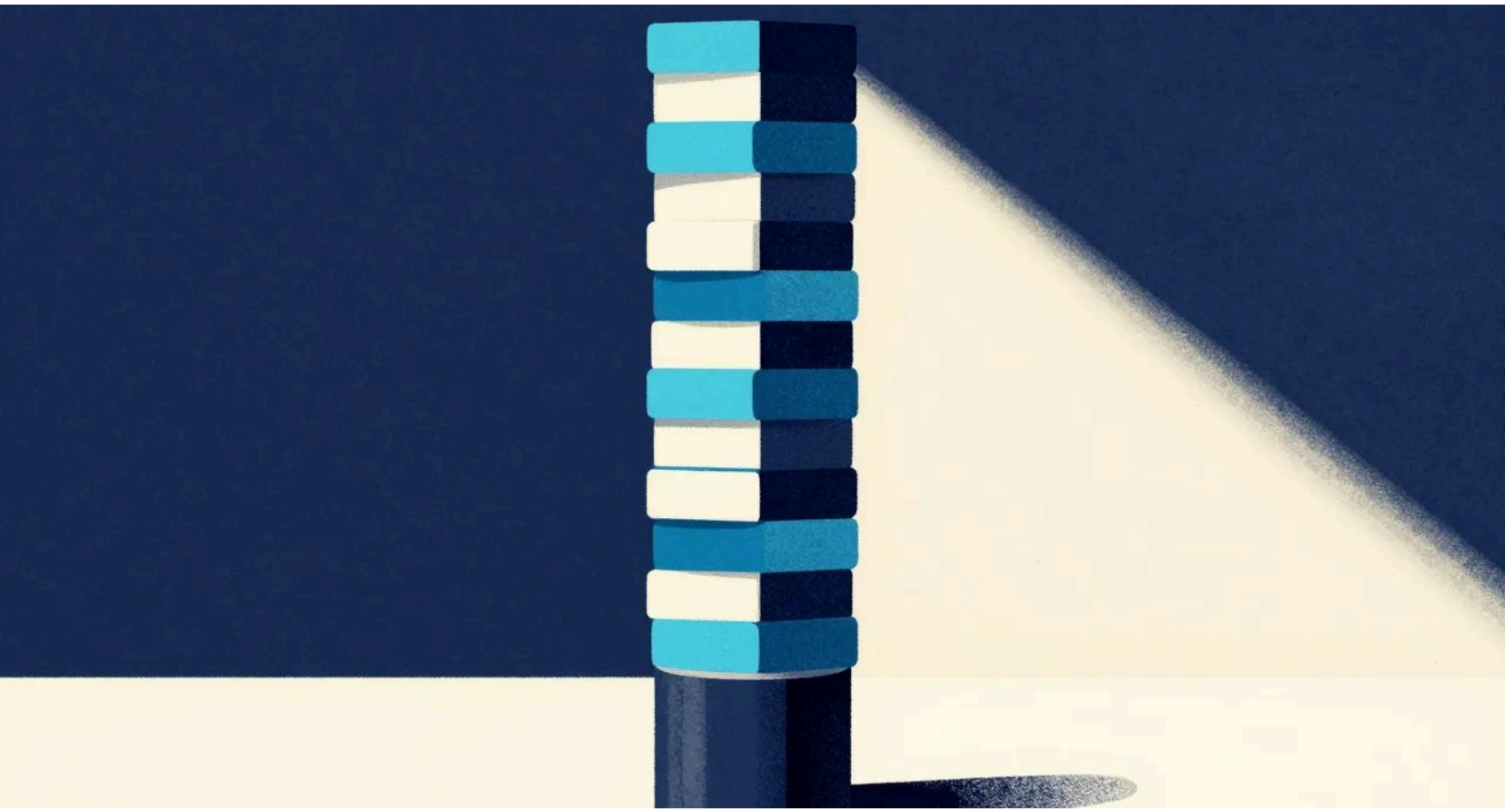


EXECUTIVE PAPER · INFRASTRUCTURE

The Database Is Your Real Single Point of Failure

Redundant servers, multiple zones and a cloud contract still sit on top of one database that every order, invoice and login depends on. Executives should give that tier a named owner and a funded plan.

