ChatGPT

Latest Trends in SQL Database Configurations (Sept 2025 – Jan 2026)

Introduction

SQL databases remain at the core of modern applications, and the period from late 2025 into 2026 has seen significant advances in popular SQL systems. Developers are doubling down on **PostgreSQL, SQLite, and Supabase** as foundational technologies, extending them with new features, plugins, and cloud integrations. The focus is on **performance improvements, developer experience, and adaptability** for emerging domains like AI-driven agents (agentic robotics) and distributed edge computing. In this timeframe, we observe:

- **PostgreSQL** pushing the envelope with major version releases (v17 and v18) that improve performance (especially on modern hardware) and add features for easier scaling and integration. New cloud-native Postgres services (e.g. Neon) embrace a serverless model to support AI and microservices workloads.
- **SQLite** evolving beyond a "simple" embedded database into a powerhouse for edge and mobile scenarios. The community has built extensions and forks (like libSQL) adding replication, WebAssembly support, and cloud APIs – making SQLite a viable distributed database on everything from phones to global edge networks.
- **Supabase** emerging as a **full-stack backend platform** centered on Postgres. It's hugely popular among "DevOps vibe-coders" – developers who rapidly prototype apps – because it bundles SQL databases with authentication, storage, and new capabilities like background jobs and vector search for AI. Supabase's rapid growth and multi-billion valuation underscore the trend of choosing open-source SQL (Postgres) with developer-friendly tooling over proprietary systems [1] [2].

Below, we delve into each of these technologies, highlighting the latest configurations, tools, and projects that rose to prominence in late 2025, with notes on cross-platform usage (from NixOS desktops to mobile and IoT devices) and advanced plugins/packages relevant to robotics and distributed computing.

PostgreSQL: New Features and Serverless Innovations

PostgreSQL solidified its position as *"the world's most advanced open-source database"* with the release of **PostgreSQL 18** in late 2025. This version introduced a new asynchronous I/O subsystem yielding up to **3× faster read performance** in certain workloads, and expanded indexing capabilities to accelerate more queries [3]. Major-version upgrades became smoother – PostgreSQL 18 can retain planner statistics after upgrade, meaning an updated server reaches optimal performance much faster than before [4] [5]. Developers also gained features like **virtual generated columns** (computed at query time instead of stored), a built-in `uuidv7()` function for better-indexed UUIDs, and OAuth2 single sign-on support [6] [7]. These improvements benefit all deployments, but especially DevOps teams managing large distributed systems: less downtime for upgrades and better performance tuning out-of-the-box.

PostgreSQL 17, released in 2024, laid the groundwork with upgrades like the SQL/JSON `JSON_TABLE` function (for treating JSON data as relational rows) and incremental backup support [8] [9]. Together, PG 17 and 18 give developers safer upgrades, improved high-availability (e.g. logical replication enhancements), faster query execution (skip scans on multi-column indexes, improved joins), and clearer observability of query timing [8] [10]. Notably, PostgreSQL continues to optimize for modern **hardware architectures** – v18 added specific CPU optimizations for ARM processors (NEON/SVE intrinsics) to speed up operations like bit counting [11]. This means Postgres runs efficiently not just on x86 servers but also on ARM-based devices common in robotics (e.g. Raspberry Pi or Jetson boards), aligning with the "cross-hardware" goal.

Beyond core server improvements, **PostgreSQL's extensibility and ecosystem are driving its popularity in new domains like AI and serverless computing**. PostgreSQL's design has always favored extensibility (supporting custom data types, indexes, etc.), and this is paying off in the AI era [12]. A key example is the **pgVector** extension, which enables efficient vector similarity search inside Postgres. This allows one database to handle traditional relational data *and* AI embedding vectors together. By late 2025, many startups chose Postgres with pgVector instead of separate vector databases. The integration of relational and vector data in one system simplifies AI applications – e.g. using SQL to store user data and perform semantic searches on embeddings in the same query [13] [14]. Developers no longer need to glue together a Postgres for metadata, a specialized vector DB for AI, and maybe a cache; Postgres with extensions can do it all in a unified, **ACID-compliant** way [15] [16]. This "Swiss-army knife" capability is a major reason **AI startups are gravitating to PostgreSQL** [12].

