organizations’ greater use of internal knowledge bases reflects their capacity to invest in more sophisticated, context-aware infrastructure, while smaller organizations’ lower use of APIs or connectors may stem from limited technical capacity or uncertainty about reliable data delivery mechanisms.

Patterns across storage architectures further reinforce this divide: Hybrid environments appear to enable more advanced and dynamic approaches like real-time APIs and prompt construction, possibly because they must manage data movement across systems; they also tend to reflect higher overall technical maturity. On the other hand, on-premises environments continue to lag in adopting newer paradigms like vector databases for retrieval-augmented generation. This may point to slower modernization cycles or limited alignment between legacy data systems and emerging AI infrastructure.

The picture for AI data quality practices is similar: There is broad acknowledgment of their importance but uneven and potentially declining adoption across the industry. The general decrease in reported use of AI data quality methods since July could point to tool fatigue, budget constraints, or simply a reallocation of focus as teams move from experimentation to implementation. Larger organizations’ continued emphasis on schema validation and deduplication suggests they are more likely to have formal governance processes, while mid-sized and smaller organizations appear less structured or uncertain about what data quality safeguards are even in place.

Finally, differences in AI data quality practices based on data storage strategies reinforce a similar maturity gradient: Hybrid organizations that have to navigate complex, distributed data pipelines seem more likely to adopt advanced quality control techniques like anomaly detection and data versioning, while on-premises organizations remain focused on maintaining legacy stability rather than innovating on AI readiness.

ADDITIONAL TABLES

Table 27. Methods for supplying data to generative/agentic AI apps: July–October 2025

Method	Jul	Oct	Δ*
Use APIs or connectors to provide real-time data to LLMs or agents	39%	35%	-4%
Use vector databases for RAG	36%	32%	-4%
Build internal knowledge bases or context stores	28%	30%	+2%
Dynamically construct prompts with structured/unstructured data	33%	26%	-7%
Other, write in	4%	3%	-1%
I don’t know	14%	16%	+2%
Not applicable	16%	31%	+15%
n=	107	96	-

**Change shown as percentage points*

Table 28. Methods for supplying data to generative/agentic AI apps by organization size*

Method	Organization Size			Overall
	1-99	100-999	1,000+	
Use APIs or connectors to provide real-time data to LLMs or agents	25%	46%	39%	35%
Use vector databases for RAG	25%	42%	34%	32%
Build internal knowledge bases or context stores	21%	27%	39%	30%
Dynamically construct prompts with structured/unstructured data	21%	27%	32%	26%
I don't know	13%	19%	17%	16%
Not applicable	46%	23%	22%	31%
n=	24	26	41	96

**% of columns; "Other, write-in" row omitted*

Table 29. Methods for supplying data to generative/agentic AI apps by data storage type*

Method	Storage Type			Overall
	Cloud	Hybrid	Private	
Use APIs/connectors to provide real-time data	31%	45%	27%	35%
Use vector databases for RAG	35%	43%	15%	32%
Build internal knowledge bases/context stores	31%	30%	31%	30%
Dynamically construct prompts	19%	38%	19%	26%
I don't know	4%	25%	15%	16%
Not applicable	42%	10%	46%	31%
n=	26	40	26	96

**% of columns; "Other, write-in" row omitted*

Table 30. Practices for ensuring data quality for AI/ML models: July–October 2025

Practice	Jul	Oct	Δ*
Schema validation	47%	37%	-10%
Data deduplication or null-value handling	47%	33%	-14%
Anomaly detection or outlier filtering	44%	29%	-16%
Data profiling	24%	25%	+1%
Data versioning	31%	19%	-13%
Label consistency checks for supervised learning	30%	15%	-14%
Other, write in	1%	0%	-1%
I don't know	15%	19%	+4%
None	12%	25%	+13%
n=	108	97	-

**Change shown as percentage points*

Table 31. Practices for ensuring data quality for AI/ML models by organization size*

Practice	Organization Size			Overall
	1-99	100-999	1,000+	
Schema validation	17%	31%	55%	37%
Data deduplication or null-value handling	25%	23%	45%	33%
Anomaly detection or outlier filtering	33%	27%	31%	29%
Data profiling	29%	23%	26%	25%
Data versioning	13%	23%	21%	19%
Label consistency checks for supervised learning	17%	19%	14%	15%
I don't know	13%	31%	17%	19%
None	38%	19%	14%	25%
n=	24	26	42	97

**% of columns; "Other, write-in" row omitted*

Table 32. Practices for ensuring data quality for AI/ML models by data storage type*

Practice	Storage Type			Overall
	Cloud	Hybrid	Private	
Schema validation	46%	41%	27%	37%
Data deduplication or null-value handling	31%	37%	31%	33%
Anomaly detection or outlier filtering	27%	39%	15%	29%
Data profiling	23%	32%	19%	25%
Data versioning	15%	27%	12%	19%
Label consistency checks for supervised learning	15%	15%	19%	15%
I don't know	15%	20%	23%	19%
None	27%	12%	35%	25%
n=	26	41	26	97

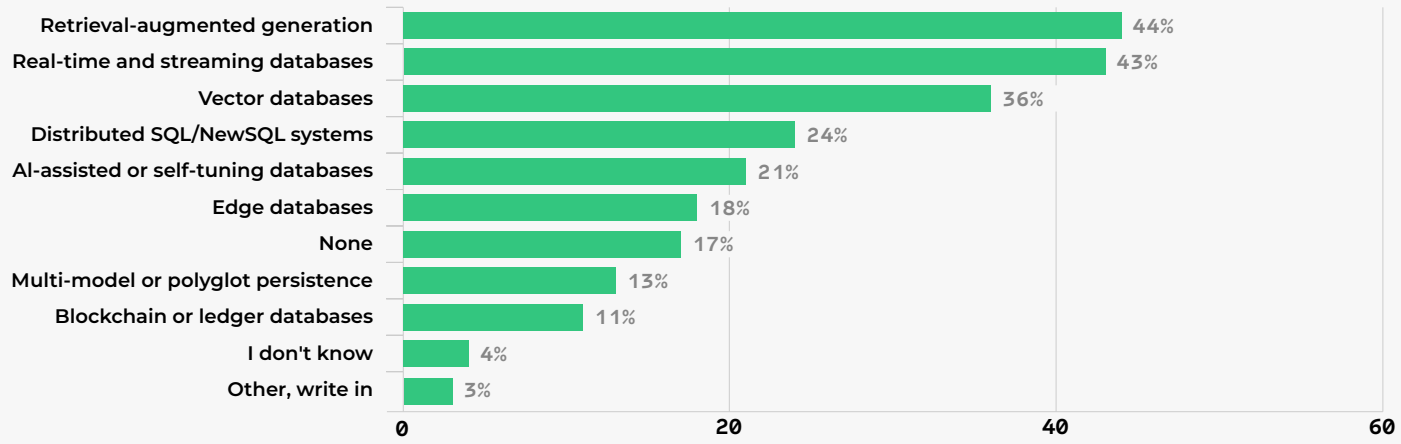
**% of columns; "Other, write-in" row omitted*

New and Emerging Paradigms

We asked the following question: *Which emerging database paradigms is your organization exploring?*

Results:

Figure 17. Emerging database paradigms organizations are exploring [*n*=94]



OBSERVATIONS

Over three-quarters of respondents (78%) said their organization uses at least one of the eight "emerging database paradigms" we listed, and over half (57%) said their organization uses two or more. No single paradigm had a response rate over 50%, but "Retrieval-augmented generation (RAG)" and "Real-time and streaming databases" were the most commonly selected.

