Episode Book

EPISODE #6520

Mapping Multi-Tenancy Across Vector Databases

PUBLISHED	10 November 2026 at 11:46 UTC
DURATION	27:41
LISTEN	myweirdprompts.com/episode/vector-database-multi-tenancy-isolation-96702/
HOSTS	Corn, Herman, Dorothy, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	16 cited by the research (listed at the end)
BOOK GENERATED	11 November 2026 at 06:49 UTC

SYNOPSIS

Approximate nearest neighbor indexes scan neighborhoods without knowing who owns a vector, forcing vector databases to resolve multi-tenancy either inside physical structures or through post-search filtering. Examining architectural trade-offs across Pinecone namespaces, Weaviate sharded tenants, Milvus partition keys, Chroma, and pgvector reveals why physical data separation is often vastly cheaper and safer than relying on metadata filters. The five-tier isolation ladder explains how to balance schema flexibility, latency, and noisy neighbors without getting lost in vendor jargon.

TRANSCRIPT



Corn

Five vendors, five words for the same wall. Namespace, tenant, partition, database, row-level security. Daniel wants to know what's actually behind the drywall.



Herman

And, knowing Daniel, he doesn't want the feature-matrix version. He wants the thing he can carry in his head to whichever database lands on his desk next year.



Corn

That's the ask, yes. Pinecone namespaces, Weaviate tenants, Milvus with a four-word vocabulary all to itself, Chroma with tenants and databases, and pgvector where you're back in Postgres arguing with row-level security. His question is what the real isolation levels are under the names, and how you pick one. He said he wants a mental model, not a comparison table. So let's strip the vocabulary off and ask what actually gets isolated.



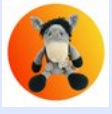
Herman

Start with the mechanism, because everything else follows from it. An approximate nearest neighbor index doesn't see tenant boundaries. It walks a graph or probes a set of lists, hands the executor whatever candidates are closest, and row visibility gets reconciled afterward. That's true of every vector database, from the managed ones down to pgvector sitting in a Postgres schema. The index is a neighborhood machine. It has no idea who owns a vector.



Corn

So the question underneath all the vocabulary is one question. Does the index know about the tenant, or does the index find neighbors blindly and someone else cleans up after it?



Herman

That's the whole ladder, compressed. If the boundary is inside the index, or the shared index doesn't exist, then isolation is structural. If the boundary is outside the index, you're paying for it in recall, latency, and money, every single query.



Corn

Give me the rungs.



Herman

Five, hardest to softest. Separate instance, own compute, own failure domain, nothing shared. Then separate index or collection, a distinct physical object with its own schema and its own graph. Then partition or namespace, a physically separated slice inside a shared object, addressed by name when you query. Then partition key, which is a routing key that maps tenants onto a fixed number of physical partitions, several tenants to a partition. And at the bottom, metadata filter on a shared index. One graph, a tenant_id field, a where clause.



Corn

The first three move the wall inside the building. The last two leave it painted on the floor.



Herman

Nicely put, even. And the levels aren't cosmetic. Levels one through three change what the index physically contains. Levels four and five keep one graph for everyone and reconcile visibility after the fact. That reconciliation is where the interesting problems live, and it's also where the money goes.



Corn

Okay. So map the vendors. Pinecone first, since Daniel named it first.



Herman

Pinecone namespaces are level three, cleanly. A namespace is a physically separated slice inside a serverless index, and you address it by name at query time. Their docs are unusually direct about it. They call namespaces the most efficient way to do multitenancy, and they say each namespace is stored separately, so you get physical isolation between tenants, which reduces the risk of application bugs searching the wrong tenant's data. Limits run from a hundred namespaces on the Starter plan to a million on Enterprise.



Corn

A hundred to a million. That's not a pricing tier, that's a different product depending on your invoice.



Herman

It does mean the limit is a business line, not an architecture line. The architecture goes further, they say they'll accommodate more for specific cases. What I love about Pinecone's docs is they publish the cost arithmetic. Search cost is based on namespace size, one read unit per gigabyte. A hundred tenants of one gig each, searching one tenant's namespace costs you one RU. Put all hundred tenants in a single hundred-gigabyte namespace and use metadata filtering, the same search costs a hundred RUs, because it scans everything regardless of the filter.