Another massive trend is **serverless Postgres**. Projects like **Neon** (an open-source cloud Postgres service) decouple storage from compute, allowing databases to **pause when idle and instantly clone or spin up** on demand. Neon's approach (built on Postgres compatibility) attracted attention for enabling use cases like ephemeral databases for AI agents. In fact, *over 80% of databases created on Neon by late 2025 were automatically generated by AI agents*, not humans [17]. This indicates a new pattern: autonomous processes spinning up short-lived Postgres instances to handle bursts of tasks. Neon's technology (which can launch a new Postgres instance in ~0.5 seconds and auto-scale it down) is tailored for machine-driven workloads that need speed and cost-efficiency beyond what always-on databases can provide [18]. The importance of this paradigm was underscored by **Databricks' acquisition of Neon for ~$1B in 2025**, aiming to integrate serverless Postgres into data/AI platforms [19]. For DevOps teams, the rise of serverless Postgres means less manual scaling and the possibility of running many isolated Postgres instances across a distributed environment (e.g. one per microservice or per robotics mission) with minimal overhead.

Finally, PostgreSQL's rich **extension ecosystem** remains a highlight. In 2025, popular extensions and plugins saw active development: **PostGIS** (for geospatial SQL), **TimescaleDB** (time-series), **pg_cron** (scheduling jobs inside the DB), and **logical replication tools** for distributed systems. Cloud providers and projects like Supabase and Azure Postgres bundle many of these to provide "Postgres++" experiences. The key takeaway is that **Postgres is not standing still** – it's rapidly incorporating features needed for modern workloads while retaining the reliability and robustness built over decades [20]. For teams using NixOS, Postgres is well-supported (NixOS modules exist for declarative Postgres setup [21]), making it straightforward to integrate the latest Postgres (or specific versions) into reproducible deployments. In summary, PostgreSQL as of 2026 is faster, more flexible, and more cloud-ready than ever – fitting the needs of both **traditional enterprise data and cutting-edge AI/robotics projects**.

SQLite: From Lightweight Embedded DB to Edge Computing Workhorse

SQLite has always been known for its **zero-configuration, embedded design** – famously *"small, fast, reliable. Choose any three."* It remains ubiquitous on mobile devices and IoT (every smartphone has SQLite on it), but recent developments show SQLite expanding its domain with **advanced configurations and new projects** that make it a star in cloud and edge computing.

One notable trend is **enhanced functionality within SQLite itself**. The SQLite project continues to release updates frequently; in late 2025 it reached **version 3.51** with significant new features. For example, SQLite 3.51.0 (Nov 2025) introduced **JSON improvements** – new functions `jsonb_each()` and `jsonb_tree()` that, like their `json_each()` counterparts, iterate over JSON values but return them in a binary JSON (JSONB) format for efficiency [22]. This complements SQLite's existing JSON support (which was already quite robust) and shows a commitment to handling semi-structured data. SQLite 3.50 (mid-2025) similarly strengthened JSON capabilities and other areas (e.g. fixes for JSON5 handling [23]). In addition, performance tweaks are constant – v3.51 reduced CPU overhead for committing read transactions and optimized certain queries (like fast detection of empty joins) [24]. **Data integrity and type safety** got a boost too (enforcing strict typing on generated columns) [25]. Notably for cross-platform use, SQLite's *JavaScript/WebAssembly build* received attention: by end of 2025 the official SQLite could be compiled to a 64-bit WASM module easily [25], making it simpler to run a full SQL database directly in web browsers or other WASM environments. This is important for projects where you might want a database in a frontend or a serverless function at the edge.