Segmenting responses by organization size, we saw the following (see Table 33):

- Respondents at mid-sized organizations were more likely than others to say their organization is looking into "Retrieval-augmented generation (RAG)" and "Vector databases."
- Respondents at small organizations were less likely than others to say their organization is exploring "Real-time and streaming databases" and "Edge databases." They were less likely than others to select "None."

Segmented by data storage type, we found the following (see Table 34):

- Respondents at organizations storing data "In the cloud" were less likely than others to say their organization is exploring "Real-time and streaming databases."
- Respondents at organizations using "Hybrid" cloud data storage were more likely than others to say their organization uses "Retrieval-augmented generation (RAG)" and "Vector databases." They were less likely than others to select "None."
- Respondents at organizations using "Private (on-premises)" cloud data storage were less likely than others to say their organization uses "Retrieval-augmented generation (RAG)" and "Multi-model or polyglot persistence."

CONCLUSIONS

Experimentation with emerging database paradigms is becoming common, though adoption remains varied across a wide range of technologies rather than concentrated in a single trend. The fact that no paradigm exceeded 50% adoption suggests that organizations are still exploring which approaches provide the most tangible value, particularly as AI, real-time processing, and distributed data needs continue to converge. The prominence of retrieval-augmented generation (RAG) and real-time/streaming databases likely reflects the growing intersection of database systems with AI workloads, especially as developers seek faster access to contextual or time-sensitive data.

Mid-sized organizations' higher engagement with RAG and vector databases may signal their role as early adopters in applying AI-native data architectures, possibly due to a balance between agility and available technical resources. In contrast, smaller organizations' lower engagement with real-time and edge databases could stem from limited operational complexity or a lack of immediate use cases that justify the additional infrastructure overhead. Interestingly, the fact that small organizations were also less likely to select "None" suggests that even modest teams are beginning to experiment with at least one new paradigm, albeit at a smaller scale.

Patterns across storage architectures further reinforce the idea that hybrid environments are becoming a testing ground for innovation. Organizations operating across cloud-only and on-premises boundaries appear less likely to explore AI-aligned paradigms like RAG and vector databases, potentially because they often lack the integrated data pipelines and interoperability that hybrid systems foster. Meanwhile, organizations using fully on-premises storage appear slower to experiment with emerging paradigms, perhaps due to longer modernization cycles, integration challenges, or a preference for proven technologies over newer, less standardized approaches.

ADDITIONAL TABLES

Table 33. Emerging database paradigms organizations are exploring by organization size*

Paradigm	Organization Size			Overall
	1-99	100-999	1,000+	
Retrieval-augmented generation (RAG)	41%	58%	41%	44%
Real-time and streaming databases	32%	50%	46%	43%
Vector databases	27%	50%	37%	36%
Distributed SQL/NewSQL systems	23%	23%	29%	24%

CONTINUES ON NEXT PAGE

Paradigm	Organization Size			Overall
	1-99	100-999	1,000+	
AI-assisted or self-tuning databases	23%	23%	22%	21%
Edge databases	5%	19%	27%	18%
Multi-model or polyglot persistence	9%	15%	15%	13%
Blockchain or ledger databases	9%	8%	15%	11%
None	32%	8%	7%	17%
n=	22	26	41	94

**% of columns; "Other, write-in" and "I don't know" rows omitted*

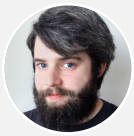
Table 34. Emerging database paradigms organizations are exploring by data storage type*

Paradigm	Storage Type			Overall
	Cloud	Hybrid	Private	
Retrieval-augmented generation (RAG)	40%	59%	27%	44%
Real-time and streaming databases	28%	49%	50%	43%
Vector databases	28%	51%	23%	36%
Distributed SQL/NewSQL systems	24%	31%	19%	24%
AI-assisted or self-tuning databases	20%	26%	15%	21%
Edge databases	12%	18%	27%	18%
Multi-model or polyglot persistence	16%	18%	4%	13%
Blockchain or ledger databases	8%	8%	15%	11%
None	20%	8%	19%	17%
n=	25	39	26	94

**% of columns; "Other, write-in" and "I don't know" rows omitted*

Future Research

Our analysis here only touched the surface of the available data, and we will look to refine and expand our database systems research as we produce future Trend Reports. Please contact publications@dzone.com if you would like to discuss any of our findings or supplementary data.



G. Ryan Spain

@grspain
@grspain
@grspain
gryanspain.com

G. Ryan Spain lives on a beautiful two-acre farm in McCalla, Alabama with his lovely wife and adorable dog. He is a polyglot software engineer with an MFA in poetry, a die-hard Emacs fan and Linux user, a lover of The Legend of Zelda, a journeyman data scientist, and a home cooking enthusiast. When he isn't programming, he can often be found watching Barry with a glass of red wine or a cold beer.

CONTRIBUTOR INSIGHTS

An Analysis of Modern Distributed SQL

By Abhishek Gupta, Product Manager at Microsoft

Distributed SQL merges traditional RDBMS reliability with cloud-native elasticity. The approach combines ACID semantics, SQL interface, and relational integrity with multi-region resilience, disaggregated compute-storage, and adaptive sharding. This article examines distributed SQL from a practitioner's perspective. It evaluates consensus algorithms, partitioning strategies, serverless implementations, vector integration, and cross-region routing techniques.

The State of Consensus

Consensus algorithms form the foundation of distributed SQL reliability guarantees: They ensure a majority of replicas agree on operation order before acknowledging writes. Without consensus, distributed databases cannot commit transactions across nodes, handle leader failures, or maintain consistent data views during network partitions.

Consensus Algorithms

Paxos provides theoretical correctness guarantees, but it is difficult to understand and implement correctly. Multi-Paxos handles sequences of decisions and addresses some practical limitations but is still opaque to most engineers. Raft solves the same problem, with understandability as its explicit design goal. It decomposes consensus into three sub-problems: leader election (selecting one node to coordinate writes), log replication (distributing operations to replicas), and safety (preventing replica divergence).

The majority of modern distributed SQL systems adopt Raft, with only legacy architectures retaining Paxos variants. Raft's leader-based model maps naturally to SQL transactional semantics. A write becomes durable once a majority of replicas acknowledge it, delivering strong consistency without complex coordination protocols.

