EDB
Postgres® for the AI Generation

From Distance to Intelligence: Evolving Your Vector Search

Bilge Ince & Boriss Mejias

©EDB 2025 — ALL RIGHTS RESERVED.

Bilge INCE

MLE @ EDB

Muay Thai, Running ❤️

@abilgegunduz

@bilge.bsky.social

©EDB 2025 — ALL RIGHTS RESERVED.

Boriss Mejías

Solutions architect @ EDB

Organizer of PGDay Lowlands

Air guitar \m/

@tchorix@mastodon.world

@tchorix.bsky.social

©EDB 2025 — ALL RIGHTS RESERVED.

Time to Rock

Find 11 rock music outfits

©EDB 2025 — ALL RIGHTS RESERVED.

Time to Rock

Find 11 rock music outfits for women

©EDB 2025 — ALL RIGHTS RESERVED.

Time to Rock

Find 11 **rock music outfits** for women

©EDB 2025 — ALL RIGHTS RESERVED.

Time to Rock

Find 11 rock music outfits for **women**

©EDB 2025 — ALL RIGHTS RESERVED.

Time to Rock

Find 11 rock music outfits for women

©EDB 2025 — ALL RIGHTS RESERVED.

LLMs

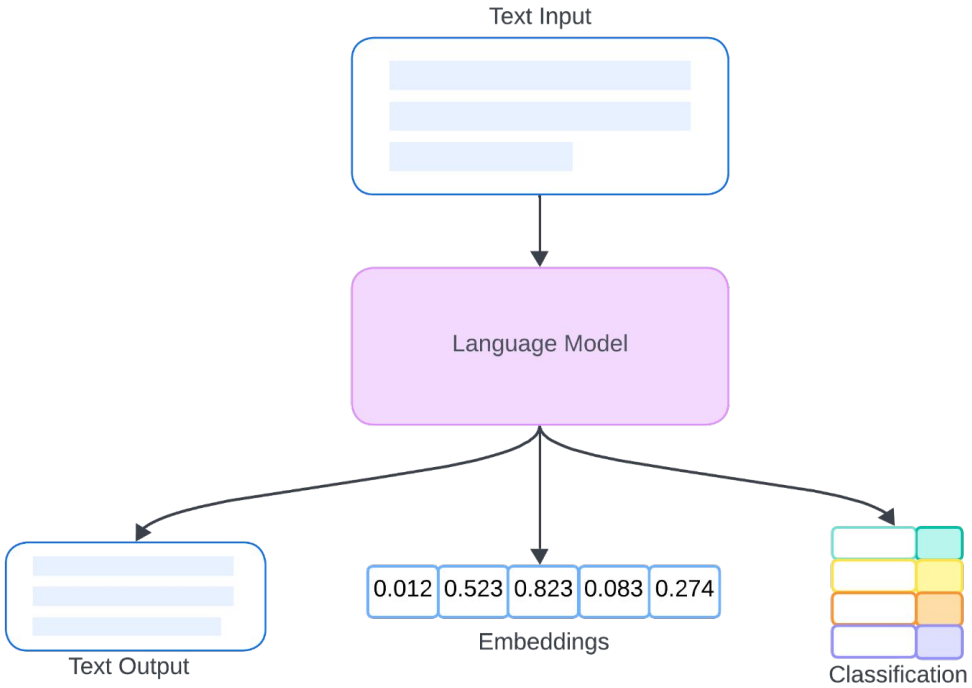


Fig: Hands on LLMs - Jay Alammar & Maarten Grootendorst

EDB Postgres for the AI Generation

©EDB 2025 — ALL RIGHTS RESERVED.

Time to Rock

Search: Rock music outfits

Filter: women

Limit to top 11

©EDB 2025 — ALL RIGHTS RESERVED.

Demo Time

Rock music outfit for women

©EDB 2025 — ALL RIGHTS RESERVED.