Corn

The isolated option is a hundred times cheaper. So the compliance feature is also the discount.



Herman

That's the inversion nobody expects. Everyone assumes isolation is the expensive, luxury option and sharing is the lean startup move. In vector search it's backwards, because cost tracks data scanned. Isolating the tenant shrinks what you scan.



Corn

And the metadata filter approach, the one that costs a hundred times more. What do Pinecone say about it?



Herman

They call it, quote, a common anti-pattern. Storing all data in a single namespace and using metadata filters to scope searches to specific users. Their objections are practical, not philosophical. Large in-filters balloon the network payload and latency, and each in or nin operator is capped at ten thousand values. Exceed it and the request just fails. Their guidance is to prefer namespaces unless you require querying across tenants.



Corn

Ten thousand values in one filter. At that point the where clause is the application.



Herman

Right, and it's a where clause you're maintaining by hand. One other Pinecone detail worth keeping, offboarding. Delete the namespace and the tenant is gone, and their word for it is that it's lightweight and almost instant. Compare that to the metadata model, where offboarding a tenant means listing every record by ID prefix and batch-deleting them. One of those operations you can run at two in the morning, and the other you schedule.



Corn

Weaviate next. Daniel called it native tenants, which sounds like marketing until you find out what's under it.



Herman

It's a shard per tenant. A shard is a self-contained storage and query unit, and in their native multi-tenancy every tenant's data lives in its own shard. Structurally, that's level three, same rung as Pinecone namespaces, different word. What Weaviate adds on top is the Tenant Controller, which I find elegant. Three states per tenant. ACTIVE, meaning it's using memory and compute. INACTIVE, resources freed but quickly accessible. OFFLOADED, pushed to cold storage like S3 until someone touches it again. And it can activate a tenant automatically the moment a search or update lands on it.



Corn

So the idle ninety percent of your tenants cost almost nothing.



Herman

That's the bet, yes. Most tenants are dormant most of the time, and keeping them all warm in memory is the actual cost driver at scale. Weaviate claims over a million tenants per cluster on that basis. Which is a vendor claim with no benchmark attached, I'd hold it loosely. And their own docs carry the honest caveat. Tenants still share node resources, CPU, memory, I/O, network. A heavy enough workload on one tenant can still squeeze the node. Sharding isolates the data, not the machine.



Corn

Noisy neighbors, one floor down.



Herman

One floor down and with a better thermostat. The other Weaviate piece is collection-level isolation, and it matters for a different reason. Every collection has its own vector space, meaning different collections can hold different embeddings of the same object. So if your tenants need different embedding models, or different dimensions, collection-per-tenant is the argument. Same logic shows up everywhere. Metadata filtering can't give a tenant its own dimensions, because the index dimensions are fixed for everyone inside it.



Corn

Milvus is the one Daniel listed with a four-part vocabulary. Databases, collections, partitions, partition keys.



Herman

Milvus is the only vendor that publishes the ladder explicitly, which I appreciate, because you can map everyone else onto it. Database level, sixty-four tenants by default, fully separated data, which they call enterprise-grade isolation ideal for regulated environments. Full schema flexibility per tenant, role-based access control supported, hot and cold data handling supported. Collection level, up to sixty-five thousand collections by default, physically isolated from one another, medium schema flexibility. Partition level, a thousand and twenty-four partitions per collection, tenant data physically separated, cross-tenant queries supported, but RBAC is not supported at partition level. And then partition key level, which they call the most scalable approach, supporting millions of tenants.



Corn

Millions of tenants, with what catch?



Herman

Their own words are relatively weak data isolation, because multiple tenants can share a physical partition. The partition key routes tenants into sixteen physical partitions. Sixteen. So tenant number four thousand and tenant number nine hundred and one may



Herman (continued)

be sitting in the same physical bucket, logically separated and not physically isolated.



Corn

Sixteen partitions for millions of tenants. That's a hotel with sixteen rooms and a very confident concierge.



Herman

And the gotcha buried in their docs is a nice one. They recommend a scalar index on the partition key field, because each physical partition holds multiple tenants, and Milvus evaluates the tenant filter against every entity in the searched partitions. Without the index, that's a full scan. So even the scalable rung quietly reintroduces the filtering problem, just one level shallower.



