Microsoft | PGConf India, 2025

All the Postgres Things at Microsoft

Sujit Kuruvilla
Director of Engineering
Azure Database for PostgreSQL at Microsoft

Community

Code

Cloud

Community

Microsoft

Talks by our Microsoft team

Training: Developing RAG Apps with Azure Database for PostgreSQL & GraphRAG

Varun Dhawan

Wed 5 Mar | 9:00

Hacking Postgres Executor For Performance

Amit Langote

Thu 6 Mar | 11:30

Graph databases, PostgreSQL and SQL/PGQ

Ashutosh Bapat

Thu 6 Mar | 14:00

Unleashing the Power of Azure Database for PostgreSQL Flexible Server

Shriram Muthukrishnan

Thu 6 Mar | 14:00

Keynote: All the Postgres Things at Microsoft

Sujit Kuruvilla

Thu 6 Mar | 16:45

Using Postgres to locate the best coffee near you

Varun Dhawan

Fri 7 Mar | 10:45

Postgres: ServerLESS is more?

Nikhil Sontakke

Fri 7 Mar | 11:30

Beginner's Guide to Partitioning vs. Sharding in Postgres

Claire Giordano

Fri 7 Mar | 14:45

Save the date
June 10-12, 2025

POSETTE: An Event for Postgres

2025 Now in its 4th year!

A free & virtual developer event

Check out the schedule → PosetteConf.com

Microsoft

Have you listened to Talking Postgres?

Ep24: Robert Haas
Ep23: Daniel Gustafsson
Ep22: Affan Dar
Ep21: Andrew Atkinson
Ep20: Tom Lane
Ep19: Melanie Plageman
Ep18: David Rowley
Ep17: Pino de Candia
Ep16: Teresa Giacomini & Aaron Wislang
Ep15: Michael Christofides
Ep14: Chris Ellis
Ep13: Arda Aytekin
Ep12: Derk van Veen
Ep11: Jelte Fennema-Nio & Marco Slot
Ep10: Lukas Fittl & Rob Treat
Ep09: Dimitri Fontaine & Vik Fearing
Ep08: Andres Freund & Heikki Linnakangas
Ep07: Paul Ramsey & Regina Obe
Ep06: Chelsea Dole & Floor Drees
Ep05: Grant Fritchey & Ryan Booz
Ep04: Melanie Plageman & Thomas Munro
Ep03: Álvaro Herrera & Boriss Mejías
Ep02: Abdullah Ustuner, Burak Yucesoy, Melanie Plageman, Samay Sharma
Ep01: Simon Willison & Marco Slot

Free Socks @ Microsoft booth!