The products table

```
CREATE TABLE products (
    id          BIGINT PRIMARY KEY
  , img_id      BIGINT REFERENCES products_images(id)
  , description TEXT
  , gender      TEXT
);
```


©EDB 2025 — ALL RIGHTS RESERVED.

The products table

```
CREATE TABLE products (
    id          BIGINT PRIMARY KEY
  , img_id      BIGINT REFERENCES products_images(id)
  , description TEXT
  , gender      TEXT
);
```

```
CREATE TABLE products_embeddings (
    product_id BIGINT REFERENCES products(id)
  , embedding  vector(384)
);
```

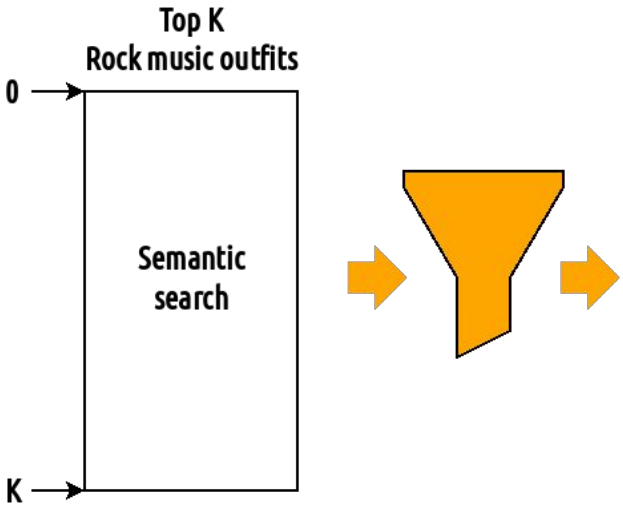

©EDB 2025 — ALL RIGHTS RESERVED.

Time to Rock

EDB
Postgres for the AI Generation

©EDB 2025 — ALL RIGHTS RESERVED.

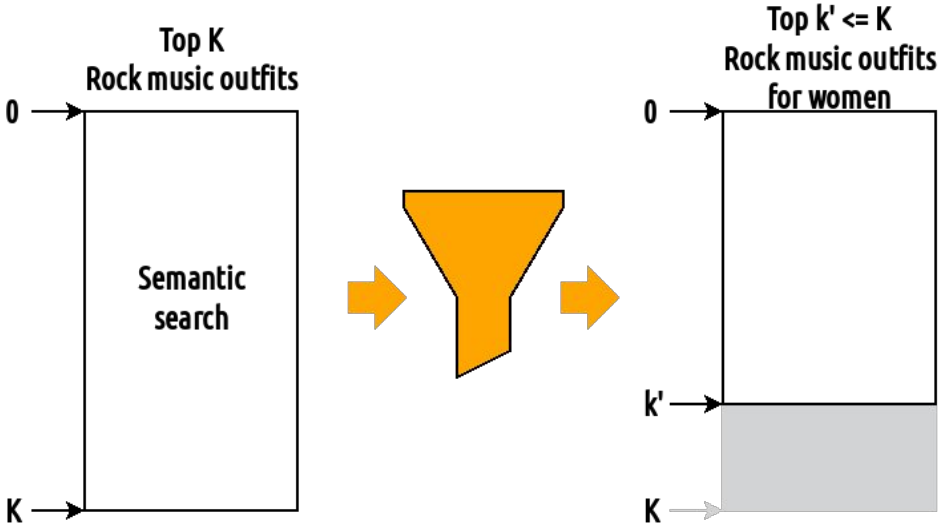
Time to Rock



EDB
Postgres for the AI Generation

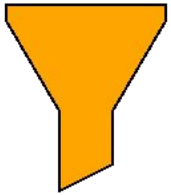
©EDB 2025 — ALL RIGHTS RESERVED.