AUTHOR

Chris Watkins

SERVICE

Infrastructure

FOR

CTO, CEO, CFO, COO · 12 min read

AT A GLANCE

The problem. Application servers are easy to duplicate. The database behind them usually is not, and when it slows, fails over badly or receives a bad schema change, every service that reads from it fails together. Recent public postmortems from GitHub, Slack, Matrix.org and Clerk trace hours of customer-facing outage to the database tier, and about half of operators in Uptime Institute's 2026 analysis name configuration and change management among the main causes of their major software outages.

Why it persists. Databases sit between teams. Developers change the schema, infrastructure runs the servers, a cloud provider runs the engine, and nobody owns the whole. Version deadlines arrive as a surprise bill, and failover is assumed instead of rehearsed.

The recommendation. Name an accountable owner for each database that carries revenue or regulated data. Fund five things for each one: failover that is tested on a schedule, a version runway with dates, capacity and slow-query alerts, a written schema-change discipline, and a yearly decision on whether to run it yourself or buy it managed.

The Decision in Front of You

Most leadership teams have paid for redundancy at the edges: paired web servers, multiple cloud zones, two network carriers. Every request still passes through one component, the database holding customers, orders, inventory and money.

The decision is whether that tier gets an owner and a budget line of its own, or stays a shared responsibility spread across developers, infrastructure staff and a cloud provider. The shared arrangement is the common default, and it sits behind each incident in this paper.

The stakes are measurable. In Uptime Institute's 2025 survey of more than 800 data center operators, 57% said their most recent major outage cost more than $100,000, and one in five put it above $1 million for the second year running [1]. A database outage is rarely contained to one product, because the database is what the products share.

What's Happening

Most outages now start in software and change

Uptime's 2026 annual outage analysis asked operators what caused their major IT system and software outages over the past three years. Firmware or software faults were named by 53%, and configuration or change management issues by 51% [1]. Capacity and congestion problems were named by 19%, and

data synchronization or corruption by 15% [1]. Those categories describe the daily work of running a database.

People are part of nearly every one of these failures. In the same 2026 survey, 92% of respondents said human error contributed at least in a minor way to their most recent impactful outage, and 31% called it a major contributor [1]. Uptime's 2025 report found that of the organizations hit by a major human-error outage, 85% traced it to staff not following procedures or to flawed procedures [2]. A database change made by hand, at an unplanned hour, without a written sequence, is the textbook setting for both.

The people who know databases are becoming scarcer inside companies

The US Bureau of Labor Statistics counted about 75,000 database administrators and 69,500 database architects in 2025, and projects administrator employment to stay roughly flat through 2035 while architect roles grow 9% [3]. BLS attributes the flat outlook to cloud services, which let fewer administrators serve more companies [3]. In many mid-size firms the full-time specialist has been replaced by a managed service and a developer who knows enough SQL, while the operational risk stays with the company. Where that knowledge sits with one person, see the companion paper on key-person risk.

Version deadlines arrive on fixed dates

Exhibit 1. Database support dates are fixed years in advance and are often close

Engine and version	Support status	Date
MySQL 8.0	End of life; no new security fixes	April 30, 2026 [4]
SQL Server 2016	End of support; paid Extended Security Updates only	July 14, 2026 [5]
PostgreSQL 14	Final community release	November 12, 2026 [6]
SQL Server 2017	End of extended support	October 12, 2027 [7]
PostgreSQL 15	Final community release	November 11, 2027 [6]
Oracle Database 19c	End of Premier Support; Extended Support to December 31, 2032	December 31, 2029 [8]

Sources: PostgreSQL Global Development Group [6], Microsoft [5][7], coverage of Oracle's lifecycle announcements [4][8].

PostgreSQL supports each major version for five years from its first release [6]. MySQL 8.0 left support at the end of April 2026, and data from Percona's monitoring tool, reported in October 2025, showed 58% of the MySQL and MariaDB instances it watched still on 8.0 and 18.8% on 5.7, which had already lost support in 2023 [4]. Managed services attach a price to the deadline in the form of extended support fees [9]. The end-of-life paper makes the general case; for databases the date is known years ahead, so the upgrade can be funded before the bill or the exposure starts.