Code

Postgres 17 Contributions

412

Postgres 17 commits authored or co-authored by Microsoft engineers

Async IO - read stream

I/O Combining

UNION & IS [NOT] NULL query planner

VACUUM

Libpq performance and cancellation

Partitioned tables performance

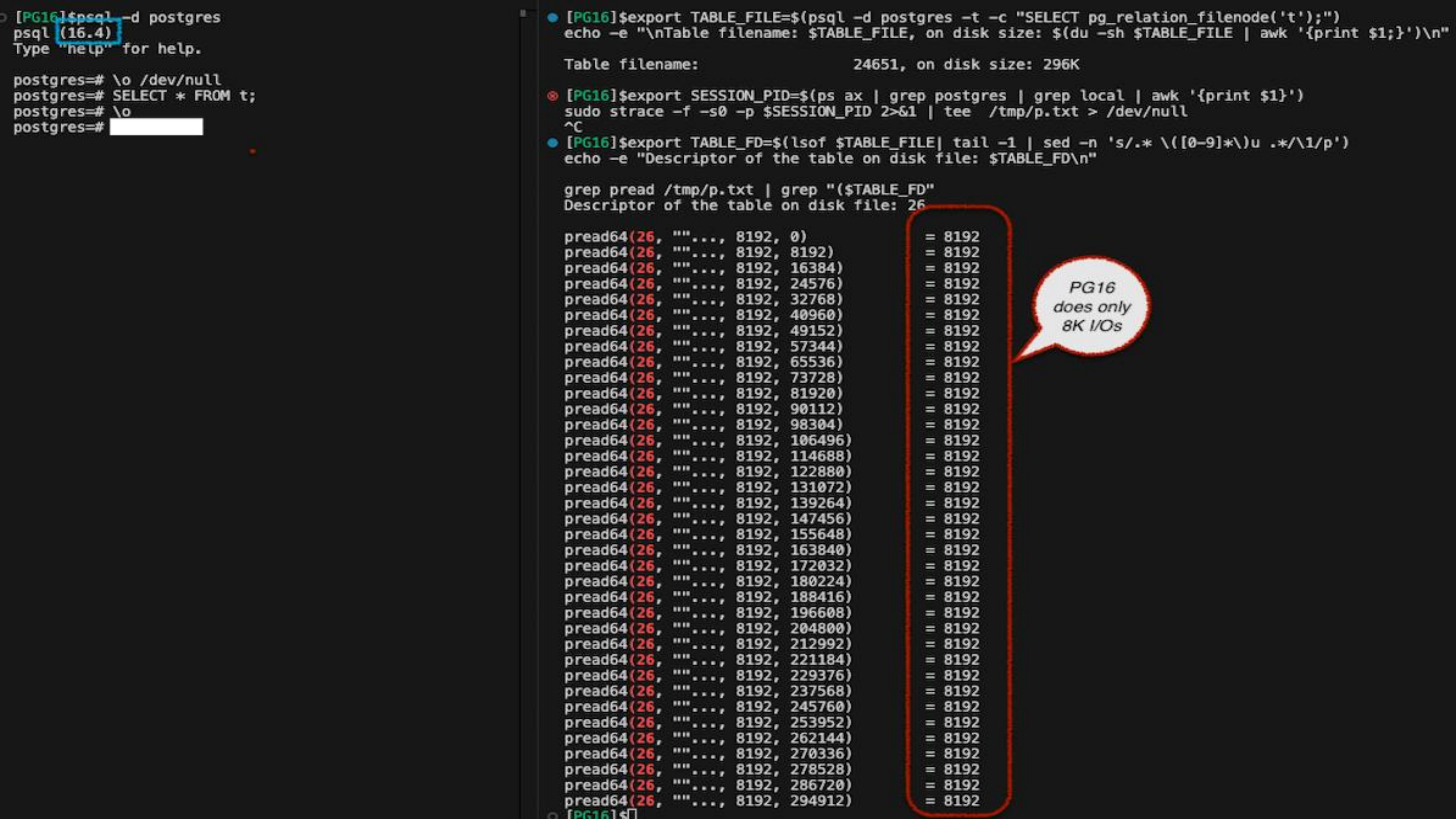
Memory performance

PG upgrade optimization

Developer tool

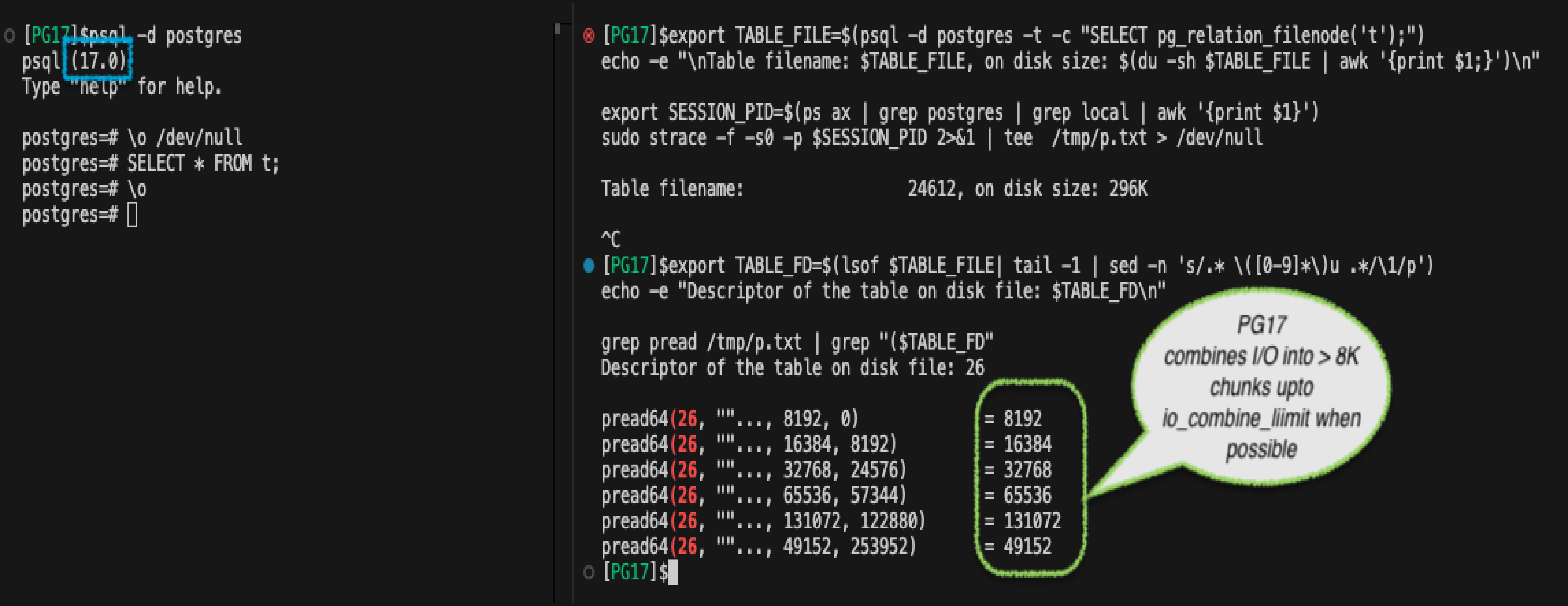
https://aka.ms/blog-pg-at-microsoft

I/O Combining



```
[PG16]$psql -d postgres
psql (16.4)
Type "help" for help.

postgres=# \o /dev/null
postgres=# SELECT * FROM t;
postgres=# \o
postgres=#
```

```
[PG16]$export TABLE_FILE=$(psql -d postgres -t -c "SELECT pg_relation_filenode('t');")
echo -e "\nTable filename: $TABLE_FILE, on disk size: $(du -sh $TABLE_FILE | awk '{print $1;}')\n"

Table filename:                    24651, on disk size: 296K

[PG16]$export SESSION_PID=$(ps ax | grep postgres | grep local | awk '{print $1}')
sudo strace -f -s0 -p $SESSION_PID 2>&1 | tee  /tmp/p.txt > /dev/null
^C
[PG16]$export TABLE_FD=$(lsof $TABLE_FILE| tail -1 | sed -n 's/.* \([0-9]*\)u .*/\1/p')
echo -e "Descriptor of the table on disk file: $TABLE_FD\n"

grep pread /tmp/p.txt | grep "($TABLE_FD"
Descriptor of the table on disk file: 26

pread64(26, ""..., 8192, 0)          = 8192
pread64(26, ""..., 8192, 8192)       = 8192
pread64(26, ""..., 8192, 16384)      = 8192
pread64(26, ""..., 8192, 24576)      = 8192
pread64(26, ""..., 8192, 32768)      = 8192
pread64(26, ""..., 8192, 40960)      = 8192
pread64(26, ""..., 8192, 49152)      = 8192
pread64(26, ""..., 8192, 57344)      = 8192
pread64(26, ""..., 8192, 65536)      = 8192
pread64(26, ""..., 8192, 73728)      = 8192
pread64(26, ""..., 8192, 81920)      = 8192
pread64(26, ""..., 8192, 90112)      = 8192
pread64(26, ""..., 8192, 98304)      = 8192
pread64(26, ""..., 8192, 106496)     = 8192
pread64(26, ""..., 8192, 114688)     = 8192
pread64(26, ""..., 8192, 122880)     = 8192
pread64(26, ""..., 8192, 131072)     = 8192
pread64(26, ""..., 8192, 139264)     = 8192
pread64(26, ""..., 8192, 147456)     = 8192
pread64(26, ""..., 8192, 155648)     = 8192
pread64(26, ""..., 8192, 163840)     = 8192
pread64(26, ""..., 8192, 172032)     = 8192
pread64(26, ""..., 8192, 180224)     = 8192
pread64(26, ""..., 8192, 188416)     = 8192
pread64(26, ""..., 8192, 196608)     = 8192
pread64(26, ""..., 8192, 204800)     = 8192
pread64(26, ""..., 8192, 212992)     = 8192
pread64(26, ""..., 8192, 221184)     = 8192
pread64(26, ""..., 8192, 229376)     = 8192
pread64(26, ""..., 8192, 237568)     = 8192
pread64(26, ""..., 8192, 245760)     = 8192
pread64(26, ""..., 8192, 253952)     = 8192
pread64(26, ""..., 8192, 262144)     = 8192
pread64(26, ""..., 8192, 270336)     = 8192
pread64(26, ""..., 8192, 278528)     = 8192
pread64(26, ""..., 8192, 286720)     = 8192
pread64(26, ""..., 8192, 294912)     = 8192
[PG16]$
```

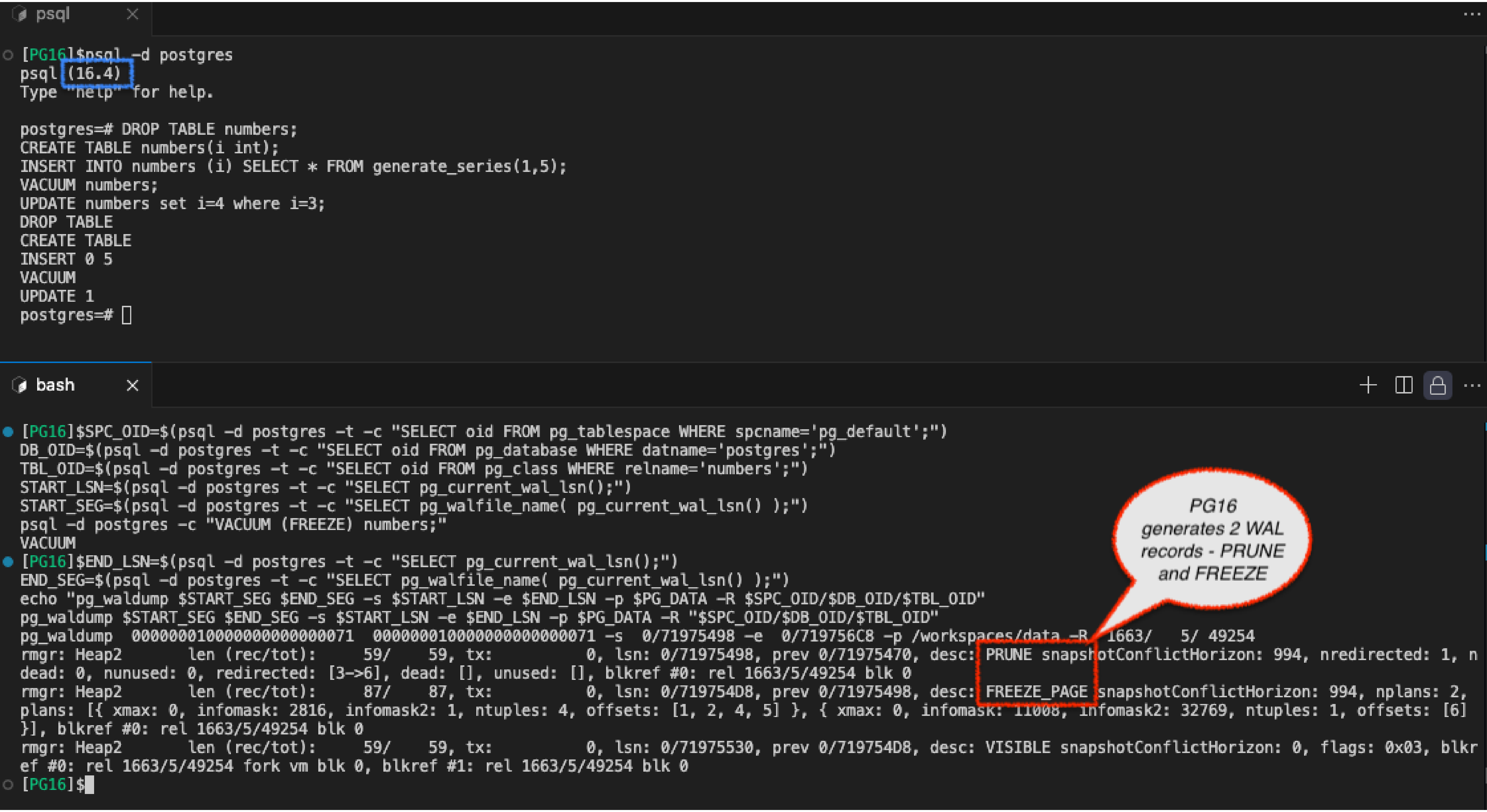


```
[PG17]$psql -d postgres
psql (17.0)
Type "help" for help.

postgres=# \o /dev/null
postgres=# SELECT * FROM t;
postgres=# \o
postgres=#
```

```
[PG17]$export TABLE_FILE=$(psql -d postgres -t -c "SELECT pg_relation_filenode('t');")
echo -e "\nTable filename: $TABLE_FILE, on disk size: $(du -sh $TABLE_FILE | awk '{print $1;}')\n"

export SESSION_PID=$(ps ax | grep postgres | grep local | awk '{print $1}')
sudo strace -f -s0 -p $SESSION_PID 2>&1 | tee  /tmp/p.txt > /dev/null

Table filename:                 24612, on disk size: 296K

^C
[PG17]$export TABLE_FD=$(lsof $TABLE_FILE| tail -1 | sed -n 's/.* \([0-9]*\)u .*/\1/p')
echo -e "Descriptor of the table on disk file: $TABLE_FD\n"

grep pread /tmp/p.txt | grep "($TABLE_FD"
Descriptor of the table on disk file: 26

pread64(26, ""..., 8192, 0)             = 8192
pread64(26, ""..., 16384, 8192)         = 16384
pread64(26, ""..., 32768, 24576)        = 32768
pread64(26, ""..., 65536, 57344)        = 65536
pread64(26, ""..., 131072, 122880)      = 131072
pread64(26, ""..., 49152, 253952)       = 49152
[PG17]$
```

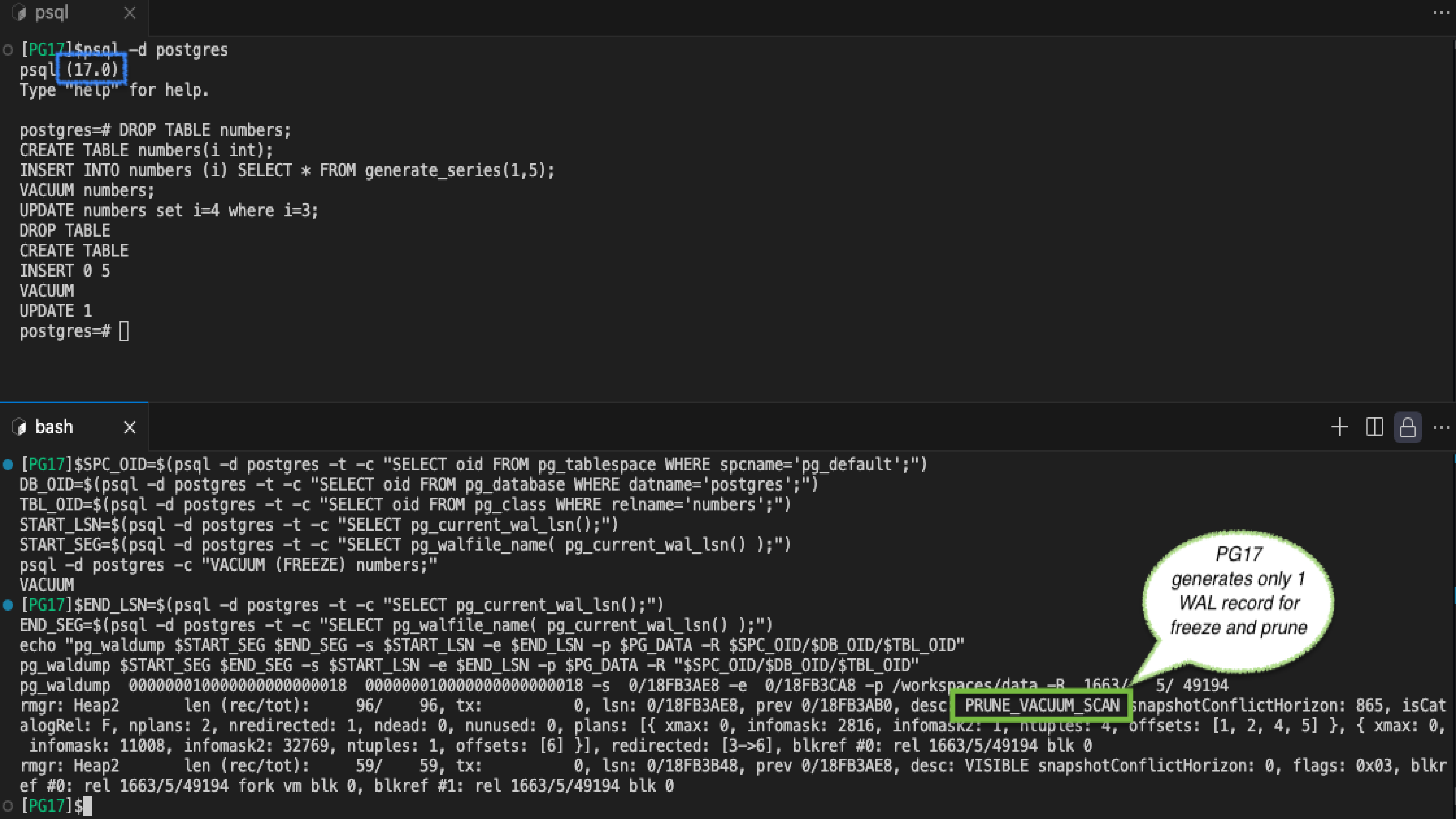
PG17 combines I/O into > 8K chunks upto io_combine_limit when possible