Operational Complexity vs. Performance Trade-Offs

Consensus creates operational overhead mainly across three areas:

- **Leader elections** – When a leader node becomes unreachable, the cluster elects a replacement. This process spans milliseconds to seconds depending on heartbeat and timeout settings. Writes stall during election windows because no leader exists to coordinate them. This is mitigated by tuning heartbeat intervals and distributing replicas across independent failure domains (racks, zones, regions).
- **Write amplification** – Every write requires acknowledgment from a majority of replicas before commit. A typical three-replica setup generates 2-3x the network traffic and disk I/O of a single-node database. Cross-region deployments multiply this overhead when replicas span continents.
- **Tail latency under contention** – Multiple transactions competing for the same key range force the leader to serialize commits for consistency. This bottlenecks write throughput at the leader's capacity. Adding replicas does not help in this situation. Systems offload reads to follower replicas, but write-heavy workloads with hotspots degrade performance significantly.

Where Consensus Fits and Where It Breaks

Managed consensus services abstract implementation complexity behind cloud APIs and deliver strong resilience with automated failovers. However, this also brings along issues tied to provider architectural decisions: Auto-scaling operations may spike latency unpredictably, misconfigured network policies could render entire regions unwritable, and multi-partition transactions demand additional coordination overhead.

For most workloads, network latency, query planning, and inefficient indexing are far less concerning than consensus overhead. The consensus "cost" is often overestimated without accounting for read scalability and fault tolerance gains. Consensus bottlenecks emerge in specific scenarios such as extreme write throughput demands (tens of thousands of writes per second per range) and latency-sensitive workloads where milliseconds matter. The consensus layer establishes a reliability floor but does not dictate the performance ceiling.

Partitioning and Sharding in the Real World

Consensus determines how distributed SQL systems replicate data safely, and partitioning determines how they distribute it efficiently. Poor partitioning strategies transform horizontal scale into a liability.

Partitioning Strategies and Their Trade-Offs

Serious workloads demand an understanding of partitioning trade-offs. The table below summarizes the core characteristics of each partitioning strategy:

Table 1

Strategy	Primary Strength	Primary Weakness	Best-Fit Workload	Operational Complexity
Hash-based	Uniform distribution eliminates write hotspots	Range scans hit all partitions	Write-heavy with point lookups, key-value access patterns	Low: fixed partition count, predictable behavior
Range-based	Preserves order for efficient range scans	Creates hotspots with skewed data (timestamps, high-value keys)	Time series, analytical queries, sequential access	Medium: requires ongoing monitoring and boundary tuning
Hybrid (range within hash, geo-partitioning)	Combines benefits: locality and distribution	Multiple failure modes, complex mid-migration states	Multi-tenant SaaS, data residency requirements	High: demands deep access pattern understanding

Hash-based partitioning uses hashing functions to distribute rows uniformly across partitions without manual tuning. This trade-off is evident in query patterns. Analytical queries performing range scans (`WHERE wcreated_at > '2024-01-01'`) turn into scatter-gather operations and end up hitting every partition. This makes cross-tenant aggregations and time series analysis inefficient.

Range-based partitioning performs optimally when data distribution aligns naturally with query patterns. This could be time series data partitioned by month or multi-tenant systems partitioned by customer ID. A single high-value customer or recent timestamp range may end up creating hot partitions.

Hybrid schemes succeed when teams thoroughly understand access patterns and possess engineering resources to maintain partition metadata, monitor split/merge operations, and handle failure modes that simpler strategies avoid.

Global Tables, Schema Changes, and Rebalancing

Most distributed SQL systems support global or reference tables: small, read-heavy tables replicated fully to every node to avoid cross-partition joins. Since every update propagates cluster-wide, it could transform a 10 MB table into a 10 GB problem when replicated across 1,000 nodes. Similar issues are associated with schema evolution. Adding columns, creating indexes, or altering constraints becomes a distributed transaction coordinating across all partitions — all this while serving production traffic. This takes hours for large tables, during which queries reconcile multiple schema versions.

Another common concern is rebalancing overhead, a by-product of automatic scaling and sharding. Adding nodes triggers data redistribution, which is competing with production traffic for network, disk, and CPU. When partitions hit size thresholds after traffic spikes, they split, move to new nodes, and trigger further splits as the load redistributes. This can hurt performance as the system spends more time rebalancing than serving queries.

Academic Designs vs. Production Stability

Distributed systems research explores many partitioning schemes such as adaptive partitioning, automatically adjusting boundaries based on access patterns, and learned partitioning, using ML models to predict data distribution. But these schemes often face practical challenges when implemented in production. Adaptive schemes create unpredictable behavior when workloads shift, complicating capacity planning. ML-driven approaches complicate debugging since operators interpret model outputs rather than review configuration files.

Production systems favor predictability. It's easier to reason about hash partitioning with fixed counts, range partitioning with manually reviewed boundaries, and hybrid schemes with explicit geo-pinning. Building debuggable systems that work for real workloads requires upfront schema design and continuous monitoring, as opposed to relying on theoretical claims.

Serverless and Autoscaling Claims

Serverless distributed SQL separates stateless compute (query execution, transaction coordination) from stateful storage (consensus, persistence), allowing compute to scale independently or down to zero without moving data. This separation introduces a performance trade-off where queries cross the compute-storage boundary over the network rather than reading from local storage.

Scaling, Storage Separation, and Cold-Start Realities

Serverless databases balance fast scaling against cost savings. Systems maintaining warm compute pools scale quickly by adding pre-provisioned nodes, while true cold-start provisioning faces significant delays that create unacceptable latency for user-facing applications. Industry implementations converge on warm-start optimizations rather than true zero-capacity scaling.

Most systems keep compute nodes idle but provisioned to reduce start-up latency. Production teams running latency-sensitive workloads configure minimum compute thresholds to maintain always-warm capacity, undermining the cost savings of scaling to zero.

Serverless delivers value for bursty workloads like nightly ETL jobs or end-of-month reporting, where teams pay for compute during active periods rather than running a 24/7 cluster. Always-on workloads with occasional spikes often cost more than right-sized provisioned clusters due to serverless pricing and warm pool overhead. Serverless provides fast scaling for anticipated load but struggles with unanticipated spikes. On the other hand, over-provisioning warm pools reintroduces the fixed costs that serverless was designed to eliminate.

What Serverless Actually Delivers

Serverless distributed SQL delivers value in specific scenarios but faces practical constraints. Systems separating compute from storage scale query layers independently without eliminating operational complexity. The term "serverless" is associated with consumption-based pricing (pay for actual usage), managed operations (abstracted infrastructure), and elastic scaling (dynamic resource adjustment), but implementations vary significantly in resource allocation, scaling speed, and performance isolation.

Scaling operates within capacity boundaries rather than infinitely. Systems maintain resource pools to reduce startup latency. Workloads with predictable patterns and acceptable latency variance benefit most from serverless architectures. Those requiring consistent sub-millisecond performance or sustained high throughput find provisioned clusters more suitable. When evaluating serverless options, examine scaling speed under load, latency penalties during scaling events, throttling behavior under resource pressure, and whether operational simplifications justify the performance trade-offs.

