POSETTE:
An Event for Postgres 2026

From Queries to Agents: The Next Era of Data Retrieval on PostgreSQL

Abe Omorogbe

Hosted by

Microsoft | June 16-18, 2026 • posetteconf.com

ABE OMOROGBE

- Product Manager at Postgres AI team at Microsoft.
- My previous experiences includes working Azure AI and Databricks
- POSETTE Last year: "Building Intelligent Apps with Graph-Based RAG" consider this the sequel

· @_aiabe

· linkedin.com/in/abeomor

· github.com/AbeOmor

Has an AI agent ever done something you didn’t want it to do?

AI • CODING

An AI agent destroyed this coder's entire database. He's not the only one with a horror story

AI NEWS TECH

Amazon blames human employees for an AI coding agent's mistake

AI vs AI: Agent hacked McKinsey's chatbot and gained full read-write access in just two hours

Why 90% Accuracy in Text-to-SQL is 100% Useless

I gave an AI agent access to my **product database** and asked it.
"What's our best bookshelf?"

It confidently recommended an **insufficient answers**.

The model was great. The retrieval was the problem.

So... I'd love to show you how we can address this problem,
step by step, with Postgres.

And build a trustworthy AI agent for data retrieval

Agenda – Build AI Agent for eCommerce Retrieval

1. RAG → Agents
2. MCP
3. Common Issues
4. Context correction and Blended retrieval
5. What's Next

RAG & Data Retrieval

RAG stands for **Retrieval-Augmented Generation**. It's a technique used to enhance the output of large language models (LLMs) by integrating external knowledge sources.