While SQLite's core stays lean and portable, the community has built **extensions and forks to add "missing" features for distributed and cloud use**. A prime example is **libSQL**, an open-source fork of SQLite announced in 2023 (and actively developed through 2025). **LibSQL** retains SQLite's lightweight, embeddable nature but adds capabilities like **built-in replication, a client/server mode (HTTP and WebSocket protocols), and cloud-friendly synchronization** [26] [27]. Essentially, libSQL = SQLite + modern enhancements for the edge: you can run it in a browser (WASM), in Node.js, or as a remote database server, and keep multiple replicas in sync. It's the engine behind the **Turso** database-as-a-service, which provides geographically distributed SQLite databases. By enabling a network layer and live replication, libSQL lets developers use SQLite for **distributed computing scenarios** – for example, a swarm of robots or IoT sensors could each use a local SQLite for fast writes and occasionally sync to a cloud endpoint. This is a shift from the old notion of SQLite as strictly local; now it can participate in cloud data flows while still being embedded and efficient. The popularity of libSQL is evident – as of late 2025 it has over 15k GitHub stars and is considered a leading-edge project in the database space [28].

Another innovative development is **serverless edge databases built on SQLite**. **Cloudflare D1** is a prominent case: D1, announced in 2022 and improved through 2025, is Cloudflare's managed SQL database that runs across their global edge network, and it uses SQLite under the hood. The idea is to marry SQLite's simplicity with Cloudflare's edge computing platform. D1 adds a **distributed replication layer** and durability on top of SQLite, so developers get an automatically replicated SQLite database accessible from Cloudflare Workers (serverless functions). This fills the niche for applications that need full SQL capabilities but not the complexity of running a dedicated PostgreSQL/MySQL instance [29] [30]. As Cloudflare's engineers noted, SQLite was *"the perfect fit for our first entry into databases"* at the edge because Workers run between client and server, and a lightweight file-based DB that can be spun up quickly aligns with that

model [30]. The **developer experience** is a major selling point – you can go from nothing to a globally deployed SQL database **"in an instant"** [31]. By late 2025, D1 was in open beta and developers were experimenting with using it for things like caching, user preferences, and even light analytics at the edge. Its design favors eventual consistency (replicas sync in background) for the sake of speed and global scale [32]. The success of D1 and similar offerings underscores a broader trend: **SQLite is eating the cloud** (as one article title put it) by appearing in many distributed systems where using a heavyweight DB would be overkill [33].

From a **cross-hardware perspective**, SQLite's footprint remains unparalleled: it can run on virtually any device (from a microcontroller up to a mainframe). The new enhancements don't change that – e.g. **SQLite on NixOS** is trivial since it's just a package (and one can even compile custom extensions into it via Nix). On mobile, Android and iOS continue to use SQLite for app storage, and now with official support for things like **ARM64 and WASM 64-bit**, it's ready to run natively in new environments (like directly in a web app via WASM, or on Apple's M-series ARM chips without issue). **Energy efficiency** is also a hidden strength: because SQLite is lightweight (no server process) and written in C for efficiency, it tends to use less power – an interesting point for robotics or edge devices running on battery. It's often more energy-efficient to use SQLite for local storage than to constantly send data over a network to a cloud database.

In summary, **SQLite in late 2025/early 2026 is far from "just a local file database."** It's evolving to support modern app needs: handling JSON data seamlessly, syncing across the cloud, running in web/edge contexts, and even doing basic distributed processing. Projects like libSQL and D1 show that developers want the *simplicity and speed* of SQLite **combined with** the *scalability and connectivity* of a cloud database. For an organization using a NixOS + Nushell environment, this means you can pull in SQLite or libSQL as dependencies via Pixi/Nix, script your database tasks in Nushell or Python, and have a lot of flexibility in where and how your data is stored – from on-device SQLite for a robot, to a libSQL instance that syncs to the cloud for aggregate analysis.