The Vector Era: Indexing for Embeddings

Generative AI has pushed distributed SQL systems to support high-dimensional vector embeddings alongside traditional relational data. SQL engines optimize for exact matches and structured queries, while vector search relies on approximate nearest neighbor (ANN) algorithms that fit unnaturally into relational query planning. This creates performance and integration challenges that teams evaluate against unified data platform convenience.

Distributed SQL systems integrate vector search through extensions like pgvector or native implementations. Common indexing algorithms include Hierarchical Navigable Small World (HNSW) for graph-based approximate search, Inverted File with Product Quantization (IVF-PQ) for clustering-based approaches, and flat indexes for exact search. Distributed query execution scatters vector similarity searches across shards and merges top-*k* results at the coordinator.

Performance Bottlenecks

Vector search in distributed SQL encounters bottlenecks that stem from fundamental mismatches between ANN algorithms and traditional SQL query execution models:

- **Index construction overhead** – Building vector indexes is computationally intensive and competes with production traffic. Distributed environments compound this by fragmenting indexes across partitions, requiring result merging that degrades recall.
- **Query planning limitations** – SQL optimizers lack statistics to efficiently plan queries that combine vector similarity with traditional predicates. Systems struggle to determine optimal execution order, often defaulting to strategies that perform poorly for certain access patterns.

- **Cross-partition execution costs** – Vector queries require scatter-gather operations across all partitions, with distance recalculation at the coordinator. This doubles computational work and scales latency with partition count.

Inside or Beside: The Architectural Debate

Integrated vector support succeeds when consistency and operational simplicity matter more than raw performance, making distributed SQL viable for moderate-scale workloads without adding another system. The separation becomes necessary when scale demands specialized optimizations, similar to how teams use dedicated search engines for full-text queries. Most production deployments adopt a hybrid approach where SQL remains the source of truth while vector databases handle high-throughput similarity searches, trading consistency and operational overhead for performance where it matters most.

Cross-Region Latency and Smart Routing

Multi-region deployments expose fundamental limitations imposed by network latency. Cross-region round-trips add measurable overhead that consensus algorithms and caching strategies cannot eliminate. Mature systems provide explicit controls for balancing consistency, locality, and latency per query, while simpler implementations rely on fixed defaults that work for common cases but lack the flexibility for edge scenarios.

Latency Mitigation Techniques

Three techniques dominate cross-region optimization, each addressing latency through different trade-offs:

- **Follower reads** route queries to local replicas instead of distant leaders, reducing latency at the cost of serving slightly stale data. This performs well for read-heavy workloads like dashboards and analytics, but it requires careful handling for read-modify-write patterns where stale reads cause data inconsistencies.
- **Regional replicas** (geo-partitioning) pin data to specific regions based on locality, keeping queries within a single region fast, while cross-region transactions still face full latency costs. This approach aligns well with data residency requirements but does not eliminate cross-region coordination entirely.
- **Adaptive routing** attempts to optimize query placement dynamically based on current latency and load conditions, but most production systems rely on simpler static routing rules because they offer greater predictability and easier debugging.

Common Production Practices and How To Strike a Balance

Most deployments start single-region, add read replicas for disaster recovery, then enable active-active writes only when necessary. Active-active multi-region is fitting for applications that need global writes. The fundamental challenge is not eliminating cross-region latency but deciding where to accept it.

Systems differ in how they distribute costs between write latency, read consistency, and operational complexity. Single-region leaders keep reads fast through follower replicas while penalizing cross-region writes, whereas multi-region write capabilities reduce regional write latency but add coordination overhead for consistency. Production-ready systems make these trade-offs transparent through documented performance characteristics, explicit configuration options for staleness tolerance, and detailed metrics that cover query routing and replication behavior.

Observability is key to successful deployments. Teams test failover procedures regularly since disaster recovery configurations often fail during actual outages due to DNS propagation delays or misconfigured routing. Cross-region bandwidth costs drive design choices that pricing calculators obscure.

A Rubric for Future-Proofing Distributed SQL

Production-ready implementations require evaluation against multiple criteria beyond ACID compliance and horizontal scalability claims:

- **Observability and operational maturity** – Mature systems expose metrics for consensus health, partition-level query rates, and transaction coordination, and provide snapshot backups with automated failover capabilities.
- **Elasticity and resource sharing** – Scaling capabilities range from manual node addition with slow rebalancing to automatic scale-out. Multi-tenancy provides cost efficiency at the expense of workload isolation; single-tenancy provides isolation at a higher cost.
- **Consistency guarantees** – Strong consistency delivers traditional RDBMS correctness with a latency cost, particularly across regions. Many systems allow per-query configuration with options like follower reads and bounded staleness for workloads that are tolerating slight data lag.

- **Vector support for AI workloads** – Mature implementations provide native vector types and indexing algorithms like HNSW or IVF. Some systems explore ML-driven query planning to optimize execution paths for hybrid vector and relational queries.
- **Community and ecosystem** – Strong ecosystems include wide ranges of client libraries, monitoring tools, and operational documentation beyond vendor materials. Evaluate through third-party conference talks, active community channels, and contributor diversity, not just GitHub star counts.

Guidance for Teams Modernizing From a Monolithic or Legacy RDBMS

Single-node best practices like joins, secondary indexing, and schema flexibility become distributed anti-patterns where cross-partition joins are expensive, indexes multiply write amplification, and schema changes coordinate across hundreds of nodes. The lowest-risk path starts with distributed SQL as a read layer: Keep the monolith authoritative for writes, replicate to a distributed cluster, and route reads there for immediate scalability.

Migrate writes incrementally, starting with partition-friendly workloads. Schema must be partition-aligned early by replacing auto-incrementing IDs with composite keys like `(tenant_id, user_id)` or uniformly distributed UUIDs, and ensuring that frequent queries include partition keys in `WHERE` clauses. Multi-table updates that are trivial in single-node databases become expensive distributed transactions spanning partitions. Identify early whether they can be denormalized, made asynchronous via event-driven architectures, or batched to reduce coordination overhead.

Budget sufficient time for phased migration since moving from monolithic SQL to distributed SQL is more of an architectural transformation than just lift-and-shift.

Conclusion

Distributed SQL has matured from research concepts into production-ready systems. While partitioning schemes and consensus algorithms are established, standards for emerging capabilities still require careful evaluation. Prioritize systems with proven architectures (strong consistency, partition-aligned schemas, predictable behavior) before adopting features that introduce new complexity. Evaluate each against actual requirements rather than marketing claims. The convergence of distributed SQL with AI infrastructure will reshape query optimization and indexing strategies as vector embeddings and traditional relational data increasingly coexist.

Additional resources:

- *Designing Data-Intensive Applications* by Martin Kleppmann
- Jepsen analysis reports – rigorous fault-injection testing exposing consistency gaps
- *Google Site Reliability Engineering* principles
- ANN Benchmarks – comparative analysis of HNSW, IVF, and indexing algorithms
- pgvector documentation
- OpenTelemetry documentation



CORE

Abhishek Gupta

@abhirockzz

@abhirockzz

Abhishek is a product manager at Microsoft. Over the course of his career, he has worn multiple hats, including engineering, consulting, and product management. Most of his work has revolved around open-source technologies, including distributed data systems and cloud-native application development platforms. He is also an open source contributor, speaker, and author.