Time to Rock



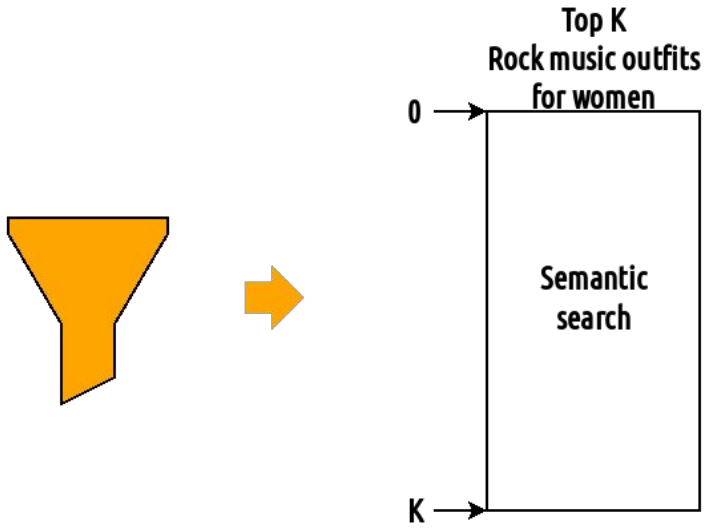
©EDB 2025 — ALL RIGHTS RESERVED.

Time to Rock



©EDB 2025 — ALL RIGHTS RESERVED.

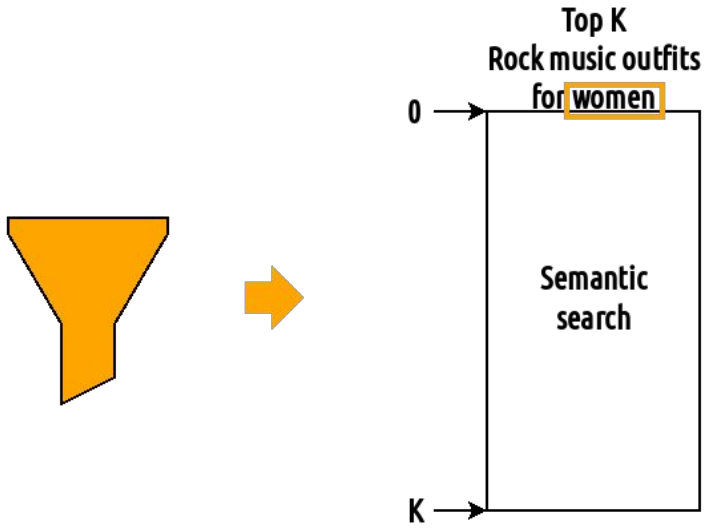
Time to Rock



EDB
Postgres for the AI Generation

©EDB 2025 — ALL RIGHTS RESERVED.

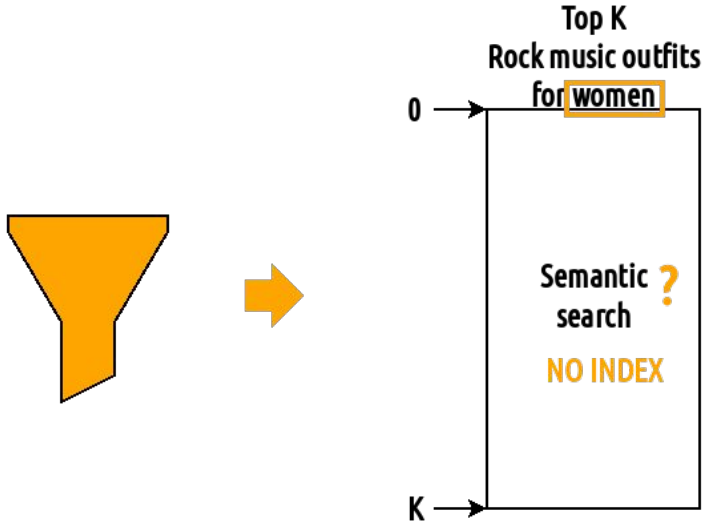
Time to Rock



EDB
Postgres for the AI Generation

©EDB 2025 — ALL RIGHTS RESERVED.

Time to Rock



©EDB 2025 — ALL RIGHTS RESERVED.