Supabase: The Integrated Platform for Modern Development

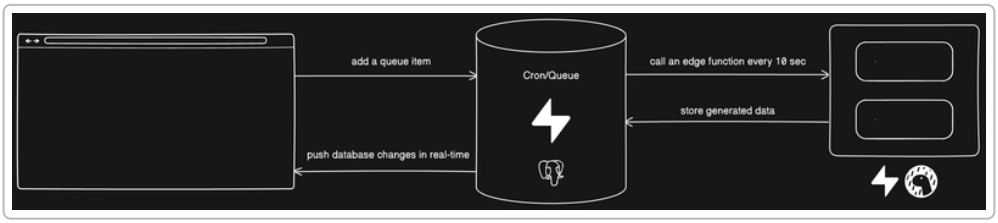

Figure: Supabase's unified architecture brings together the database (PostgreSQL) and serverless services (auth, storage, functions, queues) under one roof. This "new world" architecture simplifies what used to require multiple separate services (queue managers, job runners, etc.) [34]. In the diagram, all background tasks, scheduling, and data live within Supabase's platform, illustrating why it's called *"the backend that eats backends."*

Supabase has had a meteoric rise through 2025 as a go-to solution for building applications quickly. It is an **open-source backend-as-a-service** that combines a Postgres database with a suite of tools (authentication, file storage, real-time subscriptions, and more). Its appeal to developers – especially those in startups, hackathons, or "vibe coding" communities – is that it significantly cuts down the DevOps overhead. By late 2025, Supabase became *"the backend of choice for the vibe-coding world,"* as one TechCrunch piece put it [1].

This is reflected in its **massive community adoption and funding**: Supabase's developer-centric approach (emphasizing ease of use and not catering exclusively to enterprise demands) led to a $200M funding round at a $2B valuation in April 2025, and by the end of 2025 it had reached a $5B valuation [2] [1]. In other words, the market recognizes that many developers prefer a **Postgres-based stack** with an open source ethos, rather than proprietary closed-source services.

What's new in Supabase lately? Supabase has rapidly expanded its feature set in late 2024 and 2025, turning it from "just a database + auth" into a **full-fledged serverless platform**. One big addition was **Supabase Edge Functions** (powered by Deno runtime) and support for **background tasks, cron jobs, and queue workers**. In the past, if you used Supabase for your database, you might still need external services for job queues or scheduled tasks (for example, using Redis & BullMQ for queues, or an AWS Lambda for cron jobs). In late 2024, Supabase introduced a built-in **Queue system and Cron scheduler** that work with its Edge Functions [35] [36]. By early 2025, this was refined into an experience where *all* these pieces live in Supabase: you can enqueue jobs in a Postgres-backed queue table, have a Deno edge function process them, and schedule that function with a cron – all configured through Supabase's dashboard or CLI. The result is a **dramatically simpler architecture** for developers. As the example in the figure above shows, previously one might string together Railway or Render to host a worker, plus a Redis instance for state, plus Supabase for data [37] [38]. Now, *"everything lives under one roof… just pure, streamlined efficiency."* [34]. Supabase provides the queue, the function hosting, and even an easy UI for cron scheduling. This "batteries-included" approach allows small teams (or even solo developers) to implement complex backend logic (like processing videos, sending emails, running AI inference) without standing up separate infrastructure. A dev community blog exclaimed that with these additions, *"Supabase now offers everything you need in one platform… They didn't just add a missing piece – they completed the backend puzzle."* [39] [40].

Another area of rapid progress is **integration of AI and data features**. Because Supabase sits on Postgres, it benefited from the aforementioned pgVector extension boom. Supabase **offers pgVector out-of-the-box**, letting developers store embeddings and perform vector similarity searches without setting up a separate vector DB [41]. In April 2025, Supabase even launched "**Automatic Embeddings**" – a feature to generate embeddings from text data and store them in a pgVector column asynchronously [42]. This caters to the wave of **AI-powered applications** (for example, semantic search, recommendation systems, chatbots that use a knowledge base). Essentially, Supabase is positioning itself as *an AI-ready backend*: you have your data in Postgres (with JSON support for unstructured data), and you have vectors for AI similarity, all under uniform auth and APIs. One Supabase article touted the two-pronged storage approach: use **pgVector for tight, low-latency vector queries on your core data**, and if you have truly massive embedding sets, they introduced **"Vector Buckets"** to offload large vectors to external storage while keeping smaller ones in Postgres [43]. This nuance shows that Supabase is aware of balancing Postgres's strengths with scalability limits – small to medium AI use-cases can stay entirely in the Supabase/Postgres ecosystem, which is simpler for developers.