It works! But it is a **single-turn, retrieval-only system**

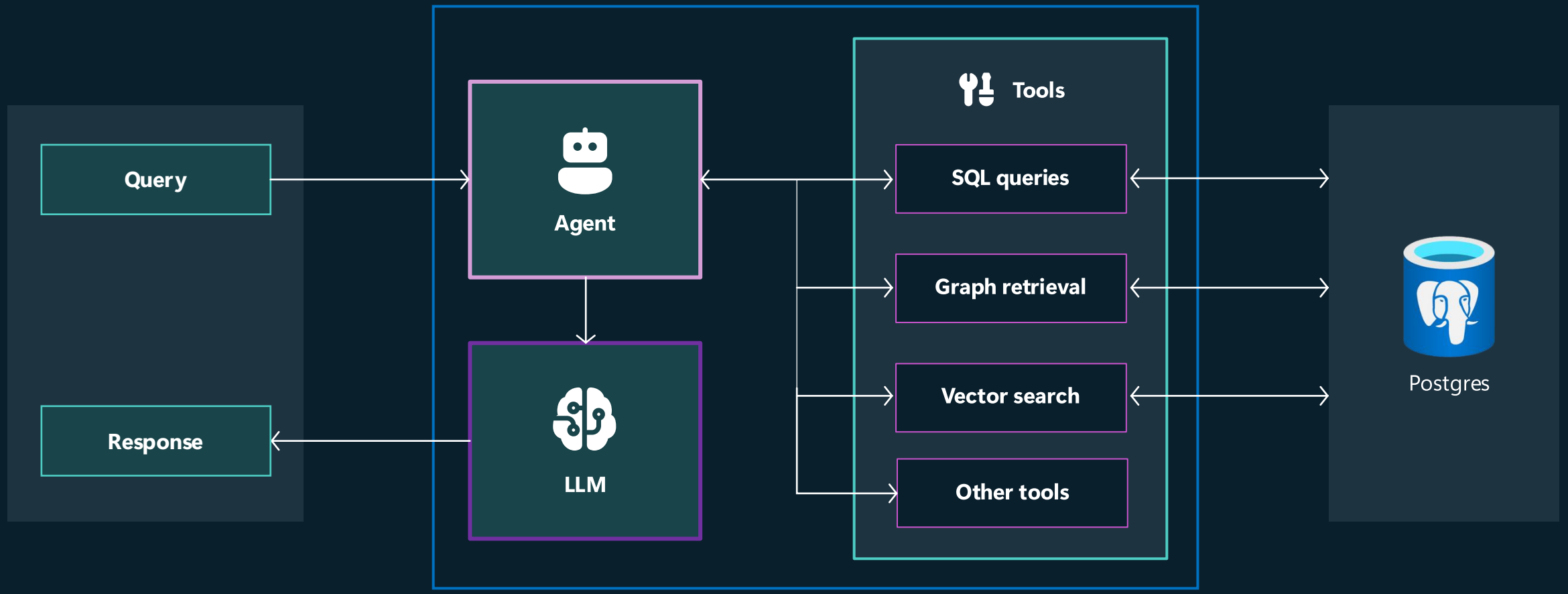
What users actually want

> *"I'm redecorating my living room. Find me a stylish bookshelf under $100 that pairs well with the Furinno end table, and check if any similar storage furniture is a bargain."*

1. Semantic Search
2. Price Filter
3. Product Graph
4. Bargain Check

RAG gives you **1 step**. **Agents** can chain all 4.

The Agent Pattern



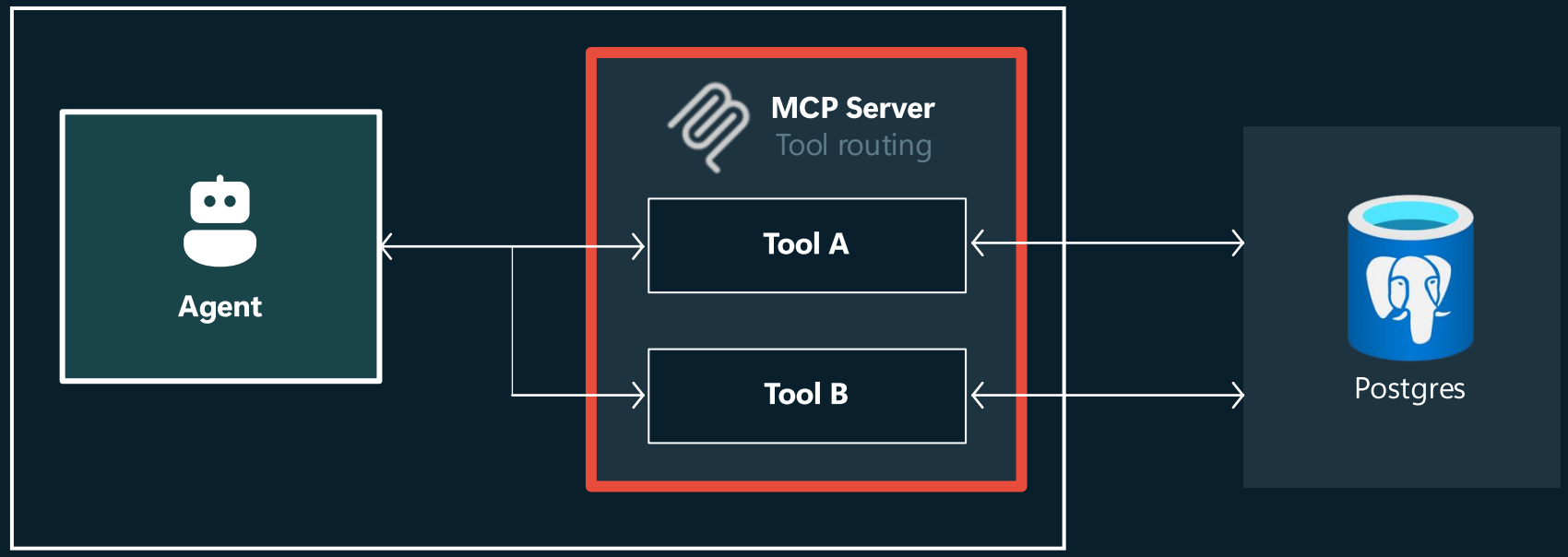
The tool layer is where everything gets interesting and where everything breaks.

So.... how do agents actually talk to Postgres?

Model Context Protocol (MCP)

The "USB-C for AI" a standardized plug between agents and tools

- Open protocol for connecting AI agents to data sources
- Growing ecosystem, MCP servers for Postgres already exist
- Agents call tools instead of generating raw SQL



This works!

But...

MCP gives structure, but the agent still has to generate **good queries**.