Why Getting Search Right Matters

Finding exactly the right product vs. irrelevant suggestions; Slack, Healthcare sector, e-commerce, game

©EDB 2025 — ALL RIGHTS RESERVED.

Demo

Floral Dress for the summer party

©EDB 2025 — ALL RIGHTS RESERVED.

The Promise of Understanding Semantic Meaning

- Finds **related concepts** even without keyword overlap.
- Excellent for **discovery** and **recommendation**.

©EDB 2025 — ALL RIGHTS RESERVED.

The Promise of Understanding Semantic Meaning

- Finds related concepts even without keyword overlap.
- Excellent for discovery and recommendation.
- Handles **synonyms** and related concepts naturally.

©EDB 2025 — ALL RIGHTS RESERVED.

The Promise of Understanding Semantic Meaning

- Finds related concepts even without keyword overlap.
- Excellent for discovery and recommendation.
- Handles synonyms and related concepts naturally.
- Enables **natural language querying** and Question Answering.
- Can search **across modalities** (text-image, text-speech).

©EDB 2025 — ALL RIGHTS RESERVED.

The Reality Behind The Hype

"Nearest" ≠ "Best"

The "Black Box”

Computational Cost

Keyword Blindness

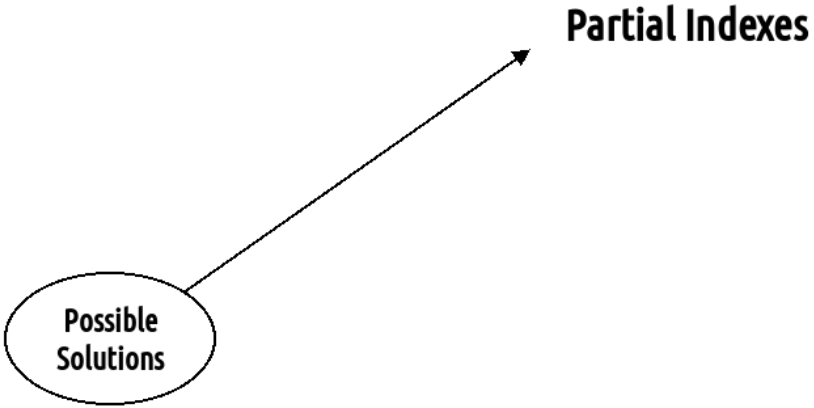
Context Ambiguity

©EDB 2025 — ALL RIGHTS RESERVED.

What can we do?

©EDB 2025 — ALL RIGHTS RESERVED.

What can we do?



©EDB 2025 — ALL RIGHTS RESERVED.

The products table

```
CREATE TABLE products (
    id          BIGINT PRIMARY KEY
  , img_id      BIGINT REFERENCES products_images(id)
  , description TEXT
  , gender      TEXT
);
```

```
CREATE TABLE products_embeddings (
    product_id BIGINT REFERENCES products(id)
  , embedding  vector(384)
);
```


©EDB 2025 — ALL RIGHTS RESERVED.

The products table

```
CREATE TABLE products (
    id          BIGINT PRIMARY KEY
  , img_id      BIGINT REFERENCES products_images(id)
  , description TEXT
  , gender      TEXT
  , embedding   vector(384)
);
```

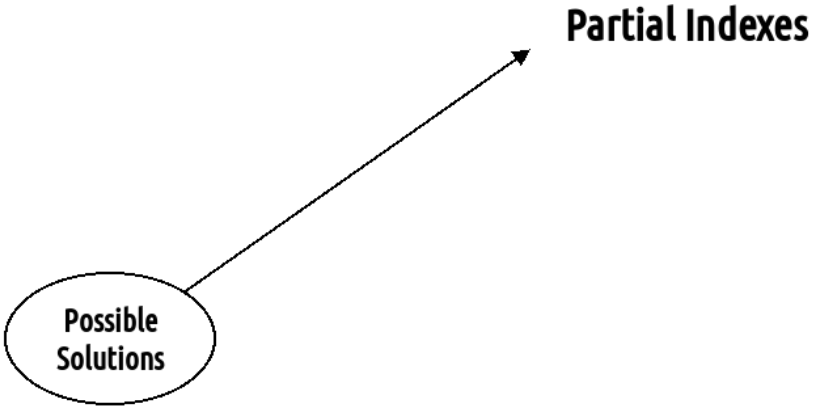

©EDB 2025 — ALL RIGHTS RESERVED.