VACUUM
WAL volume reduction & performance improvements



```
[PG16]$psql -d postgres
psql (16.4)
Type "help" for help.

postgres=# DROP TABLE numbers;
CREATE TABLE numbers(i int);
INSERT INTO numbers (i) SELECT * FROM generate_series(1,5);
VACUUM numbers;
UPDATE numbers set i=4 where i=3;
DROP TABLE
CREATE TABLE
INSERT 0 5
VACUUM
UPDATE 1
postgres=# 
```

```
[PG16]$SPC_OID=$(psql -d postgres -t -c "SELECT oid FROM pg_tablespace WHERE spcname='pg_default';")
DB_OID=$(psql -d postgres -t -c "SELECT oid FROM pg_database WHERE datname='postgres';")
TBL_OID=$(psql -d postgres -t -c "SELECT oid FROM pg_class WHERE relname='numbers';")
START_LSN=$(psql -d postgres -t -c "SELECT pg_current_wal_lsn();")
START_SEG=$(psql -d postgres -t -c "SELECT pg_walfile_name( pg_current_wal_lsn() );")
psql -d postgres -c "VACUUM (FREEZE) numbers;"
VACUUM
[PG16]$END_LSN=$(psql -d postgres -t -c "SELECT pg_current_wal_lsn();")
END_SEG=$(psql -d postgres -t -c "SELECT pg_walfile_name( pg_current_wal_lsn() );")
echo "pg_waldump $START_SEG $END_SEG -s $START_LSN -e $END_LSN -p $PG_DATA -R $SPC_OID/$DB_OID/$TBL_OID"
pg_waldump $START_SEG $END_SEG -s $START_LSN -e $END_LSN -p $PG_DATA -R "$SPC_OID/$DB_OID/$TBL_OID"
pg_waldump  000000010000000000000071  000000010000000000000071 -s  0/71975498 -e  0/719756C8 -p /workspaces/data -R  1663/   5/ 49254
rmgr: Heap2       len (rec/tot):     59/    59, tx:          0, lsn: 0/71975498, prev 0/71975470, desc: PRUNE snapshotConflictHorizon: 994, nredirected: 1, n
dead: 0, nunused: 0, redirected: [3->6], dead: [], unused: [], blkref #0: rel 1663/5/49254 blk 0
rmgr: Heap2       len (rec/tot):     87/    87, tx:          0, lsn: 0/719754D8, prev 0/71975498, desc: FREEZE_PAGE snapshotConflictHorizon: 994, nplans: 2,
plans: [{ xmax: 0, infomask: 2816, infomask2: 1, ntuples: 4, offsets: [1, 2, 4, 5] }, { xmax: 0, infomask: 11008, infomask2: 32769, ntuples: 1, offsets: [6]
}], blkref #0: rel 1663/5/49254 blk 0
rmgr: Heap2       len (rec/tot):     59/    59, tx:          0, lsn: 0/71975530, prev 0/719754D8, desc: VISIBLE snapshotConflictHorizon: 0, flags: 0x03, blkr
ef #0: rel 1663/5/49254 fork vm blk 0, blkref #1: rel 1663/5/49254 blk 0
[PG16]$
```



```
[PG17]$psql -d postgres
psql (17.0)
Type "help" for help.

postgres=# DROP TABLE numbers;
CREATE TABLE numbers(i int);
INSERT INTO numbers (i) SELECT * FROM generate_series(1,5);
VACUUM numbers;
UPDATE numbers set i=4 where i=3;
DROP TABLE
CREATE TABLE
INSERT 0 5
VACUUM
UPDATE 1
postgres=# 
```

```
[PG17]$SPC_OID=$(psql -d postgres -t -c "SELECT oid FROM pg_tablespace WHERE spcname='pg_default';")
DB_OID=$(psql -d postgres -t -c "SELECT oid FROM pg_database WHERE datname='postgres';")
TBL_OID=$(psql -d postgres -t -c "SELECT oid FROM pg_class WHERE relname='numbers';")
START_LSN=$(psql -d postgres -t -c "SELECT pg_current_wal_lsn();")
START_SEG=$(psql -d postgres -t -c "SELECT pg_walfile_name( pg_current_wal_lsn() );")
psql -d postgres -c "VACUUM (FREEZE) numbers;"
VACUUM
[PG17]$END_LSN=$(psql -d postgres -t -c "SELECT pg_current_wal_lsn();")
END_SEG=$(psql -d postgres -t -c "SELECT pg_walfile_name( pg_current_wal_lsn() );")
echo "pg_waldump $START_SEG $END_SEG -s $START_LSN -e $END_LSN -p $PG_DATA -R $SPC_OID/$DB_OID/$TBL_OID"
pg_waldump $START_SEG $END_SEG -s $START_LSN -e $END_LSN -p $PG_DATA -R "$SPC_OID/$DB_OID/$TBL_OID"
pg_waldump  000000010000000000000018  000000010000000000000018 -s  0/18FB3AE8 -e  0/18FB3CA8 -p /workspaces/data -R  1663/   5/ 49194
rmgr: Heap2       len (rec/tot):     96/    96, tx:          0, lsn: 0/18FB3AE8, prev 0/18FB3AB0, desc: PRUNE_VACUUM_SCAN snapshotConflictHorizon: 865, isCat
alogRel: F, nplans: 2, nredirected: 1, ndead: 0, nunused: 0, plans: [{ xmax: 0, infomask: 2816, infomask2: 1, ntuples: 4, offsets: [1, 2, 4, 5] }, { xmax: 0,
 infomask: 11008, infomask2: 32769, ntuples: 1, offsets: [6] }], redirected: [3->6], blkref #0: rel 1663/5/49194 blk 0
rmgr: Heap2       len (rec/tot):     59/    59, tx:          0, lsn: 0/18FB3B48, prev 0/18FB3AE8, desc: VISIBLE snapshotConflictHorizon: 0, flags: 0x03, blkr
ef #0: rel 1663/5/49194 fork vm blk 0, blkref #1: rel 1663/5/49194 blk 0
[PG17]$
```

Sneak peek into PG18 and upcoming changes

- Multithreading
- Async IO
- Security improvements – OAuth support
- Partitioning improvements.
- Query planner and executor optimizations.
- VACUUM enhancements.
- Memory – plasticity and observability
- Large SKU performance analysis and improvements.
- Quality improvements – test coverage, CI/CD improvements.

Cloud

Azure Top Level Investments

Enterprise

Developers

Azure Database for PostgreSQL

2024

In Review

April

Geo-Disaster Recovery - GA

Virtual Endpoints

Built-in PgBouncer version bump

On-demand Extension Update

Azure Database for PostgreSQL

2024

In Review

May

Automatic Index Recommendations

Azure Advisor Recommendations

AI Extensions

- **Azure_ai – GA**
- **Azure_local_ai – Preview**

Independent IOPS scaling

Azure Database for PostgreSQL

2024

In Review

June

PG 16 Major Version Upgrade - GA

Pgvector 0.7.0

New Azure Regions
- China North 2
- China East 2

Long Term Backup for CMK servers

Azure Database for PostgreSQL

2024

In Review

July

System Managed Identity - GA

Recovery improvements for PITR

New monitoring metric

- Database Size
- Transaction per sec (TPS)