Supabase's popularity in **DevOps and "vibe coding"** circles comes from its **developer-first philosophy**. The company famously turned down large enterprise contracts that would distract from their core product vision [1] [44]. The bet (which seems to be working) is that by focusing on developer experience – "optimize for developer happiness over enterprise features" – they'll capture the huge market of projects that want quick, powerful backends without enterprise bloat [45]. This strategy aligns with broader trends: an explosion of startups and personal projects needs reliable backends, and AI-assisted coding tools are making it easier to generate apps – all those apps will need a database and serverless functions [46]. Supabase is positioning itself to catch that wave, providing the default backend for the new generation of

apps (in many ways trying to succeed Firebase's role, but with SQL power). Even for **agentic robotics** or IoT scenarios, Supabase can be useful: a fleet of robots could log data to a central Supabase database, or use its real-time features to sync state. Because it's built on Postgres, it can easily run on-premises or in the cloud, and tools like **NixOS** can deploy Supabase's Docker containers in a reproducible way if needed.

In the context of NixOS/Pixi/Nushell environments: while Supabase is primarily a cloud service, it's open source, which means you can run **Supabase locally** (there's a Docker-based CLI to start the whole stack). This is great for development on NixOS – one could use Pixi/Nix to ensure the required Docker or Podman is present, then `supabase start` to get a local instance for testing. Nushell, being a modern shell, can easily interact with Supabase's HTTP APIs or the `psql` CLI for Postgres in structured ways (Nushell can parse JSON outputs, etc.), potentially making it a nice interface for scripting Supabase operations or analyzing data.

In summary, Supabase represents the convergence of SQL databases with the convenience of a managed platform. It exemplifies "advanced configuration" not in the sense of DB tuning, but in bundling **advanced capabilities (extensions, background workers, etc.) into a plug-and-play package**. Its recent popularity surge and feature rollouts (cron, queues, vector search, SSO integrations) make it a **leading project on GitHub** (high stars and contributions) and a cornerstone in many new tech stacks. For a team focused on distributed compute and robotics, Supabase offers a quick way to stand up a cloud backend that is robust and scales, without having to individually configure Postgres, authentication, and messaging systems – those come pre-integrated. This allows developers to focus on the *application logic and device integration*, while trusting the platform for the heavy lifting of data persistence and job processing.

Conclusion

Across PostgreSQL, SQLite, and Supabase, the late 2025 landscape shows a clear theme: **the SQL ecosystem is innovating to stay relevant in an AI-driven, distributed world.** These tools are not stuck in the old client-server paradigm – they're embracing serverless deployment, edge computing, and integration with non-SQL data types (JSON, vectors). Projects on GitHub reflect this trend, from serverless Postgres services to SQLite-based cloud databases and the booming Supabase repository, which has one of the most active communities.

Crucially, these advancements maintain a **cross-platform philosophy**. Whether you're running NixOS on a laptop, deploying ARM-based robots, or building mobile apps, you can leverage the same database engines and configurations: - PostgreSQL can run on your development NixOS machine and then scale to a cloud cluster or a tiny ARM device as needed, with consistent behavior and configuration (thanks to tools like Nix for managing its config). - SQLite (and its variants like libSQL) can be embedded in mobile apps, run in a browser via WASM, or serve as an edge cache on a Cloudflare data center – the SQL syntax and lightweight reliability remain constant. - Supabase offers a unified interface to these SQL capabilities through RESTful and GraphQL APIs, Deno functions, etc., accessible from any platform or language, lowering the barrier to use Postgres in diverse environments.

For **agentic robotics**, these trends mean easier data sharing between agents and cloud: a robot can carry a SQLite database for local operation and periodically sync to a central Postgres (via extensions or custom tools), or directly interact with a Supabase instance for live telemetry and commands. For **DevOps teams**, the tooling around SQL is better than ever – you can declare your DB setups in Nix, automate migrations, and use cloud services for scalability without abandoning the comfort of SQL.