The products table

```
CREATE TABLE products (
    id          BIGINT PRIMARY KEY
  , img_id      BIGINT REFERENCES products_images(id)
  , description TEXT
  , gender      TEXT
  , embedding   vector(384)
);
CREATE INDEX ON products USING hnsw (embedding vector_cosine_ops)
WHERE gender = 'women';
```

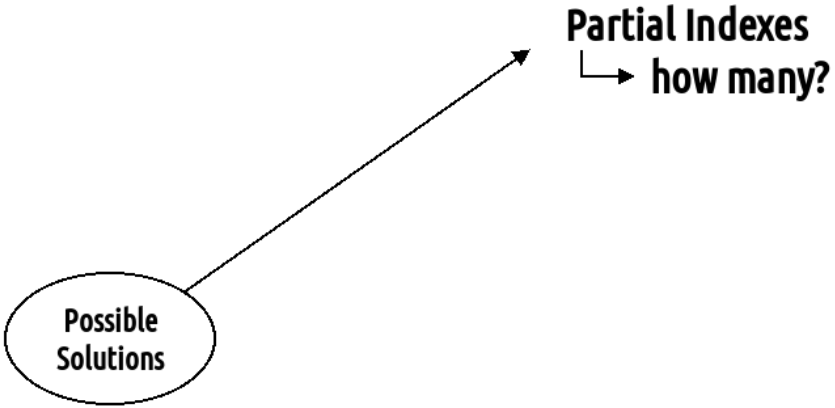

©EDB 2025 — ALL RIGHTS RESERVED.

What can we do?



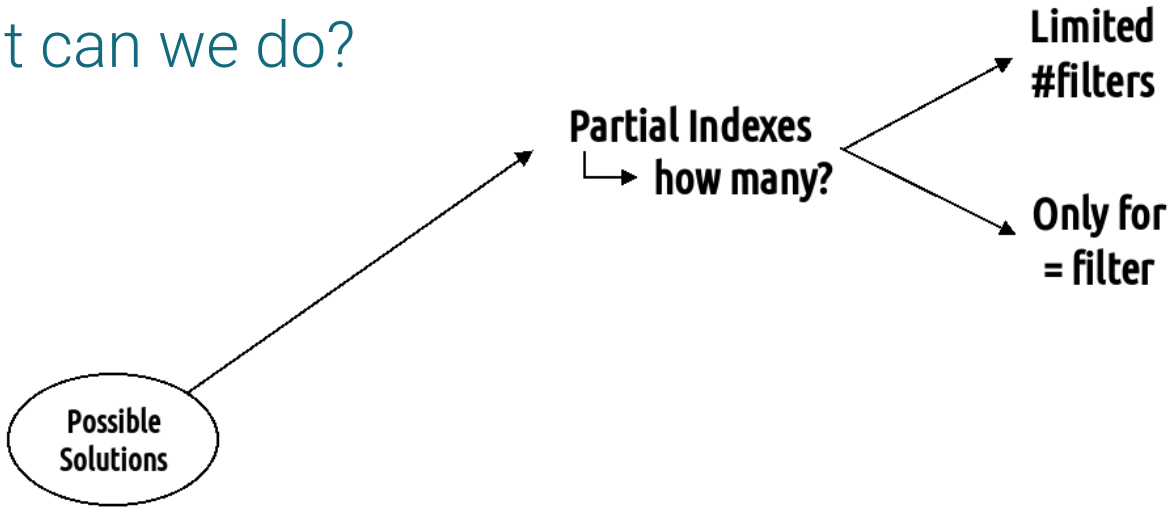
©EDB 2025 — ALL RIGHTS RESERVED.