Index tuning enhancements

Azure Database for PostgreSQL

2024

In Review

Aug

Reserved Pricing for Intel & AMD V5

Latest postgres minor versions

New Extensions

- postgres_protobuf
- postgresql_anonymizer

Support for TimescaleDB extension in migration service

Azure Database for PostgreSQL

2024

In Review

Sep

Postgres 17 - Preview

DiskANN Vector Index - Preview

Fabric Mirroring - Private Preview

Auto Migrations – Single to Flexible server

Azure Database for PostgreSQL

2024

In Review

Oct

Elastic Clusters – Preview

AI Updates
- Semantic Ranking Solution
- GraphRAG Solution

Index Tuning – GA

On-demand backup – Preview

Oracle_FDW extension – Preview

Azure Database for PostgreSQL

2025

In Review

Jan

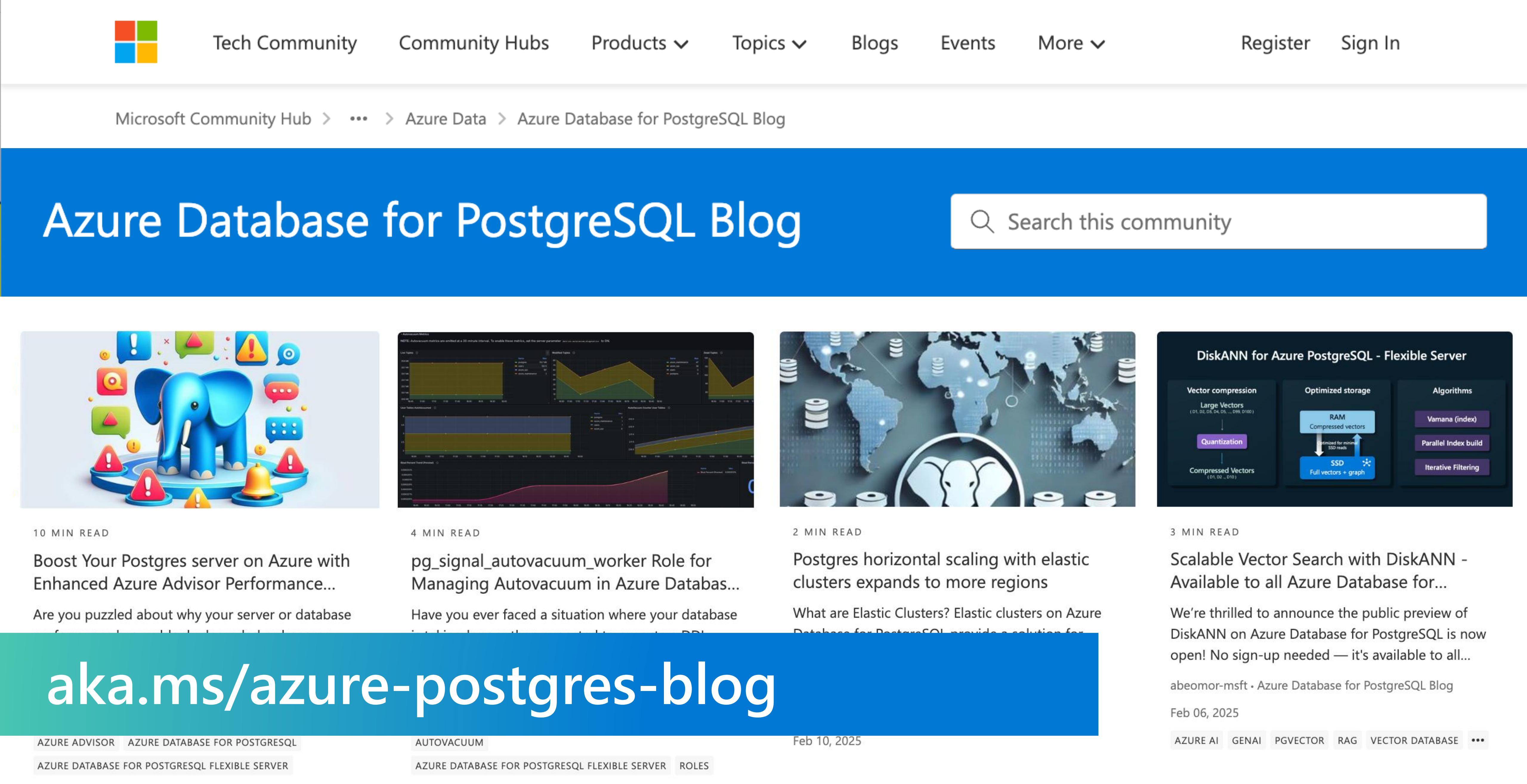
PG_DiskANN – Preview

Latest postgres minor versions

New Extensions

- postgresql-hll
- topN
- Tdigest

QuickStart guide for .NET SDK



Tech Community   Community Hubs   Products   Topics   Blogs   Events   More   Register   Sign In

Microsoft Community Hub > ••• > Azure Data > Azure Database for PostgreSQL Blog

Azure Database for PostgreSQL Blog

Search this community

10 MIN READ

Boost Your Postgres server on Azure with Enhanced Azure Advisor Performance...

Are you puzzled about why your server or database

AZURE ADVISOR   AZURE DATABASE FOR POSTGRESQL   AZURE DATABASE FOR POSTGRESQL FLEXIBLE SERVER

4 MIN READ

pg_signal_autovacuum_worker Role for Managing Autovacuum in Azure Databas...

Have you ever faced a situation where your database

AUTOVACUUM   AZURE DATABASE FOR POSTGRESQL FLEXIBLE SERVER   ROLES

2 MIN READ

Postgres horizontal scaling with elastic clusters expands to more regions

What are Elastic Clusters? Elastic clusters on Azure

Feb 10, 2025

3 MIN READ

Scalable Vector Search with DiskANN - Available to all Azure Database for...

We're thrilled to announce the public preview of DiskANN on Azure Database for PostgreSQL is now open! No sign-up needed — it's available to all...

abeomor-msft • Azure Database for PostgreSQL Blog

Feb 06, 2025

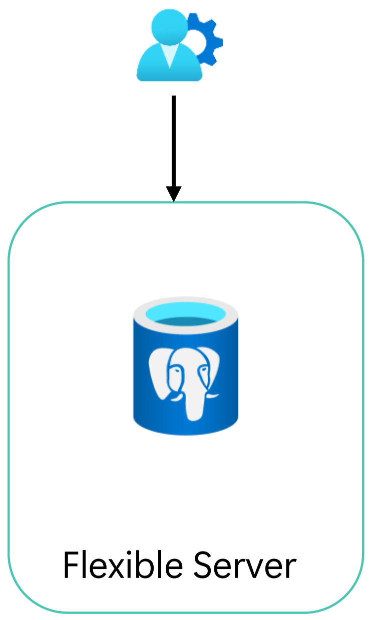
AZURE AI   GENAI   PGVECTOR   RAG   VECTOR DATABASE   •••

aka.ms/azure-postgres-blog

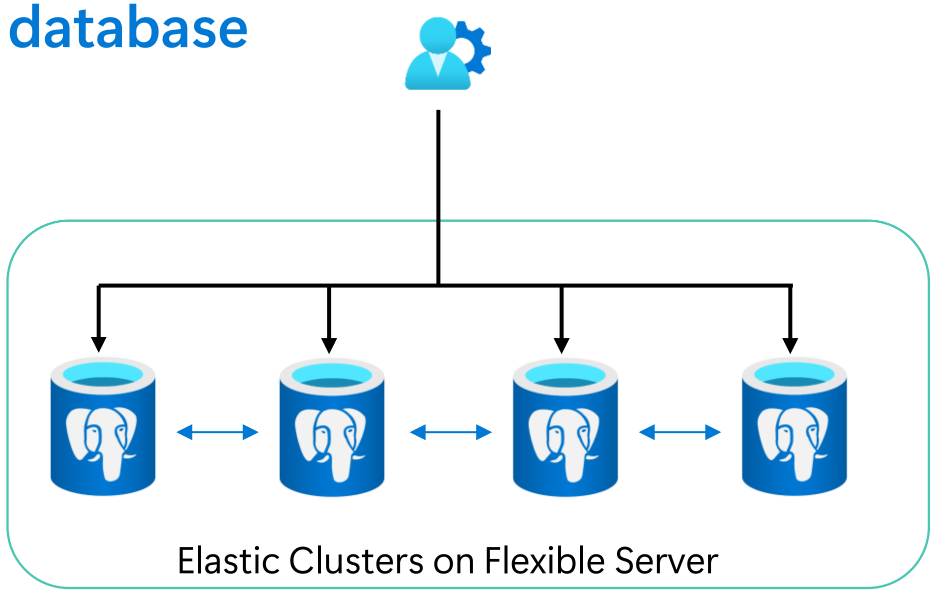
Elastic Clusters

Elastic clusters on Azure Database Postgres Flexible Server

Postgres Server



Multi-node Cluster – sharded database



Single cluster endpoint
Scale a single database horizontally
Shared nothing architecture
Powered by Citus OSS extension

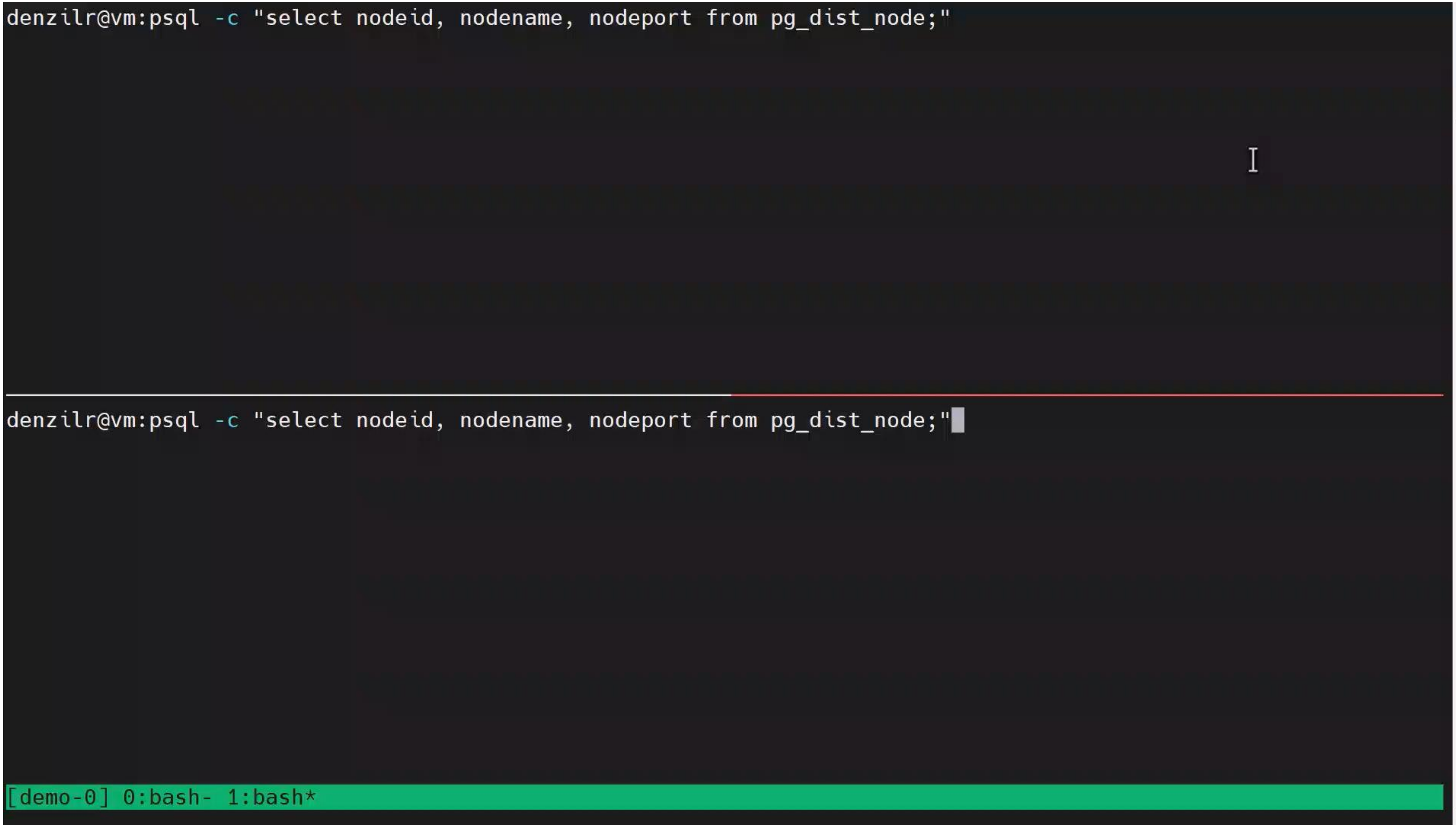
What sharding model do I use for Elastic Clusters in Azure Database for PostgreSQL (aka Citus)?