Let's stress-test our shopping agent

Common Issues

> *"I'm redecorating my living room. Find me a stylish bookshelf under $100 that pairs well with the Furinno end table, and check if any similar storage furniture is a bargain."*

This requires:

`Vector – product semantics` `SQL – price & stock` `Graph – relationships` `Agent – multi-step`

Failure Mode 1 — Schema Hallucination

Agent generates SQL referencing tables that don't exist

SELECT query

```
SELECT * FROM product_recs -- Table doesn't exist!
WHERE category = ‘storage’ AND
compatibility_score > 0.8; -- Column doesn't exist!
```

```
ERROR: relation "product_recs" does not exist
```

Failure Mode 2 — Semantic Misunderstanding

Syntactically valid SQL that's semantically wrong

SELECT query

```
SELECT * FROM products
WHERE name LIKE ‘%stylish bookshelf%’;
-- Misses: ”modern", ”sleek", ”chic"
```

Results: 8 rows (should be 430)

Failure Mode 3 — Missing Guardrails

Generated SQL may expose internal data.

Dangerous queries

```
SELECT wholesale_price, supplier_id FROM products
WHERE category = ‘storage’
-- Customers shouldn’t see the wholesale price
```

No guardrails = **no production readiness**

So how do we solve this?

The real problem isn't SQL generation

It's Context

Agents fail because they don't have enough context about your specific database

Context Correction — Four Layers

Schema Context
Detailed table/column descriptions, not just names

Semantic Context
What values mean — "stylish bookshelf" = "modern", "sleek" and "chic"

Relationship Context
How tables connect — products → reviews → categories

Safety and Governance Context
Allowed queries, row limits, hidden columns

Ecommerce Demo Dataset

Ecommerce furniture catalog — the data our agent will retrieve from

products

id	name	brand	price	embedding
1042	Pasir End Table, Walnut	Furinno	$34.99	[0.121, 0.123…]
1856	Turn-N-Tube Bookcase	Furinno	$54.81	[0.121, 0.123…]
2391	3-Shelf Open Bookcase	Sauder	$89.99	[0.121, 0.123…]
4127	Whisper Wave Space Heater	Lasko	$79.95	[0.121, 0.123…]
5829	Tilt Floor Lamp, Brass	Brightech	$112.00	[0.121, 0.123…]

```
-- category column (JSONB), shown for id=1042:
["Home & Kitchen","Furniture","Tables","End Tables"]
```

100K+ products · 156 brands · 24K relationships · **1536-d embeddings with pgvector**

Adding context directly in Postgres

context_setup.sql

```sql
-- Schema context via COMMENT ON
COMMENT ON COLUMN products.category IS
'JSONB array of category path strings, ordered general → specific. '
'Example: ["Home & Kitchen", "Storage", "Bookcase"]. '
'Last element is the most specific category.';

-- Semantic dictionary table
CREATE TABLE semantic_dictionary (
  term       text,    -- "stylish bookshelf"
  canonical  text,    -- "modern bookshelf storage shelf sleek chic design"
  context    text,    -- "search"
  notes      text     -- "Expand bookshelf style terms"
);
```

Agent gets context **BEFORE** generating SQL

Without Context vs. With Context

query > ‘Find me a stylish bookcase’

Before

```
WHERE name LIKE ‘%stylish bookcase%’;
```

8 results

After

```
vector search for 'modern bookcase
storage shelf sleek chic design'
```

430 results

ranked by relevance

Better! But we're still doing
one retrieval mode at a time...

What if the agent could use **ALL of Postgres's retrieval modes** together?

Azure HorizonDB

PostgreSQL power. Infinite possibilities.

Built for Business

Comprehensive security, high availability, workload scalability, and disaster recovery for enterprise applications

Engineered for Developers

Industry leading AI features, advanced vector indexing , first-class VS Code IDE , mirroring to Fabric for real-time analytics

`vector` `pg_fts` `pg_diskann` `age*` `azure_ai`

One Database. Four Retrieval Modes.

MODE	WHAT IT DOES	POSTGRES FEATURE
Hybrid Search	Exact terms + semantic meaning	`pg_fts + diskann + vector + RRF`
Relational SQL	Structured filters	`WHERE price < 200`
Graph Traversal	Product relationships	`Apache AGE`
Re-ranking	Final relevance pass	`azure_ai.rank()`

No other database gives you all of this natively.