Corn

The trade-off table. You said they publish it.



Herman

Clearest in the industry. Three rankings. Scalability, partition key beats partition beats collection beats database. Strong isolation, database and collection tie at the top, then partition, then partition key. Schema flexibility, database, then collection, then partition and partition key tied at the bottom. And the cross-tenant column, only partition and partition-key support it. Read that table and you realize the vocabulary differences between Pinecone and Weaviate and Milvus are almost decoration. Pinecone namespaces and Weaviate tenants are both level three. Milvus just spans levels two through four and labels each rung. Qdrant, which we're leaving for the other episode, sits on the same ladder with its own words.



Corn

Daniel used the word federation for all this, by the way.



Herman

Worth one line of cleanup. In the vector DB world, federation usually means querying across separate clusters. What we're talking about is isolation levels within one deployment. Cross-tenant search inside one system, which is different from federating systems.



Corn

So that's the ladder, and most of the vocabulary mapped. Chroma next.



Herman

Chroma gives you tenant, then database, then collection, a three-deep hierarchy. Tenant is the top-level account boundary, database is a project space inside the tenant, collection is the dataset inside that. On a single-node setup, all of it, tenancy, databases, collections, documents, sits in one SQLite file.



Corn

The entire multi-tenant security model, in a SQLite file. There's an honesty to that.



Herman

There really is. Their cookbook names four strategies, and I do admire the candor of the framing. User per doc, which is the metadata filter, level five. User per collection. User per database. User per tenant. And they label the whole thing, quote, a naive approach to multi-tenancy and probably not suited for production environments. Their own ladder, their own warning label.



Corn

They call it naive themselves.



Herman

They do, and they spell out why. User per doc is error-prone, messing up the filtering can leak data across users, and filtering gets slow as users and documents grow. User per collection is error-prone in a different direction, messing up the collection name can leak data. One typo swaps the rung you're standing on.



Corn

pgvector, then. Where Daniel says you're back to row-level security.



Herman

Back to Postgres primitives, which means three models, and I actually think this is the clearest teaching version of the whole ladder, because Postgres makes you build each rung by hand. Shared table plus RLS is level five. One document embeddings table, every tenant's rows, a tenant_id column, an RLS policy reading a session variable. Densest layout, cheapest to operate, one index, one vacuum target. And also, in the words of the pgvector guide I read, the model where the ANN-versus-visibility reconciliation problem is sharpest. The index is built over all tenants, so your query walks foreign rows before they're filtered out. You spend recall budget on candidates you're going to throw away.



Corn

The index does the work, then the bouncer checks the list at the door.



Herman

The bouncer checks the list after everyone's already inside, which is worse. Schema-per-tenant is level two. Each tenant owns a schema with its own embeddings table and its own index, and isolation becomes a grant and search_path question rather than a per-row predicate. The index contains only that tenant's vectors, so recall isn't diluted. The cost is multiplicity. Hundreds of schemas means hundreds of indexes to build, tune, and vacuum, and every DDL migration fans out across all of them. Best for few large tenants with strict compliance separation. Partition-per-tenant is the middle ground, level three and four hybrid. Postgres prunes irrelevant partitions before RLS is even evaluated, and each partition carries its own ANN index, so scans stay tenant-local while DDL stays centralized. The guide's threshold is once any single tenant exceeds a few million rows, roughly five million, you move that way.



Corn

Which is the tenant-size-skew answer in one sentence.



Herman

It is. Many small tenants, shared table. One whale tenant, partition-per-tenant. Few big regulated tenants, schema each.



Corn

What does the recall penalty actually cost you in Postgres? Numbers, not adjectives.



Herman

The guide puts numbers on it, which almost nobody else does. With shared-table RLS you're post-filtering, so you over-fetch candidates and then discard the foreign ones. To recover the recall you lost, they suggest raising hnsw ef_search from its default of forty to somewhere between one hundred and two hundred. If you're on ivfflat, raise probes from the default of one to ten to forty. That's the closest thing in all this research to a quantified recall penalty for the metadata-filter approach, and it's Postgres-specific. Nobody has published the equivalent for the vector databases.



Corn

Turn the knob up, burn compute, pretend nothing happened.