Schema-based sharding

- No schema changes required.
- No changes to queries.
- Easy for “lift-and-shift” for existing database per tenant apps.
- Each tenant isolated to own shard.
- Lower density (< 10K tenants).
- Can isolate tenant to a node.

Row-based sharding

- Highest density (> 100K tenants).
- Potential schema changes - sharding key.
- Queries should include tenant_id column.
- Security via row level security.
- Single table schema across tenants.
- Tenant isolation possible.



```
denzilr@vm:psql -c "select nodeid, nodename, nodeport from pg_dist_node;"
```

```
denzilr@vm:psql -c "select nodeid, nodename, nodeport from pg_dist_node;"
```

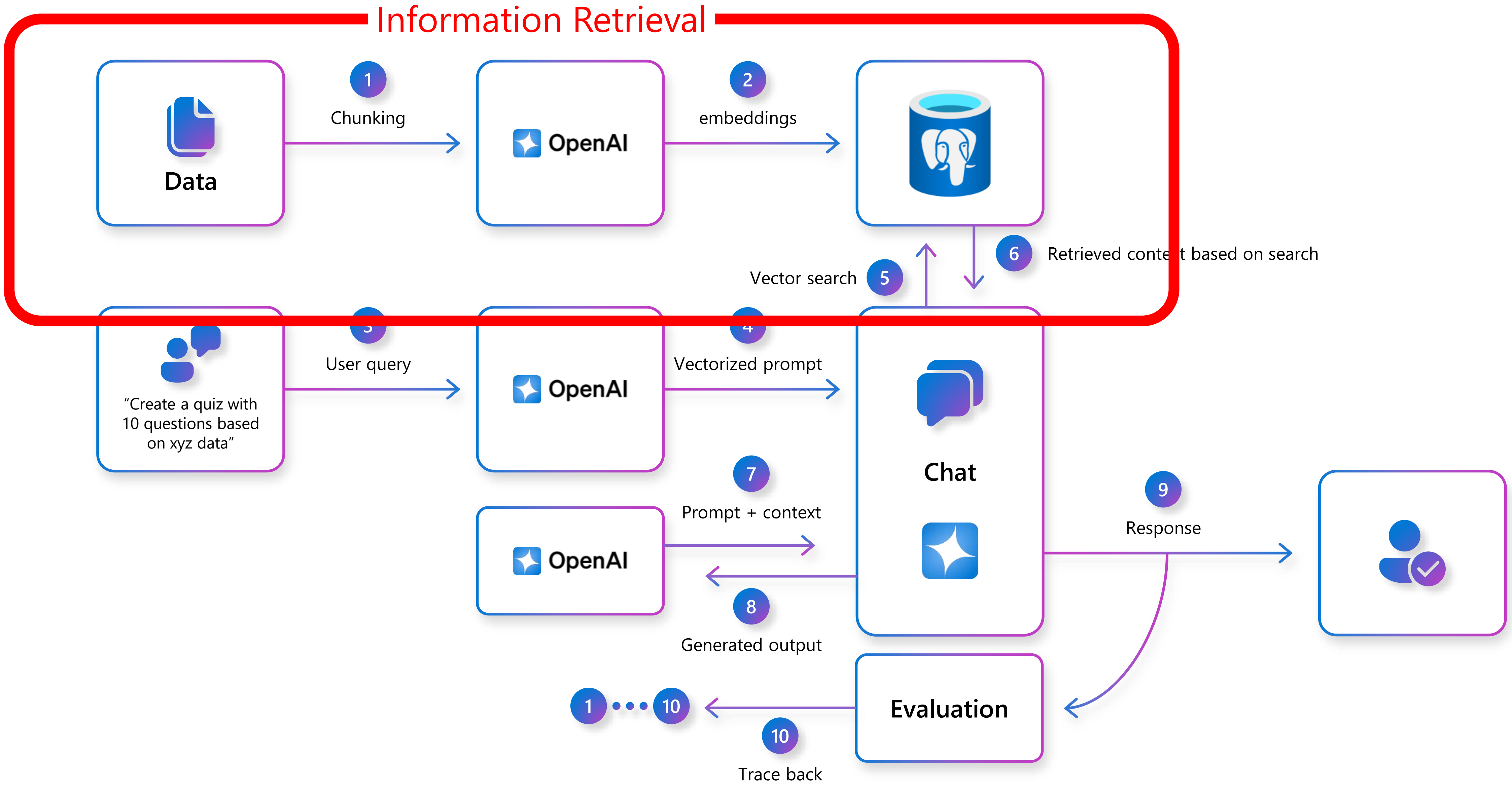
```
[demo-0] 0:bash- 1:bash*
```

Elastic Clusters on Azure Database for PostgreSQL Flexible Server

Business Continuity	Cluster Level Availability Zone Resiliency
	Cluster Level Backups and restores
	Cluster level HA
Scalability	Cluster level metrics
	Choice of Compute & Storage
	Ability to Scale up or scale out
Security	Private Link
	Active Directory Authentication
Manageability	Online Rebalancing
	Tenant Isolation
	Tenant Level Monitoring

AI for building intelligent applications

Basic Retrieval Augmented Generation (RAG)



Two Problems in Information Retrieval

Basic RAG

- **Scale** – efficiently scaling vector stores to 10M+ of vectors is hard.

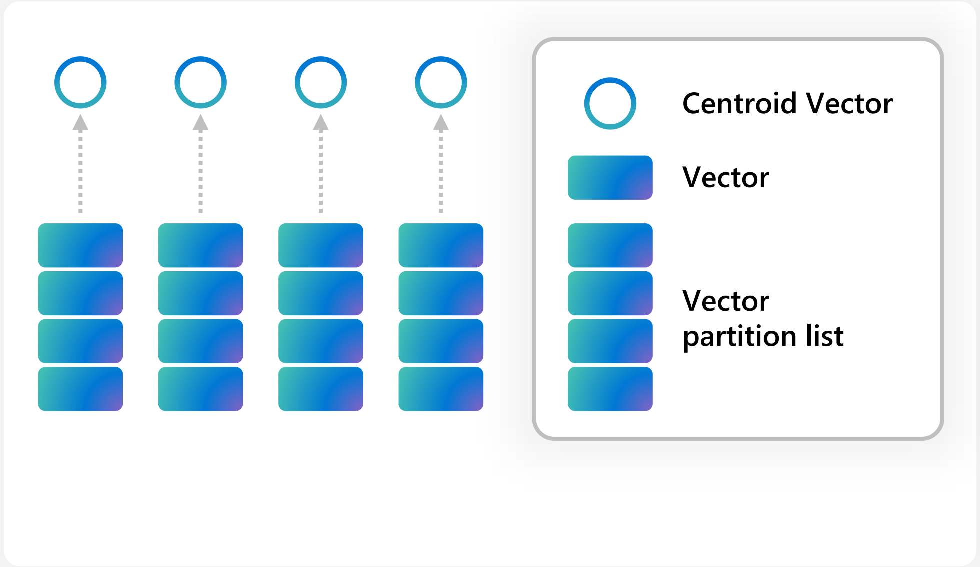
Advanced RAG

- **Accuracy** – quality of GenAI app responses and vector search accuracy need to improve.

Vector indexes popular today

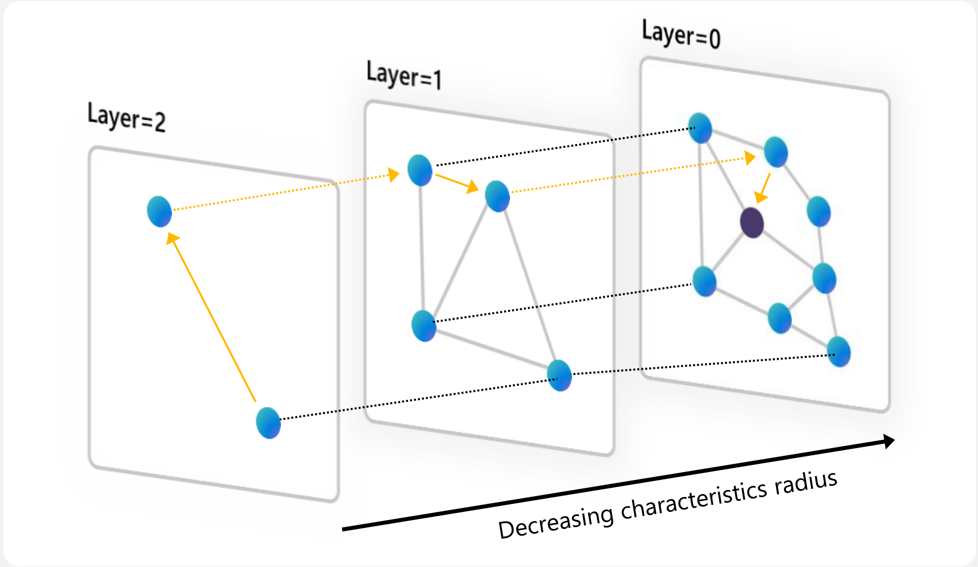
IVFFlat

- Clusters vectors by applying k-means clustering.
- Memory efficient but requires index rebuilds.