PARTNER CASE STUDY

SumUp

Why (and How) SumUp Migrated Their Global Payments Platform to CockroachDB

CREATED IN PARTNERSHIP WITH

Cockroach Labs

SumUp
FinTech
3,000+ employees
London, England

Solutions Used
CockroachDB Cloud, AWS Cloud

Primary Outcome
A globally available, strictly consistent, and always-on platform built on CockroachDB

"Our shift to CockroachDB is not just a database choice, it's a choice for a new system that can solve challenges at global scale.

It totally simplifies development and allows us to build truly global payment services across multiple regions."

—**Anton Antonov**, *Engineering Manager, SumUp*

Read the full case study here.

Headquartered in London, SumUp is Europe's leading mobile point-of-sale (POS) company, empowering over 4 million merchants across 36 markets to build thriving businesses. Founded in 2012, SumUp provides a full suite of tools for business owners, from card payments and invoicing to online stores and loyalty programs, enabling them to manage end-to-end operations.

Challenges

As SumUp's global customer base expanded, its legacy database infrastructure, based on AWS RDS for PostgreSQL, began to show serious limitations. The system relied on a single primary write architecture, which created performance bottlenecks and a single point of failure that put them at risk for outages. Regular maintenance and version upgrades also required planned downtime, which became increasingly unacceptable for a platform serving clients across multiple continents and time zones.

Operational complexity grew as the team sought to maintain performance and reliability, and it became clear that the existing setup could not support their ambitions for global scale. SumUp needed a distributed, always-on database that could deliver horizontal scalability and ACID guarantees, was PCI-DSS compliant, could survive AZ outages, and provided seamless multi-region operations without sacrificing developer efficiency.

Solution

After evaluating multiple options, including YugabyteDB and MongoDB, SumUp chose CockroachDB for its ability to provide strict serializability, global consistency, and resilience. CockroachDB's support for native Change Data Capture (CDC) with Debezium, online schema changes, and horizontal scalability gave the team the flexibility and reliability they needed.

SumUp executed a zero-downtime migration strategy, implementing a dual-write system through Kafka and Debezium to synchronize data between PostgreSQL and CockroachDB. This approach ensured continuous operations with no service interruptions or data loss.

Following the migration, CockroachDB has become the backbone of SumUp's payment processing platform. CockroachDB has enabled a self-healing, highly available architecture that simplifies operations, enhances developer productivity, and is future-proof, allowing multi-region and multi-cloud deployment to support regulations like DORA.

Results

By migrating to CockroachDB, SumUp can rely on resilient infrastructure for a truly global payments platform:

- Processing over 1 billion transactions annually
- Handling over 10,000 payments per minute at peak traffic
- Zero data loss and zero downtime during and after migration

CockroachDB

Build without limits with the world's most resilient database

Run your business-critical applications across any environment, whether in the cloud, on-premises, or hybrid, with global scale and unmatched resilience.



Scale elastically with distributed SQL

Say goodbye to sharding and time-consuming manual scaling.



Survive anything with bulletproof resilience

Rest easy knowing your application data is always on and always available.



Build fast with PostgreSQL compatibility

CockroachDB fits how you work, and makes migrating from other relational databases easier.

What can CockroachDB do for you?

cockroachlabs.com/dzone

Trusted by data innovators worldwide

FORTINET

BOSE

Hard Rock DIGITAL

MYTHICAL

COATS

checkout.com

CONTRIBUTOR INSIGHTS

Building Trusted, Performant, and Scalable Databases: A Practitioner's Checklist

Saurabh Dashora, Software Architect at System Design Codex

Modern databases face a fundamental paradox: They have never been more accessible, yet they have never been more vulnerable. Cloud-native architectures, distributed systems, and remote workforces have modified the dynamics of traditional network perimeters, and the usual security approaches have become obsolete. A database sitting behind a firewall is no longer safe. Breaches can increasingly come from compromised credentials, misconfigured APIs, and insider threats rather than external network attacks.

This article provides actionable checklists to help practitioners build databases that are secure, performant, and resilient. We've organized these into two main categories:

- Security and trust
- Performance and reliability

Part 1: Security and Trust

Let us first look at the most important security- and trust-related concerns. We will cover zero-trust data architecture, fine-grained authentication, data masking, and secrets/key management, with key steps for each concern.

1. Zero-Trust Data Architecture

Traditional security models assume that once inside the network perimeter, traffic could be trusted. This assumption is dangerous in modern cloud environments, where attackers can move laterally once they breach a single service. Zero-trust architecture flips this model by assuming that a breach has already occurred and verifying every access attempt regardless of origin.

Here are the key steps to support zero-trust data architecture:

- ☐ Implement network segmentation to isolate database instances
- ☐ Enforce mutual TLS for all database connections
- ☐ Deploy identity-aware proxy layers (e.g., Cloud SQL Auth Proxy, AWS RDS Proxy)
- ☐ Enable audit logging for all database access attempts
- ☐ Use short-lived credentials with automatic rotation
- ☐ Implement IP allowlisting with just-in-time access for administrative operations

For a quick verification test, attempt to connect to your database without proper credentials from an "internal" network segment. It should fail immediately. If it succeeds, the zero-trust implementation has gaps that attackers can exploit.

2. Fine-Grained Authentication and Authorization

Broad database permissions create unnecessary risk. Granting `SELECT` on entire databases when users only need specific tables, or allowing all users to see personally identifiable information (PII) when only certain roles require it, violates the principle of least privilege (PoLP). Row-level security and column-level controls ensure users access only what they absolutely need.

Here are the key steps to implement fine-grained authentication and authorization:

- ☐ Implement role-based access control with the PoLP
- ☐ Configure row-level security policies for multi-tenant applications
- ☐ Apply column-level permissions to restrict sensitive data (PII, financial info)
- ☐ Use attribute-based access control for dynamic authorization
- ☐ Enable multi-factor authentication for administrative access
- ☐ Integrate with enterprise identity providers (e.g., Microsoft Entra ID [Azure AD], Auth0)
- ☐ Regularly audit and review permission assignments

As a quick verification step, log in as different user roles and verify that data visibility matches expected permissions. Query system catalogs to identify overly permissive grants that violate least privilege.

3. Data Masking and Tokenization

Even authorized users don't always need to see raw sensitive data. Developers troubleshooting production issues, analysts running reports, and support staff assisting customers can often do their jobs with masked or tokenized data rather than actual credit card numbers, social security numbers, or personal health information.

Here are the key steps to support this:

- ☐ Implement dynamic data masking for non-production environments
- ☐ Use static data masking for analytics and reporting workloads
- ☐ Deploy tokenization for payment card data (PCI-DSS compliance)
- ☐ Apply format-preserving encryption where applications require specific data formats
- ☐ Create separate masked views for different user tiers
- ☐ Document which fields are masked and their masking rules

As a quick verification step, query sensitive columns as a low-privilege user. Data should appear masked — for example, using `**-****-****-1234` vs. a full credit card number — while maintaining referential integrity so that joins and foreign keys still work correctly.