Herman

It's a recall subsidy. And the hardening list for shared-table RLS is its own education. A nullable tenant_id is a leak waiting to happen, so it's NOT NULL, full stop. Plain ENABLE row level security exempts the table owner, so you use FORCE. You split the USING clause for reads from the WITH CHECK for writes. You bind the tenant context with SET LOCAL, per transaction, never per pool, because a pool connection outlives the request and the setting can leak to the next borrower. And you pin row_security on, because a client can set it to off, and if the role has BYPASSRLS that disables policies entirely.



Corn

Per transaction, never per pool. That's the line that separates the people who've been paged from the people who haven't.



Herman

It's the line that separates people who've debugged a connection pool at three in the morning. There's a subtler leak in the same guide, too. Bad reconciliation doesn't always surface as the wrong document in the results. It can leak cluster density through the distance-score distribution, or through query timing. You can fingerprint which tenants exist from timing alone.



Corn

Wait, that's a lot to think about. Before we go on, I think we have a call from a listener with a regular segment.

D

Dorothy

Hello dear, it's your mother. It's time for Dorothy's History of Toast.

D

Dorothy

The first pop up toaster for the home, the Toastmaster, went on sale in nineteen twenty six.

D

Dorothy

Nineteen twenty six, and I suppose that was meant to make breakfast easier, though I can't say it ever did.

D

Dorothy

Anyway, sweetheart, don't forget your scarf, it's chilly here. Ring me later.



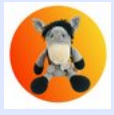
Herman

Mum, always a pleasure. The Toastmaster.



Corn

She'll deny the connection, but that was a segment about physical isolation for bread.



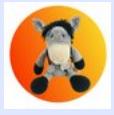
Herman

Where were we. Timing leaks, pgvector hardening.



Corn

What I want to pull out of your pgvector tour is the decision framework. We've mapped five vocabularies onto one ladder. Daniel's real question is how you pick a rung.



Herman

And it comes down to a handful of questions, none of which is which vendor you like. First, tenant count, and before that, who the tenant even is. Pinecone has a case study with a company called Shortwave, and the detail that matters is that their tenant is the individual user, not the company. Strict separation between users. The moment your tenant unit is a human being rather than an organization, your cardinality explodes, and the isolation level follows from the cardinality, not from the database. A million-user product can't do schema-per-user. It probably can't even do namespace-per-user past a point, and then you're into Pinecone's mapping-table pattern, resolving the tenant to an index first, then querying the namespace inside it.



Corn

So the first design decision isn't on the database at all. It's at the whiteboard, deciding what counts as a tenant.



Herman

Corn, you'd be amazed how often that decision is made implicitly, by whoever wrote the first insert statement, and never revisited. Second question, compliance. Milvus says it plainly, choose database, collection, or partition-level if you have strict physical isolation requirements. pgvector, schema-per-tenant for strict compliance separation. Regulated customers, that's level two or above, non-negotiable. Third, schema heterogeneity. Different embedding models or different dimensions per tenant forces you to collection-level or higher. One shared index has one vector space, full stop. You cannot give tenant B different dimensions inside tenant A's index.



Corn

Fourth is the interesting one, because it pushes the other direction.



Herman

Cross-tenant search. The one product requirement that pulls you down the ladder instead of up it. Milvus is explicit, only partition-level and partition-key-level tenancy support cross-tenant queries. Pinecone says prefer namespaces unless you require the ability to query across tenants. If your product needs a superuser searching everyone's data, that single requirement can cost you two rungs of isolation.



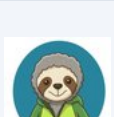
Corn

And when is that legitimate? Pinecone's own example.



Herman

Their federated search example is an internal knowledge-base chatbot federating a hundred and fifty different SaaS products, where an admin searches everything. Internal tooling, elevated user, all tenants belong to one organization anyway. That's the legitimate case, and notice it's not really multitenancy. It's one tenant with a hundred and fifty sources. When teams think they need cross-tenant search for end users, what they usually need is a shared collection everyone reads, plus per-tenant private ones.



Corn

A public library with locked offices upstairs.