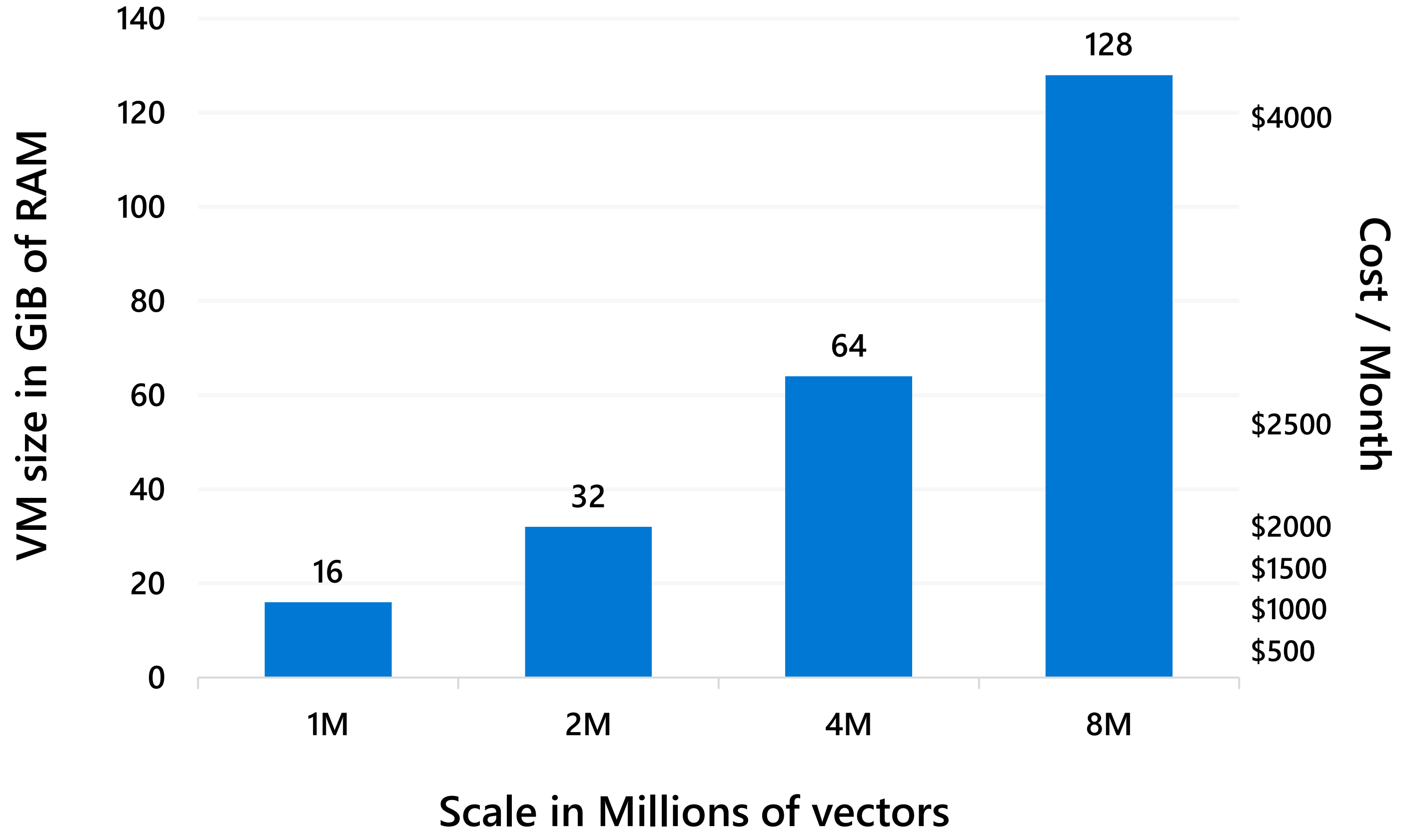
HNSW

- Builds a multi-layer graph with long and short connections between the vectors.
- The graph can be incrementally updated.



Vector store cost grows quickly

Cost of storing full 1536-dimensional embeddings on Postgres (HNSW)



Validations performed across varying SKUs of Azure Database for PostgreSQL Servers with service defaults

DiskANN Vector Index

Preview

Unique to Azure

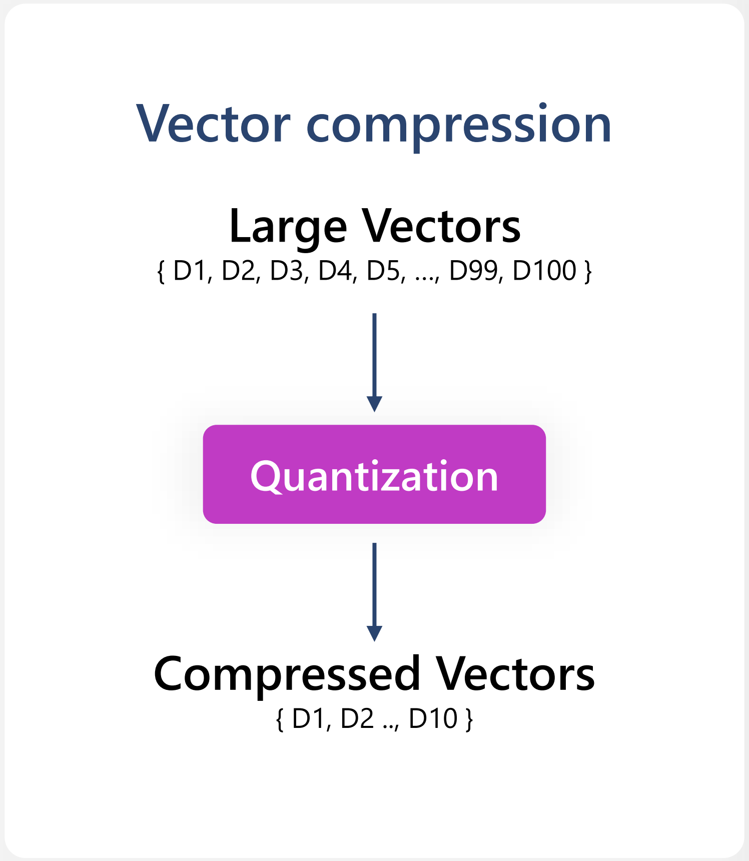
Highly **performant, scalable, and accurate** index for vectors