What can we do?



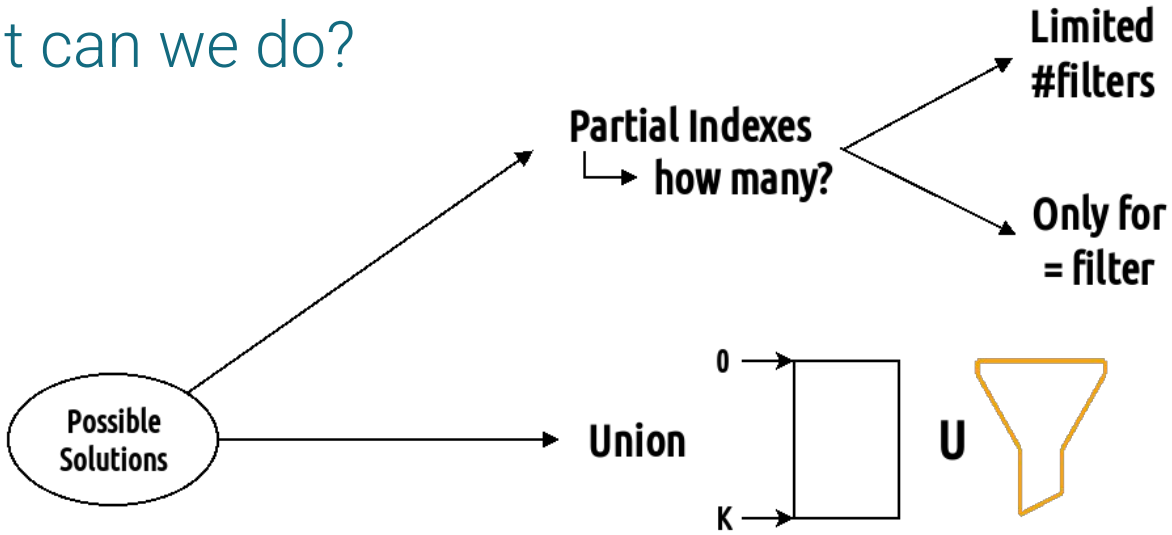
©EDB 2025 — ALL RIGHTS RESERVED.

What can we do?



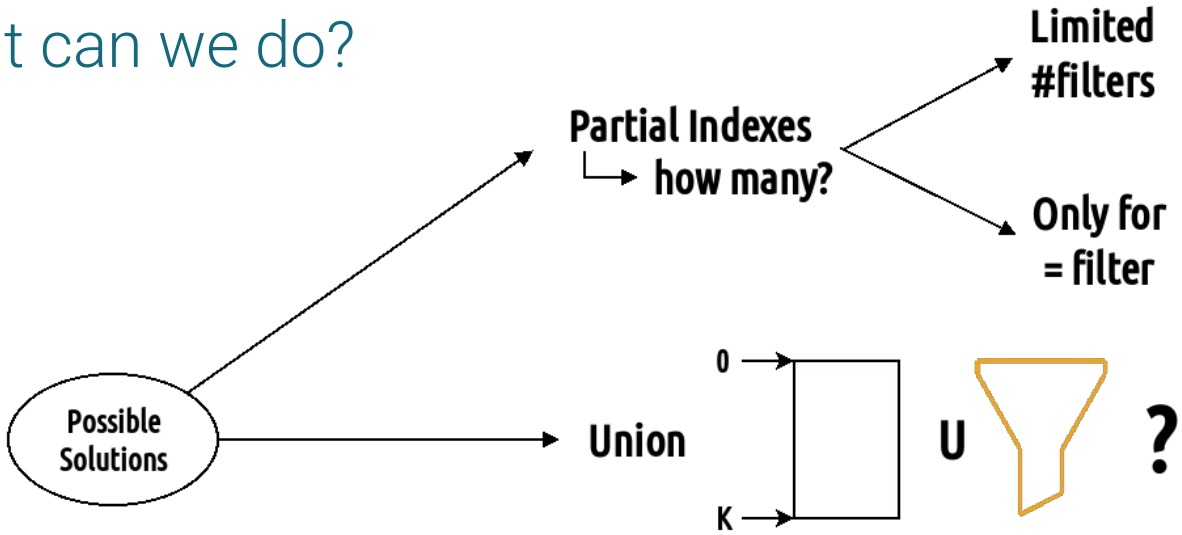
©EDB 2025 — ALL RIGHTS RESERVED.