4. Secrets and Key Management Across Clouds

Hardcoded credentials remain one of the top causes of data breaches. Examples include developers committing database passwords to Git repositories, configuration files containing API keys in plain text, and connection strings sitting in environment variables without encryption. Proper secrets management is the foundation of database security.

Here's a checklist to implement secrets and key management:

- ☐ Never store credentials in application code or configuration files
- ☐ Use dedicated secrets managers (e.g., AWS Secrets Manager, GCP Secret Manager)
- ☐ Enable automatic credential rotation (30–90-day cycles)
- ☐ Implement encryption at rest with customer-managed keys
- ☐ Use envelope encryption for sensitive data fields
- ☐ Store encryption keys separately from encrypted data
- ☐ Enable key usage audit trails
- ☐ Test disaster recovery for key material

To verify, search the entire codebase for connection strings or database passwords using tools like git-secrets. Finding any credentials indicates that immediate remediation is needed. Then, attempt to access your database after rotating credentials without restarting applications. Properly implemented secrets management should allow seamless credential updates.

Part 2: Performance and Reliability Checklists

Let us now look at the most significant concerns around performance and reliability. We will cover monitoring and observability, workload optimization, high availability with fault tolerance, and compliance verification, along with key steps to address each concern.

5. Monitoring and Observability

The key motivation for monitoring comes from the fact that you cannot optimize what you do not measure. Comprehensive observability prevents outages by detecting problems before they cascade into full failures. It also enables proactive optimization by revealing bottlenecks, inefficient queries, and resource constraints before they impact users.

To implement monitoring and observability:

- ☐ Collect the three pillars: metrics, logs, and traces
- ☐ Perform real user monitoring that collects and analyzes data from actual users as they interact with the application
- ☐ Monitor query performance with slow query logs
- ☐ Track connection pool utilization and saturation
- ☐ Set up alerts for abnormal patterns (e.g., sudden CPU spikes, connection storms)
- ☐ Implement distributed tracing for multi-service transactions
- ☐ Monitor replication lag in real time
- ☐ Track disk I/O, IOPS, and storage growth trends
- ☐ Use database-specific tools (e.g., pg_stat_statements, MySQL Performance Schema)
- ☐ Integrate with APM platforms (e.g., Prometheus, Grafana)

To verify, intentionally run an expensive query (a full table scan on a large table) and verify that it appears in your monitoring dashboards within 60 seconds. Then, simulate a replica failure by stopping a secondary database instance and confirm that alerting fires within your target detection window.

6. Workload Optimization

Most database performance issues come from poorly optimized queries and missing indexes. A single unindexed column in a `WHERE` clause can transform a sub-millisecond query into a multi-second full table scan. N+1 query patterns, where applications execute hundreds of queries in loops instead of using joins, can bring databases to their knees under load.

To optimize workloads:

- ☐ Analyze query execution plans for full table scans
- ☐ Create indexes on frequently filtered and joined columns
- ☐ Implement query result caching (Redis, Memcached)
- ☐ Use read replicas to offload analytical queries
- ☐ Configure connection pooling
- ☐ Partition large tables by date or key range
- ☐ Archive historical data to separate cold storage
- ☐ Implement prepared statements to reduce parsing overhead
- ☐ Review and optimize N+1 query patterns

To verify, run `EXPLAIN ANALYZE` on your top 10 slowest queries (identified from slow query logs) to identify missing indexes or suboptimal execution plans. Look for `Seq Scan` operations on large tables — these are prime candidates for index creation.

7. High Availability and Fault Tolerance

Downtime is expensive both financially and reputationally. Modern databases must survive hardware failures, network partitions, and even entire data center outages without losing data or experiencing extended unavailability. High availability requires redundancy, automated failover, and tested recovery procedures.

To support high availability with fault tolerance:

- ☐ Deploy multi-AZ or multi-region replicas
- ☐ Implement automatic failover with health checks
- ☐ Test failover procedures quarterly using techniques like chaos engineering
- ☐ Configure backups with point-in-time recovery
- ☐ Store backups in separate regions/accounts
- ☐ Verify backup restoration regularly
- ☐ Set up circuit breakers to prevent cascading failures
- ☐ Implement graceful degradation for read-only modes
- ☐ Document and practice runbooks for common failure scenarios
- ☐ Use consensus-based replication for strong consistency requirements

As a verification step, simulate primary database failure by stopping or network-isolating the primary instance, then measure time to recovery, including detection, failover, and application reconnection. The target should align with the recovery time objective. Additionally, restore from backup to a separate instance and verify data integrity through checksums or row counts.

8. Compliance Verification Tests

Beyond implementation checklists, organizations need regular testing to verify that controls remain effective over time. Permissions creep, configuration drift, and forgotten test accounts can undermine even the most well-designed security architectures. Schedule these tests at appropriate intervals based on your compliance requirements and risk tolerance.

As part of these tests, take care of the following points:

- ☐ Audit user permissions and remove unused accounts
- ☐ Review audit logs for anomalous access patterns
- ☐ Verify backup completion and test restoration
- ☐ Check for unencrypted data at rest
- ☐ Conduct penetration testing on the database layer
- ☐ Review and update security policies
- ☐ Test disaster recovery procedures end to end
- ☐ Validate compliance with GDPR/HIPAA/SOC 2 requirements
- ☐ Commission a third-party security audit
- ☐ Execute a full-scale disaster recovery test
- ☐ Review and update incident response playbooks

As a verification step, simulate primary database failure by stopping or network-isolating the primary instance, then measure time to recovery, including detection, failover, and application reconnection. The target should align with the recovery time objective. Additionally, restore from backup to a separate instance and verify data integrity through checksums or row counts.

Conclusion

Building trusted, performant, and scalable databases requires continuous vigilance across security and operational domains. The checklists provided in this article aren't one-time exercises but represent ongoing practices that should mature alongside the organization's database infrastructure. By systematically working through these checklists and verification tests, organizations can build database infrastructures that are not only secure and compliant but also performant and resilient enough to support business-critical applications at scale.



CORE

Saurabh Dashora

@saurabh.dashora
@saurabh-dashora
systemdesigncodex.com

I'm a full-stack architect, tech writer, and guest author in various publications. I have expertise in building distributed systems across multiple business domains such as banking, autonomous driving, and retail. Throughout my career, I have worked at several large organizations. I also run a tech blog on cloud, microservices, and web development, where I have written hundreds of articles. Apart from work, I enjoy reading books and playing video games.