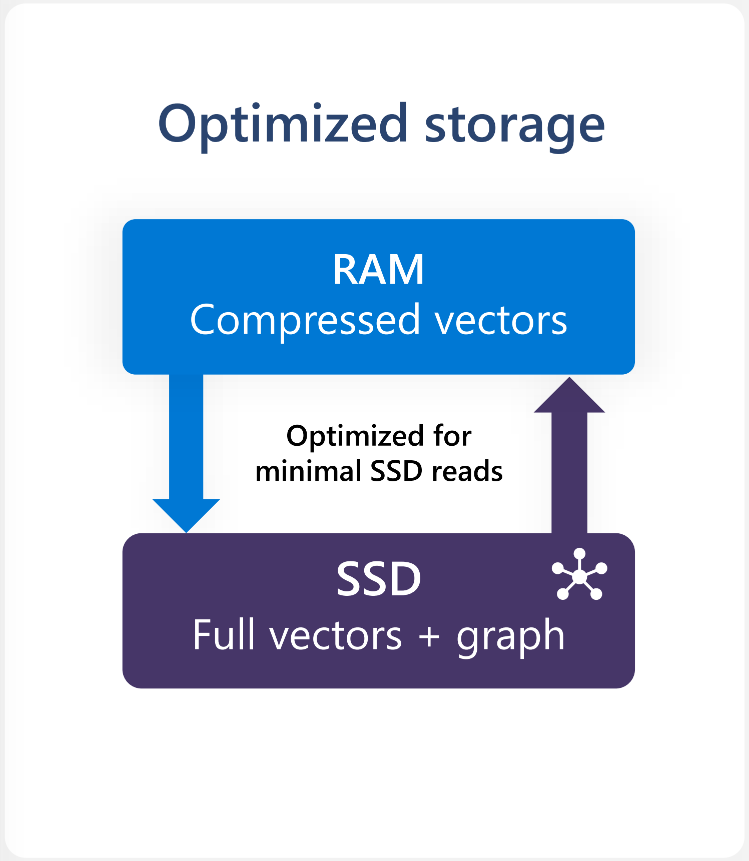
Superior to IVFLAT and HNSW

Reduced memory footprint by storing vectors on SSD

Compression and quantization **improve speed and accuracy** of vector search

Accuracy retained as data changed





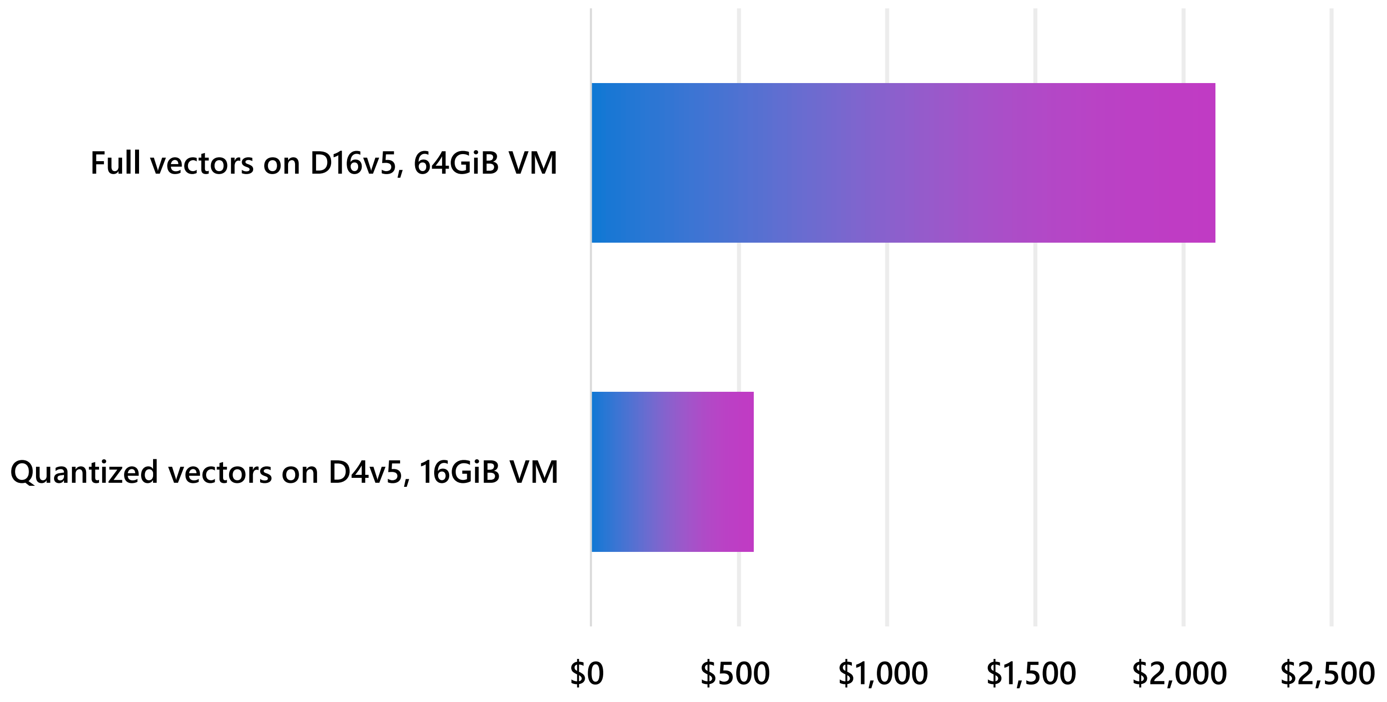
DiskANN Product Quantization - Cost

Coming Soon

Cost to achieve low latency

4x

Lower cost



Dataset: OpenAI generated 4M vectors, 1536 dimensions, 33GiB

Demo – Advanced RAG Solution Accelerator

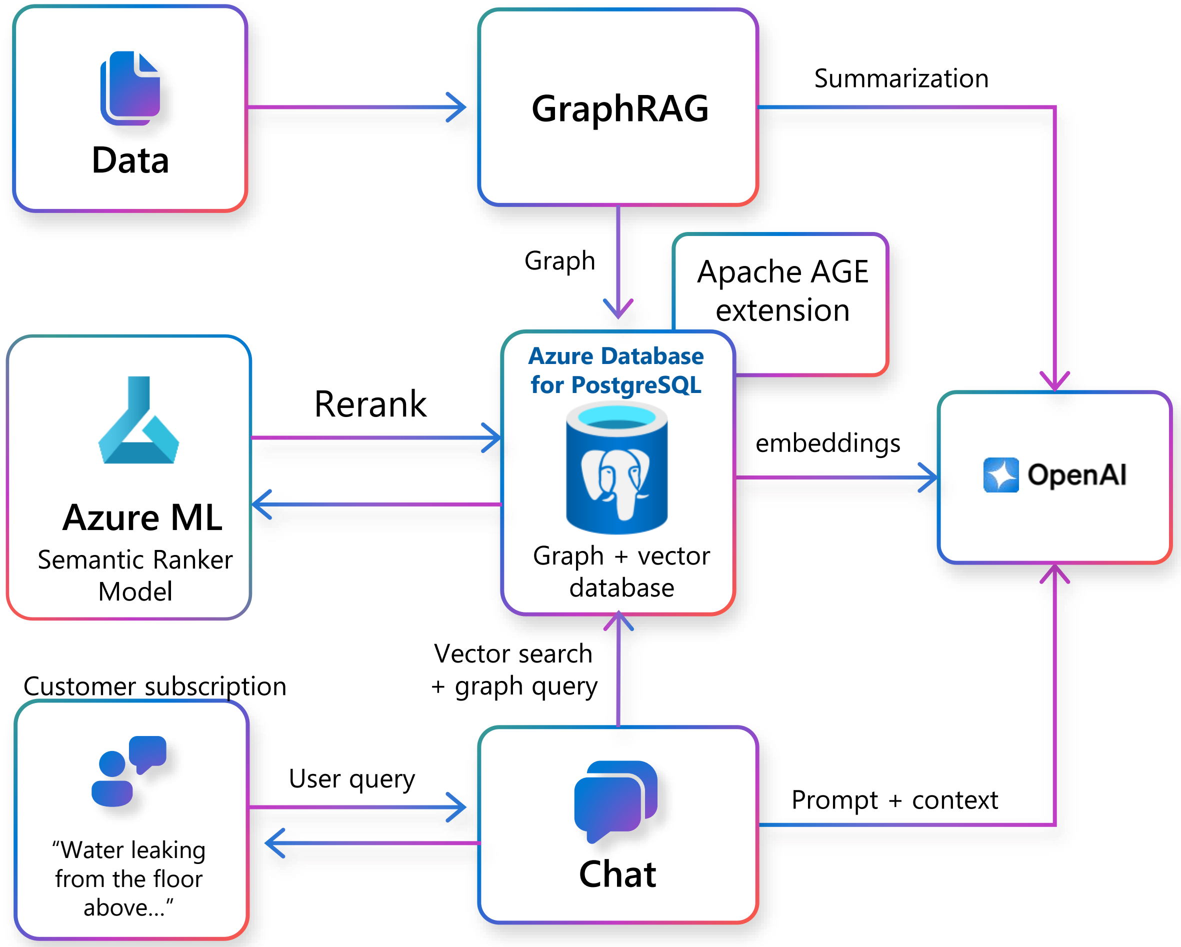
Preview

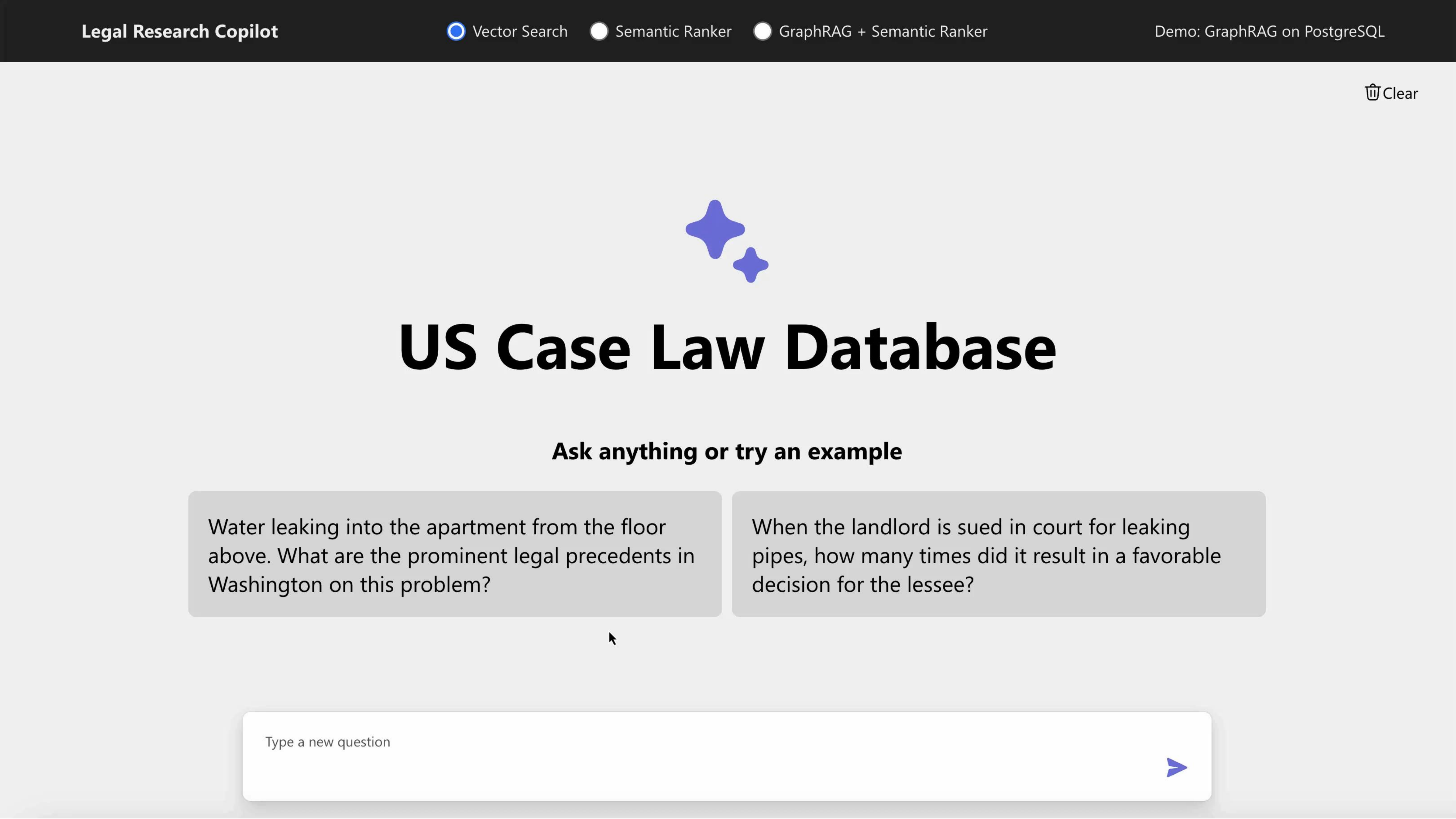
Demo Scenario

- Legal Research Copilot app
- U.S. Case Law dataset (0.5 million cases)

Key Components

- **DiskANN** for vector search
- **Semantic ranking** using BGE-reranker-v2-m3 model hosted in Azure ML
- **GraphRAG** from Microsoft Research for graph summarization
- **Apache AGE PG extension** for storing the graph
- **Azure_ai extension** provides a SQL-based interface to integrate with AI services




Legal Research Copilot
[✓ Vector Search] Semantic Ranker GraphRAG + Semantic Ranker
Demo: GraphRAG on PostgreSQL
Clear