HorizonDB Hybrid Search

BM25 — Exact Matches (pg_fts)

Production-grade **BM25 full-text search**, implemented natively inside PostgreSQL. Similar algorithm found in Elasticsearch.

Vector Search (pg_vector)

Semantic similarity search using **vector embeddings** stored and queried directly in PostgreSQL.

DiskANN Vector Index (pg_diskann)

Microsoft research's DiskANN, integrated natively into PostgreSQL for fast, accurate, and memory-efficient vector search.

Reciprocal Rank Fusion (RRF)

A ranking technique that **combines BM25 and vector results** into a single, unified relevance score.

BM25 + Vector + DiskANN + RRF — One Query

hybrid_search.sql

```
-- BM25 keyword search via pg_fts
WITH bm25_results AS (
    SELECT id, name, price, ROW_NUMBER() OVER () AS bm25_rank
    FROM products
    WHERE pgfts.fts_query('<query>','idx_products_fts')),

-- Semantic search via pgvector (indexed with DiskANN)
vector_results AS (
    SELECT id, name, price,
           ROW_NUMBER() OVER (ORDER BY embedding <=>
               azure_ai.create_embeddings('text-embedding-3-small',
                   '<query>')::vector) AS vec_rank)

-- Reciprocal Rank Fusion
SELECT COALESCE(b.id, v.id), COALESCE(b.name, v.name),
       (1.0/(60+COALESCE(b.bm25_rank,999)))+(1.0/(60+COALESCE(v.vec_rank,999))) AS rrf
FROM bm25_results b FULL OUTER JOIN vector_results v ON b.id=v.id
ORDER BY rrf DESC LIMIT 10;
```

Full Text Search

Vector Search & Azure AI

RRF

Graph in Postgres with Apache AGE

Quick refresher — model and query relationships using Cypher alongside SQL

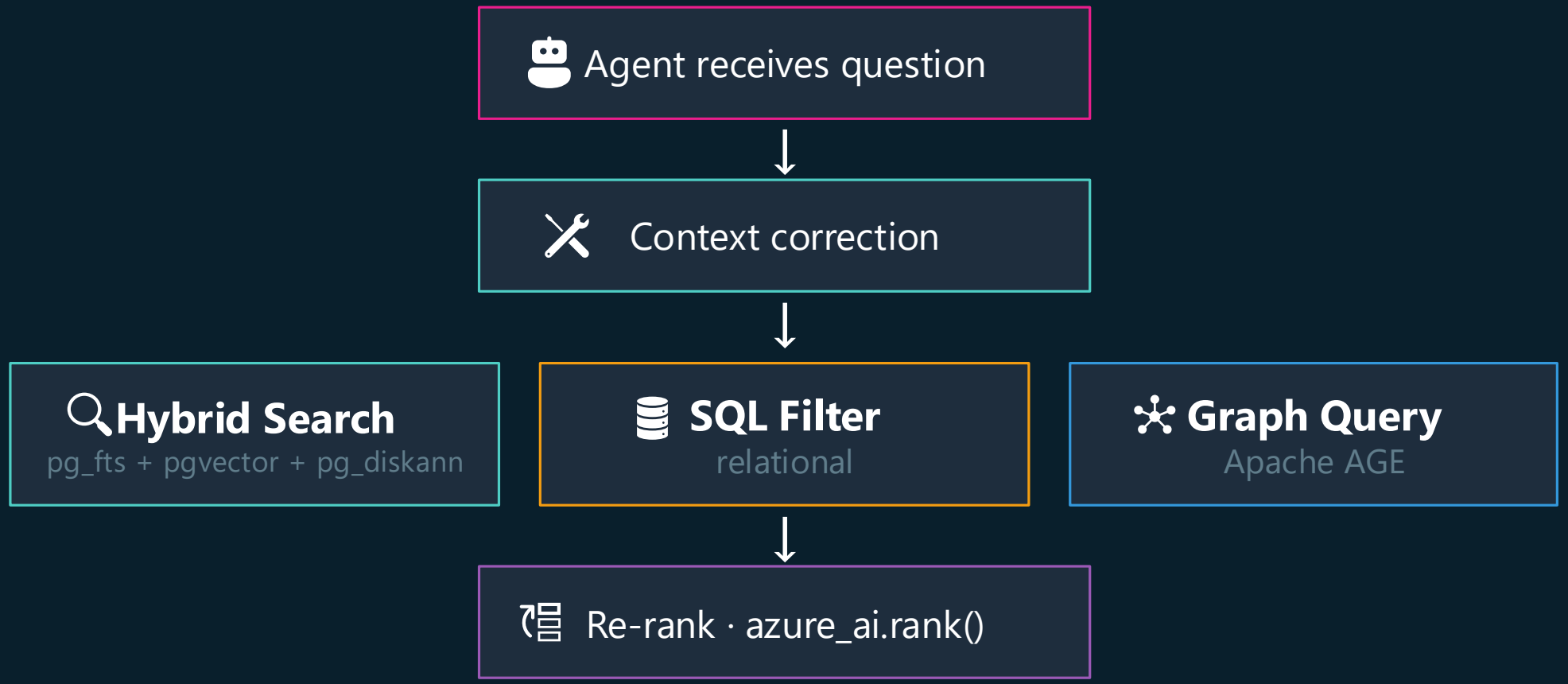
`Frequently bought together` `Customers also viewed` `Brand → Product → Category`