ADDITIONAL RESOURCES

Solutions Directory



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

DZONE'S 2025 DATABASE SYSTEMS SOLUTIONS DIRECTORY

2025 PARTNER	Product	Purpose	Availability	Website
	Cockroach Labs CockroachDB	Built for what's next: scale, resilience, and Postgres ease	Free tier	cockroachlabs.com/product/overview

Product	Purpose	Availability	Website
4D	Integrated development for database-driven apps	Trial period	us.4d.com
Actian Ingres	SQL-standard transactional database	By request	actian.com/databases/ingres
Actian NoSQL FastObjects	Transactional object management	By request	actian.com/databases/nosql
Actian Vector	Scalable, secure, and flexible analytics database	By request	actian.com/analytic-database/vector-analytic-database
Actian Zen	Edge-to-cloud data management	By request	actian.com/databases/zen
Aerospike Cloud	Fully managed DBaaS	Trial period	aerospike.com/products/aerospike-cloud
Aerospike Database 8	Real-time, scalable NoSQL database	Free tier	aerospike.com/products/database
Aiven for Apache Cassandra	Fully managed, distributed NoSQL database	Free tier	aiven.io/cassandra
Aiven for MySQL	Fully hosted and managed MySQL	Free tier	aiven.io/mysql
Aiven for PostgreSQL	Fully managed and hosted PostgreSQL	Free tier	aiven.io/postgresql
Altair Graph Studio	Graph toolset for data integration, transformation, and analytics	By request	altair.com/altair-graph-studio
Altibase Enterprise DB	Hybrid DBMS	Free tier	altibase.com/en
Amazon Aurora	RDBMS compatible with MySQL and PostgreSQL	By request	aws.amazon.com/rds/aurora
Amazon DynamoDB	Serverless, NoSQL, fully managed database	Trial period	aws.amazon.com/dynamodb
Amazon RDS	Relational database service	Trial period	aws.amazon.com/rds
Amazon Timestream for InfluxDB	Scalable and serverless time series database	By request	aws.amazon.com/timestream
Apache Cassandra	NoSQL distributed database	Open source	cassandra.apache.org/_/index.html
Apache Druid	Real-time analytics database	Open source	druid.apache.org
Apache HBase	Hadoop database; distributed, scalable big data store	Open source	hbase.apache.org
Apache Ignite	Distributed database with in-memory speed	Open source	ignite.apache.org

DZONE'S 2025 DATABASE SYSTEMS SOLUTIONS DIRECTORY			
Product	**Purpose**	**Availability**	**Website**
Apache Kafka	Distributed event streaming platform	Open source	kafka.apache.org
ArangoDB Community Edition	Graph database	Free	arangodb.com/community-server
ArangoGraph	Managed graph database	Trial period	arangodb.com/arangograph-managedgraphdb
Arcitecta Mediaflux	Data management platform	By request	arcitecta.com/mediaflux/features
BMC AMI Database Administration for Db2	Task automation for DBAs	By request	bmc.com/it-solutions/bmc-ami-database-administration-db2.html
BMC AMI Database Performance for Db2	Optimize Db2 data for performance and cost	By request	bmc.com/it-solutions/bmc-ami-database-performance-db2.html
CakeDC CakePHP	PHP framework for web app development	Open source	cakedc.com/cakephp
Cloudera Data Platform	Hybrid data platform	Trial period	cloudera.com/products/cloudera-data-platform.html
Couchbase Capella	Cloud database	Trial period	couchbase.com/products/capella
CUBRID	DBMS optimized for OLTP	Open source	cubrid.org
Dassault Systèmes NuoDB	Distributed SQL database	By request	3ds.com/nuodb-distributed-sql-database
DBmaestro Database DevOps Platform	Automate, secure, and govern database CI/CD pipelines	Trial period	dbmaestro.com
DbVisualizer	SQL client and database management	Free tier	dbvis.com
Devart dbForge Studio for MySQL	MySQL/MariaDB GUI tool for DB development, management, and admin	Trial period	devart.com/dbforge/mysql/studio/main-overview.html
Devart dbForge Studio for Oracle	Oracle management GUI and IDE tools	Trial period	devart.com/dbforge/oracle/studio
Devart dbForge Studio for PostgreSQL	PostgreSQL GUI client and SQL management	Trial period	devart.com/dbforge/postgresql/studio
Devart dbForge Studio for SQL Server	SQL server manager GUI tool and IDE	Trial period	devart.com/dbforge/sql/studio
DigitalOcean Managed Kafka	Fully managed Apache Kafka	By request	digitalocean.com/products/managed-databases-kafka
DigitalOcean Managed MongoDB	Fully managed MongoDB clusters on demand	By request	digitalocean.com/products/managed-databases-mongodb
DigitalOcean Managed MySQL	Fully managed MySQL	By request	digitalocean.com/products/managed-databases-mysql
DigitalOcean Managed PostgreSQL	Fully managed PostgreSQL	By request	digitalocean.com/products/managed-databases-postgresql
Dragonfly Cloud	Fully managed in-memory data store	Trial period	dragonflydb.io/cloud
Dragonfly Community Edition	In-memory data store	Free	github.com/dragonflydb/dragonfly
EclipseLink	Java-based ORM framework	Open source	eclipse.dev/eclipselink
EDB Postgres AI Cloud Service	Fully managed Postgres Cloud database	Trial period	enterprisedb.com/products/edb-postgres-ai-cloud-service
Embarcadero InterBase	Scalable and embeddable SQL database	Free tier	embarcadero.com/products/interbase
Exasol Data Warehouse	On-premises data warehouse	By request	exasol.com/products/data-warehouse
FairCom DB	JSON database	Free tier	faircom.com/products/db

DZONE'S 2025 DATABASE SYSTEMS SOLUTIONS DIRECTORY			
Product	**Purpose**	**Availability**	**Website**
Firebird	SQL RDBMS	Open source	firebirdsql.org
Gel	Scalable, integrated data platform on top of Postgres	Free	github.com/geldata/gel
Gel Cloud	Fully managed cloud database service	Free tier	geldata.com
Google Cloud Bigtable	Low-latency NoSQL database service	Trial period	cloud.google.com/bigtable
Google Cloud Firestore	NoSQL document database	Trial period	cloud.google.com/firestore
Google Cloud Spanner	Distributed SQL database management and storage service	Trial period	cloud.google.com/spanner
Graphite	Time series data monitoring tool	Open source	graphiteapp.org
GridGain Nebula	Cloud-native management and monitoring for Apache Ignite	Trial period	gridgain.com/products/gridgain-nebula
IBM DataStax Astra DB	NoSQL and vector database	Free tier	ibm.com/products/datastax
IBM Db2 Database	Cloud-native database	Free tier	ibm.com/products/db2-database
IBM Informix	Embeddable database for transactional workloads	Free tier	ibm.com/products/informix
Imply Polaris	Fully managed DBaaS for Apache Druid	Trial period	imply.io/imply-fully-managed-dbaas-polaris
InfluxData Telegraf	Data collection agent	Open source	influxdata.com/telegraf
InfluxDB Cloud	Time series data platform	Free tier	influxdata.com/products/influxdb-cloud
InterSystems IRIS	DBMS	Trial period	intersystems.com/products/intersystems-iris
Jetbrains DataGrip	Cross-platform tool for relational and NoSQL databases	Free tier	jetbrains.com/datagrip
jOOQ	Java library for type-safe SQL query contruction and excecution	Free tier	jooq.org
Kinetica	Real-time GPU database	Free tier	kinetica.com
Liquibase	Database change management and CI/CD automation	Open source	liquibase.com
Machbase	Time series database engine	By request	machbase.com
Mammoth	Data management platform	Trial period	mammoth.io
MariaDB Community Server	Relational database	Open source	mariadb.com/products/community-server
MariaDB Enterprise Server	Enhanced, hardened, and secured relational database	By request	mariadb.com/products/enterprise
Microsoft Azure Cosmos DB	NoSQL and relational database	Trial period	azure.microsoft.com/en-us/products/cosmos-db
Microsoft Azure Database for MySQL	Fully managed, scalable MySQL database	Trial period	azure.microsoft.com/en-us/products/mysql
Microsoft Azure SQL Database	Managed cloud database service	Trial period	azure.microsoft.com/en-us/products/azure-sql/database
MongoDB Atlas	Multi-cloud developer data platform	Trial period	mongodb.com/atlas
MyBatis	SQL mapping framework for Java	Open source	blog.mybatis.org
MySQL	RDBMS for web apps	Open source	mysql.com
Navicat for MySQL	MySQL/MariaDB administration and development	Trial period	navicat.com/en/products/navicat-for-mysql