Four recent cases

GitHub, August 2025. On August 5, a production migration dropped a column from a table behind pull requests. The column was no longer used directly, but the application's object-relational mapping layer still referenced it in some queries [10]. Errors peaked at about 4% of all web and API traffic for 32 minutes [10]. On August 27 a second column drop, on a table behind Copilot, produced the same failure: Copilot requests failed at an average of 36% and a peak of 77% for 46 minutes, and about 2% of other web and API requests failed [10]. GitHub then blocked all drop-column operations while it built longer-term safeguards [10]. In April, contention during a schema migration had already degraded several services for 20 minutes [12]. A schema change is a production release, and removal needs its own gate.

Slack, February 2025. Slack traced a nine-and-a-half-hour disruption on February 26 to a maintenance action in one of its database systems, which combined with a latent defect in its caching layer to send heavy traffic to the database [13]. About half of the instances that depended on that database became unavailable. Engineers paused a job queue to shed load, and events held during the pause were never replayed, which broke customers' integrations a second time [13]. Planned database maintenance met an untested interaction with the cache, and every service behind that database felt it.

Matrix.org, September 2025. The matrix.org service runs on one PostgreSQL cluster of two bare-metal servers holding about 51 TB, at roughly 90% of disk capacity [14]. During a disk expansion one disk dropped out of the primary's array, so the team failed over to the secondary. The new primary was not archiving its write-ahead log to backup storage because of a script issue. While cleaning up the old primary, an operator deleted the data directory of the live primary, which took both servers out [14]. The service was down for about 24 hours while 51 TB was restored and replayed; no data was lost [14]. Every step was routine. Disk was nearly full, log archiving on the standby had not been verified, and server names implied roles that had changed [14]. The restore lessons belong to the untested-backups paper.

Clerk, March 2026. The authentication provider Clerk runs on Google Cloud SQL, a managed PostgreSQL service. On March 10, a failed live migration of the database's underlying virtual machine raised disk latency, lock contention spread, and Clerk's APIs failed for about 26 minutes [15]. Clerk said a different live migration had caused a major incident in September 2025, and that it would use replica promotion for future maintenance and consider another provider or self-hosting if live migrations could not be disabled [15]. A managed database moves routine work to the provider, and the provider's maintenance schedule becomes one of your outage causes.

Cloudflare's November 2025 outage also began with a database permissions change [16]; the monitoring paper covers it.

Why the Usual Approach Fails

The usual approach treats the database as part of something else. Developers change it with their code, infrastructure teams run it as a server, and the cloud contract treats it as a service. Nobody holds the failover plan, version calendar, capacity forecast and change rules together.

Failover is the clearest example. Most production databases have a replica, and the architecture diagram shows a line to it, as Matrix.org's did [14]. Whether the replica is current, whether the promoted server archives its logs, and whether applications reconnect on their own are only known after a rehearsal on production.

Schema changes look safe for a similar reason. GitHub's dropped columns were unused by the code people read, while generated code still referenced them [10]. The established method is parallel change, also called expand and contract: add the new structure, move every caller to it, confirm nothing uses the old one, and only then remove it [17].

Capacity and query load fail slowly. A database at 90% of disk has no room for a re-sync or a restore beside the live copy [14]. GitHub's January 2025 disruption came from one new query saturating a primary for 30 minutes, and its follow-up was tooling to catch problem queries before deployment [11].

The Framework: The Database Charter

Every database that carries revenue or regulated data gets a one-page charter, signed by an accountable owner and reviewed each year. The charter covers five commitments.

Exhibit 2. The Database Charter: five commitments per critical database

Commitment	What it requires	Evidence the owner keeps
Named owner	One accountable person for availability, versions and changes, with authority to block a release	Signed charter; on-call rotation with database skills
Tested failover	Planned production switch to the replica twice a year, timed end to end, including application reconnection and log archiving	Dated test record with measured switch time and data gap
Version runway	Engine version, end-of-support date, and an upgrade funded 12 months ahead	Version register reviewed by finance and engineering
Capacity and query watch	Alerts on storage headroom, connection saturation, replication lag and slow queries, each routed to a person	Thresholds on file; monthly trend of top slow queries
Change discipline	Schema changes reviewed as releases; expand and contract for destructive steps; drops wait after usage reaches zero	Change log linking each migration to its review and rollback step

Source: author's framework, informed by the incident reports [10][11][13][14][15], Uptime Institute cause data [1], and the parallel change pattern [17].