graph_query.sql

```
SELECT * FROM cypher('product_graph', $$
    MATCH (p:Product {name: 'Furinno end table'})
          -[:BOUGHT_WITH]->(rec:Product)
    RETURN rec.name, rec.price
$$) AS (name text, price numeric);
```

Graph queries, inside Postgres, **zero extra infrastructure**.

Blended Retrieval Architecture



DEMO

E-Commerce Product Copilot

Watch the agent solve the bookcase query end-to-end

Solving the Hard Query — Step by Step

> *"I'm redecorating my living room. Find me a stylish bookshelf under $100 that pairs well with the Furinno end table, and check if any similar storage furniture is a bargain."*

1. **Context Correction** — Correct the word "stylish bookshelf" to "modern bookshelf sleek chic design"
2. **Hybrid Search** — BM25 exact + vector semantic → RRF fusion
3. **SQL Filter** — WHERE price < 100 AND in_stock = true
4. **Graph Query** — BOUGHT_WITH edges from *Furinno end table*
5. **Re-rank** — azure_ai.rank() final scoring

Agenda – Build AI Agent for eCommerce Retrieval

1. RAG → Agents
2. MCP
3. Common Issues
4. Context correction and Blended retrieval
5. What's Next

Azure HorizonDB

Everything I just showed you ships on Azure HorizonDB.

All Extensions Pre-configured
pg_fts, pgvector, DiskANN, AGE, azure_ai

AI Model Management
Provision and manage models from your database

Foundry MCP Connector
Agent-to-database connectivity, out of the box

Enterprise Ready
HA, security, scale up/down, managed infra

No external services. Everything in one database.

What's Next in HorizonDB

2024
Queries
Basic SQL

→

2025
RAG
Vector + Graph + DiskANN

→

2026
Agents
Blended retrieval + MCP

→

Next
Unified Retrieval
Managed in HorizonDB

```
SELECT * FROM ai.search('stylish bookcase',top_k => 5);
```

From Queries to Agents — and beyond

Try It Yourself

aka.ms/posette-query-agents-2026

GitHub repo with MCP server config,
context correction setup, and blended retrieval code

More Resources

Azure HorizonDB

- aka.ms/AzureHorizonDB

Postgres AI Features and Solution Accelerators

- aka.ms/agentic-advisor
- aka.ms/agentic-shop
- aka.ms/horizondb_pg_durable
- aka.ms/horizondb-diskann-filtering
- aka.ms/horizondb-ai-pipelines
- aka.ms/horizondb-pg-fts

Read the Microsoft Blog for Postgres

- aka.ms/microsoft-blog-postgres

Thank You!

Let's build the future of agent retrieval on Postgres — together.

X · @_aiabe
linkedin.com/in/abeomor
github.com/AbeOmor

Abe Omorogbe · Product Manager · Postgres AI · Microsoft