DZONE'S 2025 DATABASE SYSTEMS SOLUTIONS DIRECTORY

Product	Purpose	Availability	Website
Navicat for Oracle	Oracle administration and development	Trial period	navicat.com/en/products/navicat-for-oracle
Navicat for PostgreSQL	Graphical tool for PostgreSQL database development	Trial period	navicat.com/en/products/navicat-for-postgresql
Navicat for SQL Server	Graphical tool for database management and development	Trial period	navicat.com/en/products/navicat-for-sqlserver
Navicat Premium	Connect to multiple databases	Trial period	navicat.com/en/products/navicat-premium
Neo4j AuraDB	Fully managed graph database	Free tier	neo4j.com/cloud/platform/aura-graph-database
OpenText Gupta SQLBase	Embedded database for SMBs	Trial period	opentext.com/products/gupta-sqlbase
OpenText Vertica Data Platform	Analytics database	By request	opentext.com/products/vertica-data-platform
Oracle Autonomous JSON Database	Cloud document database service	Free tier	oracle.com/autonomous-database/autonomous-json-database
Oracle HeatWave MySQL	MySQL cloud service	Free tier	oracle.com/mysql
Oracle NoSQL Database Cloud Service	Fully managed database service	Free tier	oracle.com/database/nosql
OrientDB	Multi-model NoSQL DBMS	Open source	orientdb.org
Percona Monitoring and Management	Monitoring, observability, and management tool	Open source	percona.com/software/database-tools/percona-monitoring-and-management
pgEdge Cloud	Fully managed distributed PostgreSQL	Trial period	pgedge.com/products/pgedge-cloud
pgEdge Platform	Self-managed distributed PostgreSQL	Trial period	pgedge.com/products/self-managed-vm
Pinecone	Serverless vector database	Free tier	pinecone.io
Portworx Data Services	Data service management	By request	portworx.com/services/database
PostgreSQL	Relational database	Open source	postgresql.org
Progress MarkLogic Data Platform	Unified data platform	By request	progress.com/marklogic/data-platform
Progress MarkLogic Server	NoSQL DBMS	By request	progress.com/marklogic/server
QuestDB	High performance time series database	Free tier	questdb.io
RavenDB	NoSQL document database	Free tier	ravendb.net
Red Gate Flyway	Database deployment automation	Free tier	red-gate.com/products/flyway
Red Gate SQL Toolbelt Essentials	Tools for SQL Server development and deployment	By request	red-gate.com/products/sql-toolbelt-essentials
Red Hat Hibernate ORM	Java framework for ORM and database interaction	Open source	hibernate.org/orm
Redis Community Edition	In-memory database	Open source	redis.io/docs/latest/get-started
Riak KV	Distributed NoSQL key-value database	Free tier	riak.com/products/riak-kv
Riak TS	NoSQL time series database	Free tier	riak.com/products/riak-ts
SAP HANA Cloud	Multi-model DBMS	Trial period	sap.com/products/technology-platform/hana.html
ScaleOut In-Memory Database	In-memory database for real-time analytics and data processing	By request	scaleoutsoftware.com/products/in-memory-database

DZONE'S 2025 DATABASE SYSTEMS SOLUTIONS DIRECTORY

Product	Purpose	Availability	Website
ScyllaDB	NoSQL DBaaS	Free tier	scylladb.com
SingleStore Helios	Cloud DBaaS	Free tier	singlestore.com/product-overview
SkySQL	Fully managed MariaDB	Trial period	skysql.com
Solarwinds Database Performance Analyzer	Monitoring and optimization for multiple DBMS platforms	Trial period	solarwinds.com/database-performance-analyzer
Solarwinds Database Performance Monitor	Monitoring and optimization for databases	Trial period	solarwinds.com/database-performance-monitor
SQLite	C-language library that implements a SQL database engine	Open source	sqlite.org
StarTree Cloud	Real-time analytics platform	Free tier	startree.ai/products/startree-cloud
Studio 3T	GUI tools for MongoDB	Free tier	studio3t.com
SymmetricDS	Database replication software	Open source	symmetricds.org
Teradata VantageCloud	Cloud analytics and data platform	By request	teradata.com/platform/vantagecloud
TigerGraph Savanna	Fully distributed cloud-based graph database	Trial period	tigergraph.com/savanna
TigerGraphDB	Native parallel graph database	Trial period	tigergraph.com/tigergraph-db
Tiger Data TimescaleDB	Time series database	Open source	tigerdata.com/timescaledb
VMware Tanzu for Postgres	Enterprise-ready PostgresSQL	By request	tanzu.vmware.com/postgres
Volt Active Data	Data platform	By request	voltactivedata.com
Windocks	On-demand database environments	Free tier	windocks.com
YugabyteDB	Distributed SQL database for cloud-native transactional apps	Open source	yugabyte.com/yugabytedb

DZone

DATA ENGINEERING

AI/ML | Databases | Big Data Data | Internet of Things

As you determine the first steps for new systems or reevaluate existing ones, you're going to require tools and resources to gather, store, and analyze data.

The Zones within our Data Engineering category contain resources that will help you expertly navigate through the SDLC Analysis stage.

VISIT THE ZONE



TUTORIALS



TECHNIQUES



CODE SNIPPETS



RESEARCH

DZone

3343 Perimeter Hill Dr, Suite 100
Nashville, TN 37211
888.678.0399 | 919.678.0300

At DZone, we foster a collaborative environment that empowers developers and tech professionals to share knowledge, build skills, and solve problems through content, code, and community. We thoughtfully — and with intention — challenge the status quo and value diverse perspectives so that, as one, we can inspire positive change through technology.

Copyright © 2025 DZone. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means of electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.