The owner makes the rest happen. In each case above, the gap existed before the incident, and nobody's job was to ask whether the new primary would archive logs or whether any code still touched a column.

Failover tests switch production on purpose, in an announced window, with the time and data gap measured. Only a production switch shows the true state of the live replica. The version runway puts each engine's support date in the register finance already uses for contract renewals, so the choice between upgrading and paying for extended support is made a year early [5][9].

What a slipped runway costs

In the author's illustrative estimate, ten managed clusters on an out-of-support version, each with an 8-vCPU primary and an 8-vCPU standby, give 160 billable vCPUs. At AWS's example rates of $0.10 per vCPU-hour in years one and two and $0.20 in year three [9], running around the clock, the charge is $140,160 in each of the first two years and $280,320 in the third, or $560,640 over three years. It assumes constant fleet size, no read replicas and US West pricing.

Costs, Risks, and Tradeoffs

The most common structural choice is whether to run the database yourself or buy it as a managed service. Neither choice removes the need for an owner.

Exhibit 3. Managed database service versus self-run database

Factor	Managed service (e.g. RDS, Cloud SQL, Azure SQL)	Self-run (your servers or cloud VMs)
Failover	Built in and automated; you test whether your application survives it	You build, script and rehearse it yourself
Maintenance timing	Provider schedules patches and host moves; windows can be set but not always avoided [15]	You choose every window
Version deadlines	Provider forces upgrades or charges extended support per vCPU [9]	You decide, and carry the security exposure after end of life [4][5]
Skills required	Fewer specialists for routine work; still need tuning, schema and capacity skills [3]	A database specialist on call, plus a backup person
Visibility	Limited access to the host and engine internals during an incident	Full access to logs, disks and processes [14]
Cost profile	Higher unit price; less staff time	Lower unit price; more staff time and tooling
When this is wrong	A managed service is the wrong fit when the provider's maintenance model has already caused repeat outages you cannot prevent, or when the workload needs engine access the service does not allow. Self-running is the wrong fit when the company cannot staff a 24-hour rotation with two people who can promote a replica and restore from logs.	

Source: author's analysis, with provider behavior from the cited postmortems and pricing documentation.

The charter has costs of its own. A production failover test can cause the outage it is meant to prevent, so it needs an announced window and a rollback plan. Expand and contract stretches schema changes from minutes to days, and an owner who can block releases will sometimes slow product work. A company whose data lives entirely in SaaS products should spend this effort on vendor contracts and exit plans instead.

How to Act

Exhibit 4. A one-year roadmap

Window	Actions	Owner	Milestone
First 30 days	List every database that carries revenue, customer or regulated data. Record engine, version, end-of-support date, size, free disk and replica status. Name an owner for each.	CTO	Register complete; every critical database has a signed owner
First 30 days	Put each end-of-support date and any extended support charge into the budget calendar.	CFO with CTO	Upgrade quarter funded for every version ending within 18 months
First 90 days	Add alerts for storage headroom, connections, replication lag and slow queries, routed to a named on-call person.	Database owners	Every critical database pages a person on all four signals
First 90 days	Adopt a written schema-change standard: review as a release, expand and contract for destructive steps, waiting period before drops.	Engineering leads	All migrations in the quarter logged against the standard
First 90 days	Run the first planned production failover on the most critical database.	Database owner with operations lead	Measured switch time and data gap recorded; defects fixed
First 365 days	Run failover twice a year on every critical database. Complete overdue version upgrades. Review the managed versus self-run choice for each.	CTO with COO	Zero critical databases past end of support; two passed failover tests each

Source: author's recommendation.

Questions to Ask Your Team

1. Which databases would stop orders, billing or customer logins if they went down, and who owns each one by name?
2. When did we last switch production to the replica on purpose, how long did it take, and how much data was missing afterward?
3. Which of our database engines reach end of support in the next 18 months, and is the upgrade funded?
4. Are we paying extended support charges on any managed database today, and how much per year?
5. How much free disk does our largest database have compared with what a full re-sync or restore needs?

