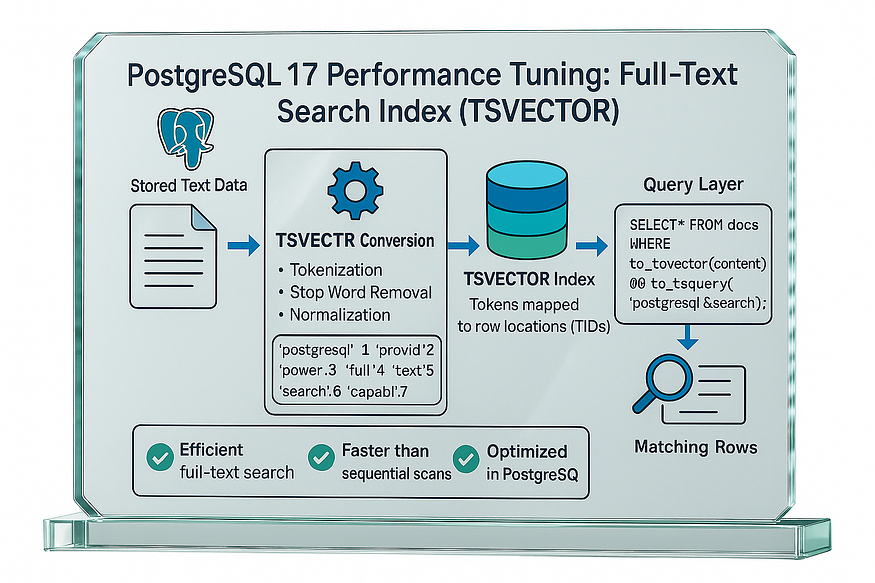
# **20 - PostgreSQL 17 Performance Tuning: Full-Text Search Index (TSVECTOR**



A ****Full-Text Search Index (TSVECTOR)**** in PostgreSQL is a special way of indexing text data so that it can be searched quickly and efficiently. Instead of storing raw text, PostgreSQL converts the text into a tsvector, which is a preprocessed form that breaks the text into tokens (words), removes stop words (like “a”, “the”), and normalizes words to their root form (e.g., “running”, “ran” → “run”).

This tsvector is then indexed, usually with a ****GIN (Generalized Inverted Index)**** or ****GiST (Generalized Search Tree)**** index, which allows PostgreSQL to match search queries (tsquery) against documents at high speed.

By using tsvector, PostgreSQL supports full-text search features like:

* Finding documents that contain specific words or phrases.
* Ranking results based on relevance.
* Highlighting matched words in search results.

In short, a Full-Text Search Index with tsvector transforms text into a searchable format and, when combined with an index, provides fast and accurate full-text search inside PostgreSQL.

## **Step 1 — Create the table**

CREATE TABLE articles (  
 id BIGSERIAL PRIMARY KEY,  
 title TEXT,  
 body TEXT,  
 published\_at TIMESTAMPTZ DEFAULT now()  
);

postgres=#  
postgres=# CREATE TABLE articles (  
 id BIGSERIAL PRIMARY KEY,  
 title TEXT,  
 body TEXT,  
 published\_at TIMESTAMPTZ DEFAULT now()  
);  
CREATE TABLE  
postgres=#

## **Step 2 — Load 1,000,000 rows (synthetic corpus)**

We’ll sprinkle common tech keywords so the search actually matches.

INSERT INTO articles (title, body)  
SELECT  
 'Post ' || g || ' about ' ||  
 (ARRAY['database','performance','indexing','postgresql','tuning',  
 'sql','json','cache','query','optimizer'])  
 [1 + (random()\*9)::int] AS title,  
 'This article discusses ' ||  
 (ARRAY['database','performance tuning','indexing strategies','PostgreSQL 17','SQL queries',  
 'GIN indexes','full-text search','planning and costing','caching','scalability'])  
 [1 + (random()\*9)::int] ||  
 ' with practical tips on ' ||  
 (ARRAY['tsvector','tsquery','plainto\_tsquery','to\_tsquery','ranking','highlighting'])  
 [1 + (random()\*5)::int] || '.' AS body  
FROM generate\_series(1, 1000000) g;

postgres=# INSERT INTO articles (title, body)  
SELECT  
 'Post ' || g || ' about ' ||  
 (ARRAY['database','performance','indexing','postgresql','tuning',  
 'sql','json','cache','query','optimizer'])  
 [1 + (random()\*9)::int] AS title,  
 'This article discusses ' ||  
 (ARRAY['database','performance tuning','indexing strategies','PostgreSQL 17','SQL queries',  
 'GIN indexes','full-text search','planning and costing','caching','scalability'])  
 [1 + (random()\*9)::int] ||  
 ' with practical tips on ' ||  
 (ARRAY['tsvector','tsquery','plainto\_tsquery','to\_tsquery','ranking','highlighting'])  
 [1 + (random()\*5)::int] || '.' AS body  
FROM generate\_series(1, 1000000) g;

## **Step 3 — Analyze statistics**

ANALYZE articles;

postgres=# ANALYZE articles;  
ANALYZE  
postgres=#

## **Step 4 — Baseline query without an index**

We’ll search for “****database performance****”. Computing the vector on the fly forces a full scan.