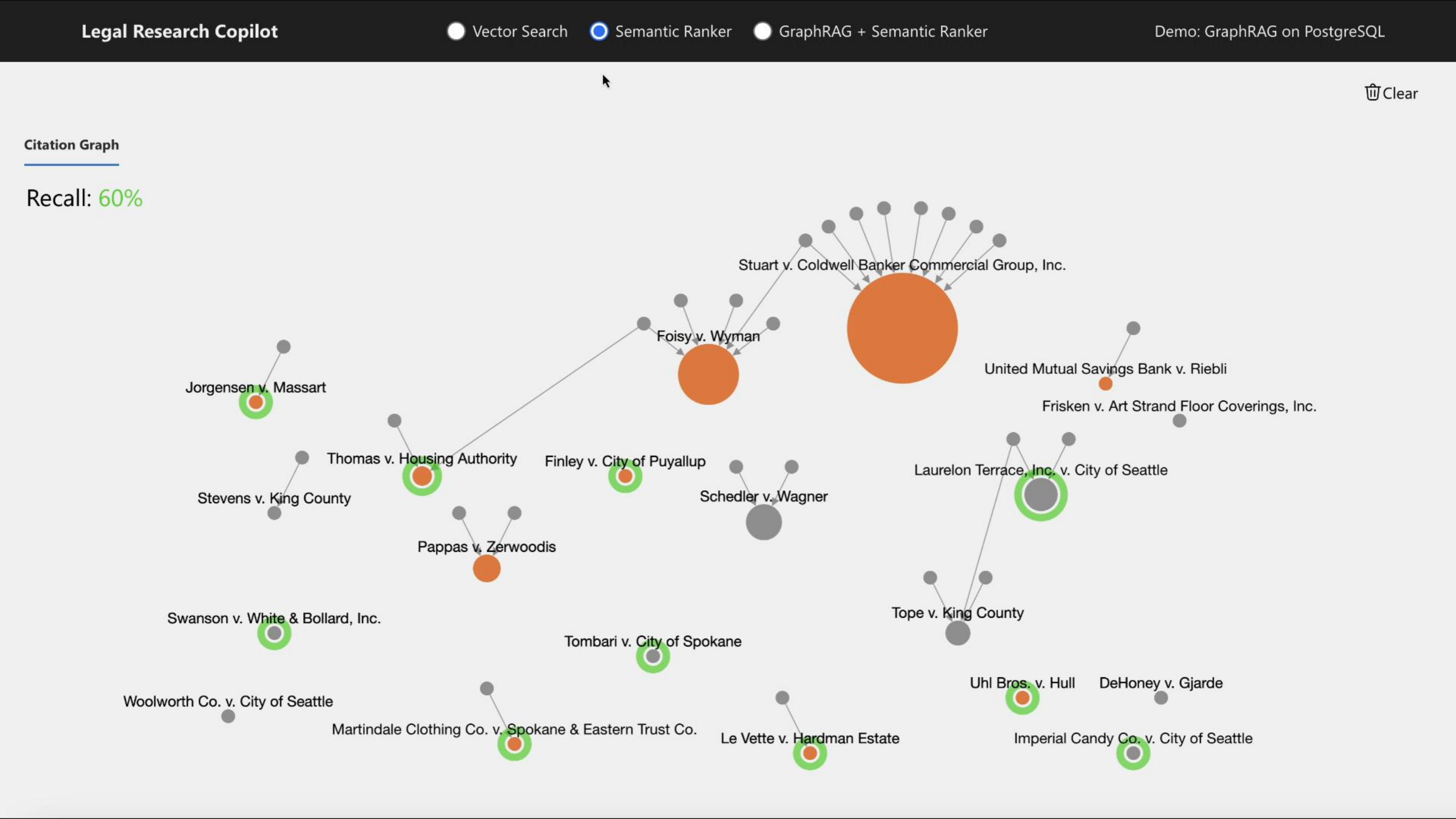
US Case Law Database

Ask anything or try an example

Water leaking into the apartment from the floor above. What are the prominent legal precedents in Washington on this problem?

When the landlord is sued in court for leaking pipes, how many times did it result in a favorable decision for the lessee?

Type a new question



Legal Research Copilot

Vector Search [✓ Semantic Ranker] GraphRAG + Semantic Ranker

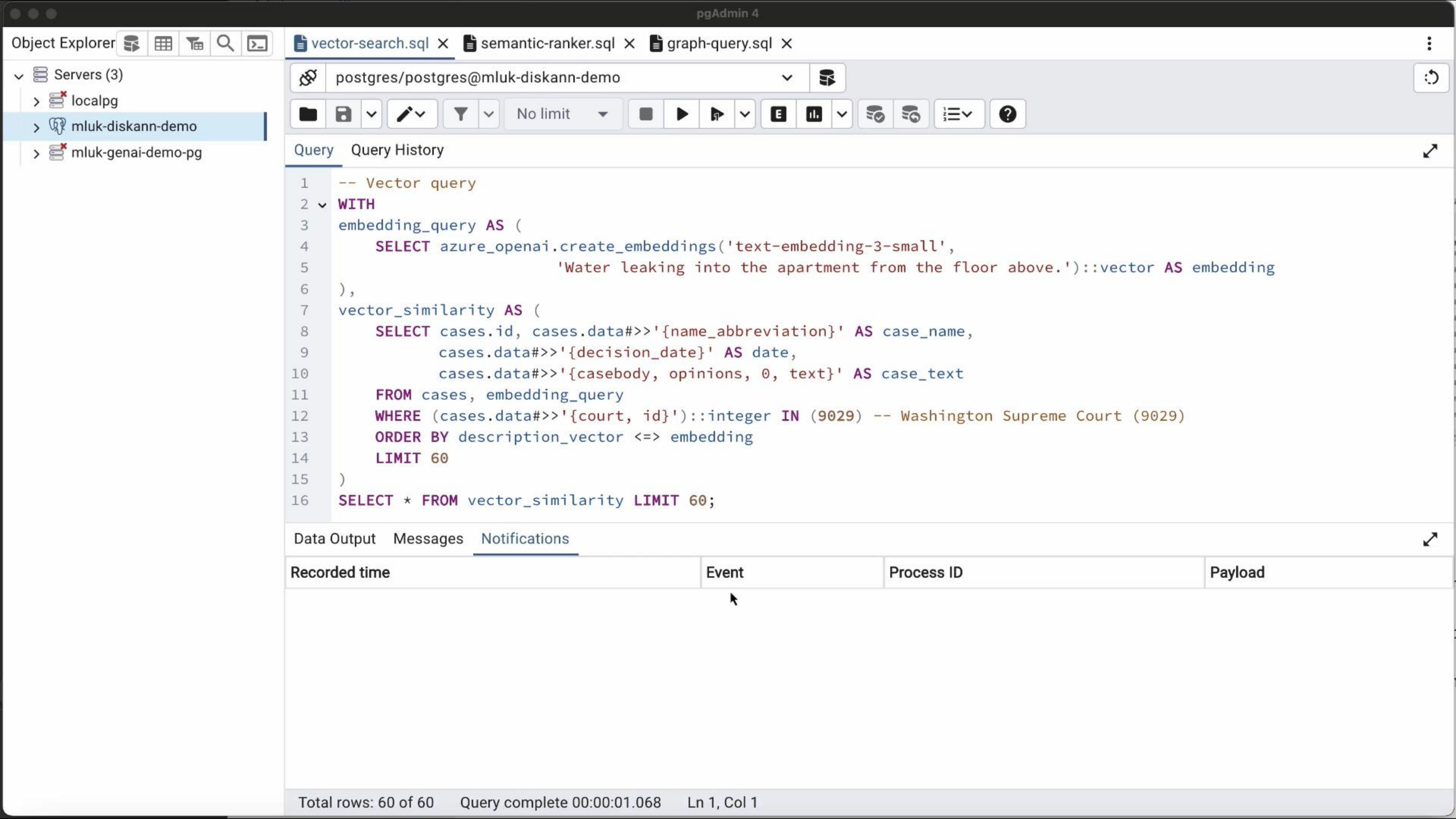
Demo: GraphRAG on PostgreSQL

Clear

Citation Graph

Recall: 60%

Stuart v. Coldwell Banker Commercial Group, Inc.
Foisy v. Wyman
United Mutual Savings Bank v. Riebli
Frisken v. Art Strand Floor Coverings, Inc.
Jorgensen v. Massart
Thomas v. Housing Authority
Finley v. City of Puyallup
Schedler v. Wagner
Laurelon Terrace, Inc. v. City of Seattle
Stevens v. King County
Pappas v. Zerwoodis
Tope v. King County
Swanson v. White & Bollard, Inc.
Tombari v. City of Spokane
Uhl Bros. v. Hull
DeHoney v. Gjarde
Woolworth Co. v. City of Seattle
Martindale Clothing Co. v. Spokane & Eastern Trust Co.
Le Vette v. Hardman Estate
Imperial Candy Co. v. City of Seattle



```sql
-- Vector query
WITH
embedding_query AS (
    SELECT azure_openai.create_embeddings('text-embedding-3-small',
                        'Water leaking into the apartment from the floor above.')::vector AS embedding
),
vector_similarity AS (
    SELECT cases.id, cases.data#>>'{name_abbreviation}' AS case_name,
           cases.data#>>'{decision_date}' AS date,
           cases.data#>>'{casebody, opinions, 0, text}' AS case_text
    FROM cases, embedding_query
    WHERE (cases.data#>>'{court, id}')::integer IN (9029) -- Washington Supreme Court (9029)
    ORDER BY description_vector <=> embedding
    LIMIT 60
)
SELECT * FROM vector_similarity LIMIT 60;
```

Recorded time	Event	Process ID	Payload

Total rows: 60 of 60   Query complete 00:00:01.068   Ln 1, Col 1

Free
Socks
@ Booth

Microsoft

Got 3 minutes? We'd love your input on some of our Postgres work

Get your FREE socks @ Microsoft booth

Microsoft

లదావ్యానధ

நன்றி

നന്ദി

Thank you

ಧನ್ಯವಾದಗಳು

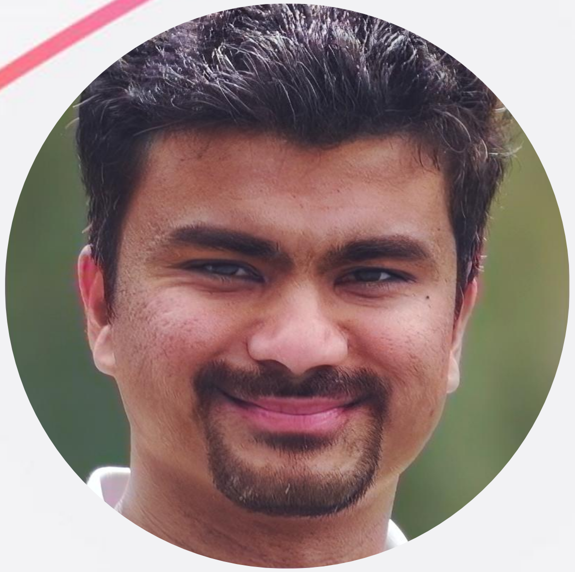
धन्यवाद

આભાર

ਤੁਹਾਡਾ ਧੰਨਵਾਦ

धन्यवाद

ଧନ୍ୟବାଦ



Sujit Kuruvilla

Director of Engineering @ Microsoft

linkedin.com/in/sujit-kuruvilla-7781a8