6. Who reviews a schema change before it reaches production, and can a column or table be dropped without that review?
7. When our cloud provider performs maintenance on our database, how much notice do we get, and what happened the last time?

Closing

Redundant web servers do not help when the database they share is the thing that fails. Give each critical database an owner with authority, rehearse the failover, put the version dates in the budget, watch capacity and slow queries, and treat schema changes as releases. The plan costs staff time and some upgrade spend each year, and it puts the company's most shared dependency under the same ownership and review as any other asset that carries revenue.

Sources

[1] Uptime Institute Intelligence. "Annual Outage Analysis 2026: The causes and impacts of IT and data center outages." Keynote Report 201. May 2026. https://datacenter.uptimeinstitute.com/rs/711-RIA-145/images/2026.AnnualOutageAnalysis.pdf.

[2] Uptime Institute. "Uptime Announces Annual Outage Analysis Report 2025." Press release. May 6, 2025. https://uptimeinstitute.com/about-ui/press-releases/uptime-announces-annual-outage-analysis-report-2025.

[3] U.S. Bureau of Labor Statistics. "Database Administrators and Architects." Occupational Outlook Handbook. 2026. https://www.bls.gov/ooh/computer-and-information-technology/database-administrators.htm.

[4] The Register (Lindsay Clark). "The clock's ticking for MySQL 8.0 as end of life looms." Coverage citing Oracle's support date and Percona monitoring data. October 31, 2025. https://www.theregister.com/2025/10/31/mysql_8_support_end/.

[5] Microsoft. "SQL Server 2016 end of support is here: plan your next steps." Microsoft SQL Server Blog. July 14, 2026. https://www.microsoft.com/en-us/sql-server/blog/2026/07/14/sql-server-2016-end-of-support-is-here-plan-your-next-steps/.

[6] PostgreSQL Global Development Group. "Versioning Policy." 2026. https://www.postgresql.org/support/versioning/.

[7] Microsoft. "SQL Server 2017." Microsoft Lifecycle Policy. https://learn.microsoft.com/en-us/lifecycle/products/sql-server-2017.

[8] The Register. "Oracle extends 19c database support to 2032, making it 'longest strategic release'." Coverage of Oracle's Lifetime Support Policy update. February 18, 2025. https://www.theregister.com/2025/02/18/oracle_extends_19c_support/.

[9] Amazon Web Services. "Estimating the charges for Amazon RDS Extended Support." AWS Cloud Financial Management Blog. March 25, 2024. https://aws.amazon.com/blogs/aws-cloud-financial-management/estimating-the-charges-for-amazon-rds-extended-support/.

[10] GitHub. "GitHub Availability Report: August 2025." September 11, 2025. https://github.blog/news-insights/company-news/github-availability-report-august-2025/.

[11] GitHub. "GitHub Availability Report: January 2025." February 11, 2025. https://github.blog/2025-02-11-github-availability-report-january-2025.

[12] GitHub. "GitHub Availability Report: April 2025." 2025. https://github.blog/news-insights/company-news/github-availability-report-april-2025.

[13] Slack. "Slack System Status: February 26, 2025." Incident history and root cause summary. 2025. https://slack-status.com/2025-02-26.

[14] The Matrix.org Foundation. "Post-mortem" of the September 2, 2025 matrix.org homeserver outage. October 2025. https://matrix.org/blog/2025/10/post-mortem/.

[15] Clerk. "Service outage postmortem, March 10, 2026." 2026. https://clerk.com/blog/2026-03-10-service-outage-postmortem.

[16] Cloudflare. "Cloudflare outage on November 18, 2025." Post-mortem. 2025. https://blog.cloudflare.com/18-november-2025-outage/.

[17] Sato, D. "ParallelChange." martinfowler.com bliki. 2014. https://www.martinfowler.com/bliki/ParallelChange.html.

The views in this paper are the author's own and do not represent any employer. Scenarios are hypothetical. This paper is not legal, financial, or compliance advice.