In essence, the SQL world is experiencing a renaissance of popularity and capability. By combining decades-hardened reliability with cutting-edge features, SQL databases (especially PostgreSQL and SQLite) and platforms built on them (like Supabase) are ensuring they remain the backbone of modern applications – from mobile apps to distributed robotic fleets – even as requirements evolve into the era of AI and edge computing. The period from September 2025 to now has proven that **innovation in the SQL space is alive and well**, with a focus on empowering developers and scaling to meet new challenges [47] [48].

Sources: The insights above are supported by the official PostgreSQL release notes [49] [11], SQLite's 2025 changelog [22] [25], industry articles on Supabase's growth and strategy [1] [45], a Medium deep-dive on why AI startups are choosing Postgres (highlighting Supabase, Neon, and pgVector) [17] [14], and various technical blogs. Notably, InfoQ and Cloudflare blogs illustrate how SQLite is used in edge services like D1 [29] [30], and developer posts on Dev.to demonstrate Supabase's new all-in-one backend features with queues and cron jobs [34]. These sources reflect the state-of-the-art as of early 2026, showing a vibrant and evolving SQL landscape.

---

[1] [44] [45] [46] [47] Supabase Says No to $Millions
https://www.beyondthebuzz.ai/p/supabase-says-no-to-millions

[2] [12] [13] [14] [17] [18] [19] [48] Why AI Startups Choose PostgreSQL: Supabase, Neon, pgVector | by Takafumi Endo | Medium
https://medium.com/@takafumi.endo/why-ai-startups-choose-postgresql-supabase-neon-pgvector-7d1e1383b3dd

[3] [4] [5] [6] [7] [10] [11] [20] [49] PostgreSQL: PostgreSQL 18 Released!
https://www.postgresql.org/about/news/postgresql-18-released-3142/

[8] [9] 10 Postgres 17 Features You'll Actually Use in Production | by Hash Block | Medium
https://medium.com/@connect.hashblock/10-postgres-17-features-youll-actually-use-in-production-1cda146ef49e

[15] pgvector/pgvector: Open-source vector similarity search for Postgres
https://github.com/pgvector/pgvector

[16] Supabase already gives you powerful tools for vectors, such as ...
https://www.threads.com/@supabasecom/post/DSpyiAxAih8/supabase-already-gives-you-powerful-tools-for-vectors-such-as-pgvector-in

[21] NixOS Manual
https://nixos.org/manual/nixos/stable/

[22] [24] [25] SQLite Release 3.51.1 On 2025-11-28
https://sqlite.org/releaselog/3_51_1.html

[23] SQLite Release 3.50.0 On 2025-05-29
https://sqlite.org/releaselog/3_50_0.html

[26] [27] LibSQL | Astro Vault
https://vault.llbbl.com/content/databases/libsql/

[28] libSQL An open-source, community-driven fork of SQLite ... - Threads
https://www.threads.com/@dailyfoss/post/DPBxzduiJxD/libsqlan-open-source-community-driven-fork-of-sqlite-with-open-contributions-sta

29 30 31 32 Cloudflare D1 Provides Distributed SQLite for Cloudflare Workers - InfoQ
https://www.infoq.com/news/2022/05/cloudflare-d1-sqlite-workers/

33 SQLite Is Eating the Cloud in 2025: Edge Databases, Replication ...
https://debugg.ai/resources/sqlite-eating-the-cloud-2025-edge-databases-replication-patterns-ditch-server

34 35 36 37 38 39 40 Supabase Just Got More Powerful: Queue, Cron, and Background Tasks in Edge Functions - DEV Community
https://dev.to/taishi/supabase-eats-the-backend-fll

41 pgvector: Embeddings and vector similarity | Supabase Docs
https://supabase.com/docs/guides/database/extensions/pgvector

42 Automatic Embeddings in Postgres - Supabase
https://supabase.com/blog/automatic-embeddings

43 Introducing Vector Buckets - Supabase
https://supabase.com/blog/vector-buckets