Herman

That, and it composes cleanly. The remaining decision inputs are quieter. Tenant size skew, the pgvector five-million-rows threshold for partition-per-tenant. Cost model, the one RU versus a hundred RU arithmetic, where isolation is usually cheaper because you scan less. Offboarding, namespace deletion is instant, metadata tenancy means prefix listing and batch deletes. And hot/cold, which I think is the hidden operational axis of the whole topic. Weaviate's ACTIVE, INACTIVE, OFFLOADED controller and Milvus's per-tenant load and release are solving the identical problem. Most tenants are idle. Milvus's own table quietly notes that partition-key tenancy doesn't support hot and cold handling. Buried in a footnote, and it's the reason the most scalable rung costs more than it looks.



Corn

Here's the thing I keep landing on, Herman. All four vendors are telling you the same secret. Pinecone calls the shared-index pattern an anti-pattern. Weaviate calls the alternatives workarounds that often lead to data leakage. Chroma labels its own ladder naive and not suited for production. The pgvector guide says the shared table is where the reconciliation problem is sharpest.



Herman

Four independent vendors, no coordination, all converging on don't do the cheap thing. When competitors agree, believe them.



Corn

The cheap option is a hundred times more expensive and leaks data. It's not even cheap.



Herman

It's cheap on day one and expensive forever, which is the standard shape of a bad architecture decision. What's your instinct, though? You've leaned pro-isolation this whole episode, and you usually defend the boring shared option on the grounds that it's one thing to operate.



Corn

I do, normally. One table, one vacuum, one thing to back up. But this is the rare case where the boring option is the fast one too. My instinct is to say, default to level three, namespace per tenant, and accept the awkwardness of cross-tenant search rather than accept the leak. The empty result set is the right error message.



Herman

A typo in a namespace returns nothing. A typo in a metadata filter returns someone else's customer.



Hilbert

Corn, do you remember that query the neighbour ran past me last month? Fella in the flat below, couldn't get his tenant data back, kept saying the thing was broken. It wasn't broken. He'd spelled the namespace wrong. One letter. The query came back empty and he spent two evenings on it. I told him an empty result is the database doing its job, and he didn't care for that much.



Herman

I imagine he didn't.



Hilbert

I've still got the sticky note with the misspelled namespace on it, on the bottom of the monitor here, as a reminder. I'm a Postgres man myself, never used one of these vector things. Read enough of the documentation to be dangerous, though. My view, it's schemas with better marketing.



Corn

You're not entirely wrong.



Herman

He's not entirely right either. A schema doesn't come with its own ANN graph per tenant.



Hilbert

Give them time. They'll put a graph in anything now. I'd take the empty result over the wrong rows every day of the week, that's all I'll say. Herman, you've been up and down that ladder for twenty minutes, you never once said what happens to the fella who spells his filter wrong.



Herman

He gets the wrong customer's documents, and no error.



Hilbert

There it is, then. Stick with the typo you can see.



Herman

The sticky note stays.



Corn

Where that leaves me is a gap. There's no cross-vendor benchmark of any of this. Nobody has run the same workload against Pinecone namespaces and Weaviate tenants and Milvus partitions and pgvector RLS and measured recall or latency on the same data. Every claim of the form our isolation is nearly free is a vendor claim. The one quantified recall penalty in the whole field is Postgres-specific, the ef_search and probes numbers.



Herman

And Chroma's production story is thinner than it looks. Their official multi-tenancy guide page doesn't resolve, and the substantive guidance lives in the community cookbook, the one that disclaims production suitability. So the honest closing image is the ladder itself. Five rungs, one question at each. Does the index know the tenant?



Corn

The vocabulary will keep churning. Somebody will invent a sixth word next year, probably a compound of two of the existing five. The rungs won't move.



Herman

They haven't in Postgres's case for decades. The one thing I'd push back on you about, before we close it. You said level three is the default. I'd say the default is actually level three or level two depending on tenant count, and the real default question is who is the tenant, which most teams answer by accident.



Corn

Accepted, with the opinion intact. The misconception for the road, though, since people ask us for one. The common belief is that metadata filtering is the simple, cheap default for multi-tenancy and isolation is the expensive upgrade path.



Herman

And it's wrong in both directions at once. It's not cheap, the hundred-RU example says it can be a hundred times more expensive, and it's not safe, it's the only rung where a missing filter is a data breach instead of an empty result. Every vendor in this space has effectively said so in their own documentation.



Corn

If you take nothing else from the episode, find out which rung you're actually standing on. You may be one typo from finding out the hard way.