EXPLAIN ANALYZE  
SELECT count(\*)  
FROM articles  
WHERE to\_tsvector('english', coalesce(title,'') || ' ' || coalesce(body,'')) @@  
 plainto\_tsquery('english','database performance');

postgres=# EXPLAIN ANALYZE  
SELECT count(\*)  
FROM articles  
WHERE to\_tsvector('english', coalesce(title,'') || ' ' || coalesce(body,'')) @@  
 plainto\_tsquery('english','database performance');  
 QUERY PLAN  
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------  
 Finalize Aggregate (cost=130449.57..130449.58 rows=1 width=8) (actual time=13185.824..13191.788 rows=1 loops=1)  
 -> Gather (cost=130449.36..130449.57 rows=2 width=8) (actual time=13184.608..13191.776 rows=3 loops=1)  
 Workers Planned: 2  
 Workers Launched: 2  
 -> Partial Aggregate (cost=129449.36..129449.37 rows=1 width=8) (actual time=13172.112..13172.115 rows=1 loops=3)  
 -> Parallel Seq Scan on articles (cost=0.00..129449.33 rows=10 width=0) (actual time=7.202..13158.179 rows=4123 loops=3)  
 Filter: (to\_tsvector('english'::regconfig, ((COALESCE(title, ''::text) || ' '::text) || COALESCE(body, ''::text))) @@ '''databas'' & ''perform'''::tsquery)  
 Rows Removed by Filter: 329211  
 Planning Time: 0.090 ms  
 Execution Time: 13191.826 ms  
(10 rows)

postgres=#

Time: 13191.826 ms (00:13.826)  
postgres=#

With no index, PostgreSQL scans all 1M rows (~*13.826* s).

## **Step 5 — Add a generated**tsvector**column (stored)**

This keeps document vectors always up-to-date without triggers.

ALTER TABLE articles  
ADD COLUMN tsv tsvector  
GENERATED ALWAYS AS (  
 to\_tsvector('english', coalesce(title,'') || ' ' || coalesce(body,''))  
) STORED;

postgres=# postgres=# ALTER TABLE articles  
ADD COLUMN tsv tsvector  
GENERATED ALWAYS AS (  
 to\_tsvector('english', coalesce(title,'') || ' ' || coalesce(body,''))  
) STORED;  
ALTER TABLE  
postgres=#

## **Step 6 — Create a GIN index on the**tsvector

CREATE INDEX idx\_articles\_tsv\_gin ON articles USING gin (tsv);

postgres=# CREATE INDEX idx\_articles\_tsv\_gin ON articles USING gin (tsv);  
CREATE INDEX  
postgres=#

*(If building online in production, use CREATE INDEX CONCURRENTLY.)*

## **Analyze statistics**

ANALYZE articles;

postgres=# ANALYZE articles;  
ANALYZE  
postgres=#

## **Step 7 — Search again (now using the index)**

postgres=# EXPLAIN ANALYZE  
SELECT count(\*)  
FROM articles  
WHERE tsv @@ plainto\_tsquery('english','database performance');  
 QUERY PLAN  
-----------------------------------------------------------------------------------------------------------------------------------------------  
 Aggregate (cost=34497.68..34497.69 rows=1 width=8) (actual time=39.425..39.429 rows=1 loops=1)  
 -> Bitmap Heap Scan on articles (cost=190.83..34440.90 rows=22714 width=0) (actual time=14.482..32.743 rows=12368 loops=1)  
 Recheck Cond: (tsv @@ '''databas'' & ''perform'''::tsquery)  
 Heap Blocks: exact=10602  
 -> Bitmap Index Scan on idx\_articles\_tsv\_gin (cost=0.00..185.15 rows=22714 width=0) (actual time=13.163..13.164 rows=12368 loops=1)  
 Index Cond: (tsv @@ '''databas'' & ''perform'''::tsquery)  
 Planning Time: 0.101 ms  
 Execution Time: 39.453 ms  
(8 rows)

postgres=#

Time: 39.453 ms  
postgres=#

GIN flips the plan to **Bitmap Index Scan**, dropping runtime from *****Time reduction:******(13,191.826 − 39.453) =******13,152.373 ms******saved (≈******99.701%******reduction)*