HOW 2026
Hello Open-source World

PostgreSQL in AI Applications

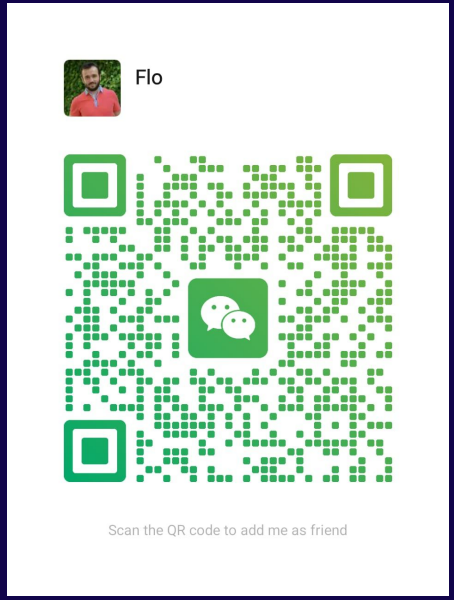
Reality Is More Complex Than Similarity Search

Florents Tselai

Florents Tselai - tselai.com

HOW 2026
Hello Open-source World

- Data Generalist
- Coming from a Data Engineering Background
- PostgreSQL Extensions
  - Vasco & pgxicor: correlation statistics
  - pgPDF
  - pgJQ
  - Spat: Redis-like extension for PG
- Core PostgreSQL: working on SQL/JSON & jsonpath



HOW 2026
Hello Open-source World

The Demo Trap

- It's easy to ship a demo: CREATE EXTENSION vector, insert some embeddings, run <=>.
- The "Demo Trap" ignores: high concurrency, data drift, disk bloat, and multi-tenant security.
- The real challenge: Pipelines, operations, and the data lifecycle.

HOW 2026
Hello Open-source World

PostgreSQL as the Orchestration Layer

- Stop treating Postgres as just a storage bucket.
- It is the orchestration layer:
- Security (RLS), Transactions (ACID), and Logic (Stored Procs/Jobs).
- Single source of truth vs. the "Specialized Silo" problem.
- Data Engineering nowdays has digressed… a lot!

HOW 2026
Hello Open-source World

Asynchronous Ingestion Pipelines

- Embeddings are expensive and slow to generate (API latency, GPU costs).
- Why synchronous generation fails: timeouts and cascading failures.
- The Solution: Decouple document intake from vector generation using internal queues.

HOW 2026
Hello Open-source World

High-Performance Job Queue

- The FOR UPDATE SKIP LOCKED pattern.
- SQL Pattern: WITH cte AS (...) UPDATE... RETURNING *.
- Parallelism: How 50 workers can grab 50 different jobs with zero lock contention.

```
# Dequeue IDs 10 at a time and pipe to a Python worker
while true; do
  psql -At -c "
    WITH cte AS (
      SELECT id FROM embedding_queue
      WHERE status = 'pending'
      ORDER BY created_at
      LIMIT 10
      FOR UPDATE SKIP LOCKED
    )
    UPDATE embedding_queue SET status = 'in_progress'
    FROM cte WHERE embedding_queue.id = cte.id
    RETURNING chunk_id;" | \
  xargs -n 1 -P 10 python3 generate_embeddings.py
  sleep 1
done
```

HOW 2026
Hello Open-source World

Managing The Data Lifcecycle

- What happens when a document changes?
- The cost of "re-embedding everything" vs. the value of incrementalism.
- Using hashing (MD5/SHA) to detect semantic deltas.

HOW 2026
Hello Open-source World

Recipe: Multi-Level Hashing

- Hashing at the file, page, and chunk level.
- Deduplication strategies: avoiding redundant embeddings for headers, footers, and legal disclaimers.
- Metadata design for efficient "delete-and-replace" operations.

HOW 2026
Hello Open-source World

Tablespaces and Hot/Cold Data

- Vector data is huge. You don't want it all on your most expensive NVMe drive.
- Implementing hot/cold storage tiers with Tablespaces.
- Moving 2024 partitions to cold storage while keeping 2025 data on "hot" disks.

```sql
-- Create a dedicated tablespace on high-performance NVMe storage
CREATE TABLESPACE fast_vector_ssd LOCATION '/mnt/nvme_ssd/pg_vector_data';

-- Main metadata table (resides in pg_default on standard disk)
CREATE TABLE documents (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    title TEXT,
    tenant_id UUID NOT NULL,
    created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Chunks table (resides on the fast NVMe tablespace)
CREATE TABLE document_chunks (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    document_id UUID REFERENCES documents(id) ON DELETE CASCADE,
    content TEXT NOT NULL,
    content_hash TEXT NOT NULL,
    embedding vector(1536)
) TABLESPACE fast_vector_ssd;
```

HOW 2026
Hello Open-source World

Selective Retrieval: Partial Indexes

- Don't search everything.
- Creating HNSW indexes with WHERE clauses.
- Practical case: Searching only "active" users or "legal" documents

HOW 2026
Hello Open-source World

Partitioning: Scaling to Billions

- Partitioning by Date, Region, or Tenant.
- The power of partition pruning: reducing search space by 90% before the vector index is even touched.
- Parallel vacuuming across partitions.