Herman

Thanks as always to Hilbert Flumingtop at the desk, producing the show out of whatever the sticky note says.



Corn

If you want more of this, try episode twelve fifteen, The Vector DB Hangover; episode nine, Benchmarking Custom ASR Tools - Beyond The WER; and episode twenty, Architectural AI. This has been My Weird Prompts. If this one earned its twenty-five minutes, the review that helps most is the honest one, wherever you listen.



Herman

See you tomorrow, Corn.



Corn

See you tomorrow.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **Pinecone docs, *Indexing overview* (namespaces, limits, metadata), retrieved 2026-11-10. [primary]**
   https://docs.pinecone.io/guides/index-data/indexing-overview

2. **Pinecone docs, *Implement multitenancy* (1 RU vs 100 RU, offboarding), retrieved 2026-11-10. [primary]**
   https://docs.pinecone.io/guides/index-data/implement-multitenancy

3. **Pinecone docs, *Design for multitenancy* (anti-pattern, `$in` limit), retrieved 2026-11-10. [primary]**
   https://docs.pinecone.io/guides/index-data/design-for-multitenancy

4. **Pinecone Learn, *Multi-Tenancy in Vector Databases* (Shortwave, mapping table, ID-prefix pattern), retrieved 2026-11-10. [primary]**
   https://www.pinecone.io/learn/series/vector-databases-in-production-for-busy-engineers/vector-database-multi-tenancy/

5. **Weaviate blog, *Rethinking Vector Search at Scale: Weaviate's Native, Efficient and Optimized Multi-Tenancy* (shard-per-tenant, Tenant Controller, buckets), retrieved 2026-11-10. [primary]**
   https://weaviate.io/blog/weaviate-multi-tenancy-architecture-explained

6. **Weaviate docs, *Data structure* (collections, per-collection vector space, named vectors), retrieved 2026-11-10. [primary]**
   https://weaviate.io/developers/weaviate/concepts/data

7. **Milvus docs, *Implement Multi-tenancy* (four levels, full comparison table, decision guidance), retrieved 2026-11-10. [primary]**
   https://milvus.io/docs/multi_tenancy.md

8. **Chroma docs, *Architecture Overview* (tenants, databases, collections), retrieved 2026-11-10. [primary]**

   https://docs.trychroma.com/reference/architecture/overview

9. **Chroma Cookbook, *Naive Multi-tenancy Strategies* (User-Per-Doc/Collection/Database/Tenant, drawbacks), retrieved 2026-11-10.**

   https://cookbook.chromadb.dev/strategies/multi-tenancy/naive-multi-tenancy/

10. **Chroma Cookbook, *Concepts* (tenant/database/collection mental model, SQLite single-node storage), retrieved 2026-11-10.**

    https://cookbook.chromadb.dev/core/concepts/

11. **returned 404 on 2026-11-10; noted as a negative finding.**

    https://docs.trychroma.com/guides/build/multi-tenancy

12. ***Security Boundaries for Vector Data* (three pgvector models, GUC tuning, RLS hardening), retrieved 2026-11-10.**

    https://www.index-management.org/pgvector-architecture-vector-fundamentals/security-boundaries-for-vector-data/

13. **Render, *Multi-Tenant RAG With PostgreSQL and pgvector* (two-table schema, composite FK), retrieved 2026-11-10.**

    https://render.com/articles/multi-tenant-rag-postgresql-pgvector

14. **AWS Database Blog, *Self-managed multi-tenant vector search with Amazon Aurora PostgreSQL* (RLS policy via `current_setting`), retrieved 2026-11-10.**

    https://aws.amazon.com/blogs/database/self-managed-multi-tenant-vector-search-with-amazon-aurora-postgresql/

15. ***Multi-Tenant RAG: How to Isolate Customer Data in a Vector Database* (cross-pattern comparison table), retrieved 2026-11-10.**

    https://dreaming.press/posts/multi-tenant-rag.html

16. ***Vector Database Security Comparison: Pinecone vs. Weaviate vs. Milvus* (primary isolation unit per vendor; contains an outdated 10,000-namespace figure), retrieved 2026-11-10.**

    https://www.decryptiondigest.com/blog/rag-vector-database-security-comparison-pinecone-weaviate-milvus