HOW 2026
Hello Open-source World

Security and Multi-tenancy (RLS)

- he risk of cross-tenant data leaks in RAG.
- Implementing RLS: ALTER TABLE... ENABLE ROW LEVEL SECURITY.
- Why "Manual WHERE clauses" are a security hole.

```sql
-- Enable the vector extension
CREATE EXTENSION IF NOT EXISTS vector;

-- Table for document chunks with tenant branding
CREATE TABLE knowledge_chunks (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL,
    content TEXT NOT NULL,
    embedding vector(1536) -- Matches OpenAI dimensions
);

-- 1. Enable Row-Level Security
ALTER TABLE knowledge_chunks ENABLE ROW LEVEL SECURITY;

-- 2. Force RLS (prevents the table owner from accidentally bypassing policies)
ALTER TABLE knowledge_chunks FORCE ROW LEVEL SECURITY;
```

```
-- Define the isolation policy
CREATE POLICY tenant_isolation_policy ON knowledge_chunks
FOR ALL
USING (tenant_id = current_setting('app.current_tenant_id')::uuid)
WITH CHECK (tenant_id = current_setting('app.current_tenant_id')::uuid);
```

```
# Within a single database transaction:
# 1. Set the tenant context securely (likely extracted from a JWT)
cursor.execute("SET LOCAL app.current_tenant_id = %s", [authenticated_tenant_id])

# 2. Perform semantic search as if it's a single-tenant DB
# RLS automatically adds: WHERE tenant_id = '...'
query = """
    SELECT content, embedding <=> %s AS distance
    FROM knowledge_chunks
    ORDER BY distance
    LIMIT 5;
"""
cursor.execute(query, [query_embedding])
results = cursor.fetchall()
```

HOW 2026
Hello Open-source World

Transactional Integrity

- The "Index says yes, Row says no" problem.
- How long-running transactions can return stale vectors.
- Ensuring embeddings stay in sync with the relational "truth".

HOW 2026
Hello Open-source World

Hybrid Query Power

- Vectors + FTS + JSONB in one SQL statement.
- Keyword search vs. Semantic search: why you need both.
-

```
WITH
-- 1. Vector (Semantic) Search CTE
semantic_search AS (
    SELECT
        id,
        name,
        description,
        metadata,
        -- Calculate rank based on vector cosine distance
        ROW_NUMBER() OVER (ORDER BY embedding <=> '[0.012, -0.054, ...]'::vector) AS
vector_rank
    FROM products
    -- JSONB Filtering applied early to reduce the search space
    WHERE metadata @> '{"category": "electronics", "in_stock": true}'
    ORDER BY embedding <=> '[0.012, -0.054, ...]'::vector
    LIMIT 100
),
```

```
-- 2. Full-Text (Keyword) Search CTE
keyword_search AS (
    SELECT
        id,
        -- Calculate rank based on keyword density/relevance
        ROW_NUMBER() OVER (
            ORDER BY ts_rank(to_tsvector('english', name || ' ' || description),
                     websearch_to_tsquery('english', 'wireless noise canceling
headphones')) DESC
        ) AS fts_rank
    FROM products
    -- The same JSONB filtering
    WHERE metadata @> '{"category": "electronics", "in_stock": true}'
      AND to_tsvector('english', name || ' ' || description) @@
websearch_to_tsquery('english', 'wireless noise canceling headphones')
    LIMIT 100
)
```

```sql
-- 3. Reciprocal Rank Fusion (RRF) Calculation
SELECT
    COALESCE(s.id, k.id) AS product_id,
    COALESCE(s.name, (SELECT name FROM products WHERE id = k.id)) AS product_name,
    COALESCE(s.metadata, (SELECT metadata FROM products WHERE id = k.id)) AS
product_metadata,

    -- The RRF Formula: 1 / (k + rank) where k is traditionally 60
    (COALESCE(1.0 / (60 + s.vector_rank), 0.0) +
     COALESCE(1.0 / (60 + k.fts_rank), 0.0)) AS rrf_score

FROM semantic_search s
FULL OUTER JOIN keyword_search k ON s.id = k.id
ORDER BY rrf_score DESC
LIMIT 10;
```

HOW 2026
Hello Open-source World

Thank you!

Flo

Scan the QR code to add me as friend