What can we do?



©EDB 2025 — ALL RIGHTS RESERVED.

What can we do?



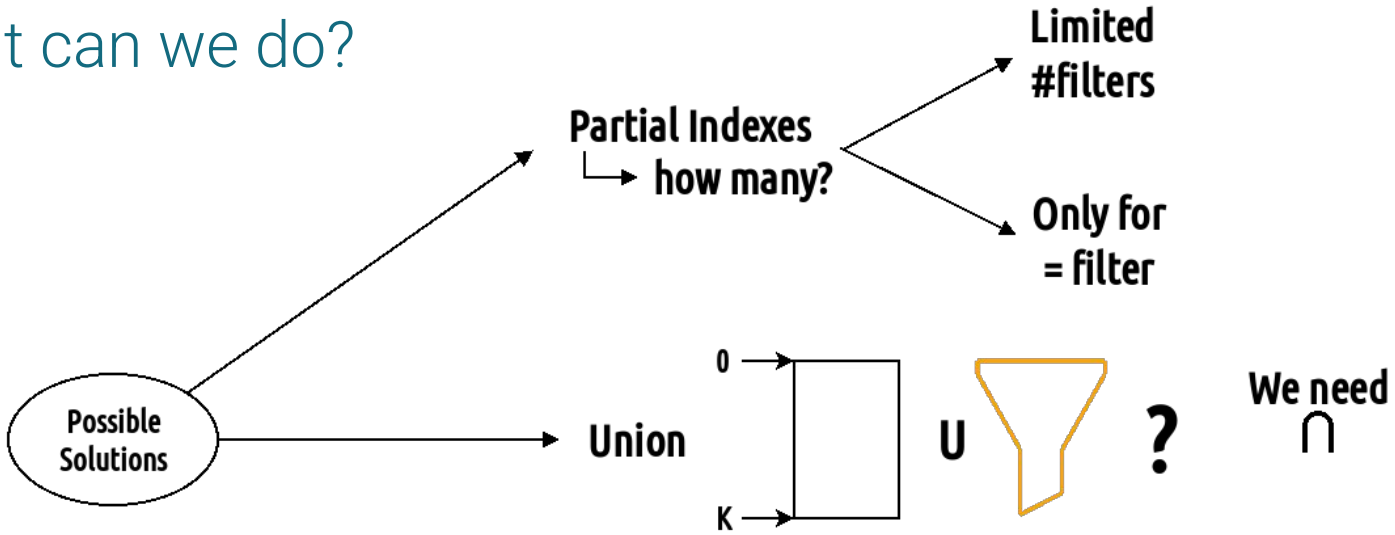
©EDB 2025 — ALL RIGHTS RESERVED.

The Solution: Hybrid Search - Union

- Run Full Text Search and Vector Search queries independently.
- Get two ranked lists of results.
- Combine these lists using an algorithm. Popular option: Reciprocal Rank Fusion

©EDB 2025 — ALL RIGHTS RESERVED.

What can we do?



©EDB 2025 — ALL RIGHTS RESERVED.

The Solution: Hybrid Search - Intersect

- Leverage the best of both worlds
- Find semantically relevant items that also satisfy keyword constraints or other filters.

The Method:

Use full text search for essential keywords to get an initial set of candidates (e.g., top 100-500 results).

Then run semantic search on the candidate data for user query that fits with semantic need.

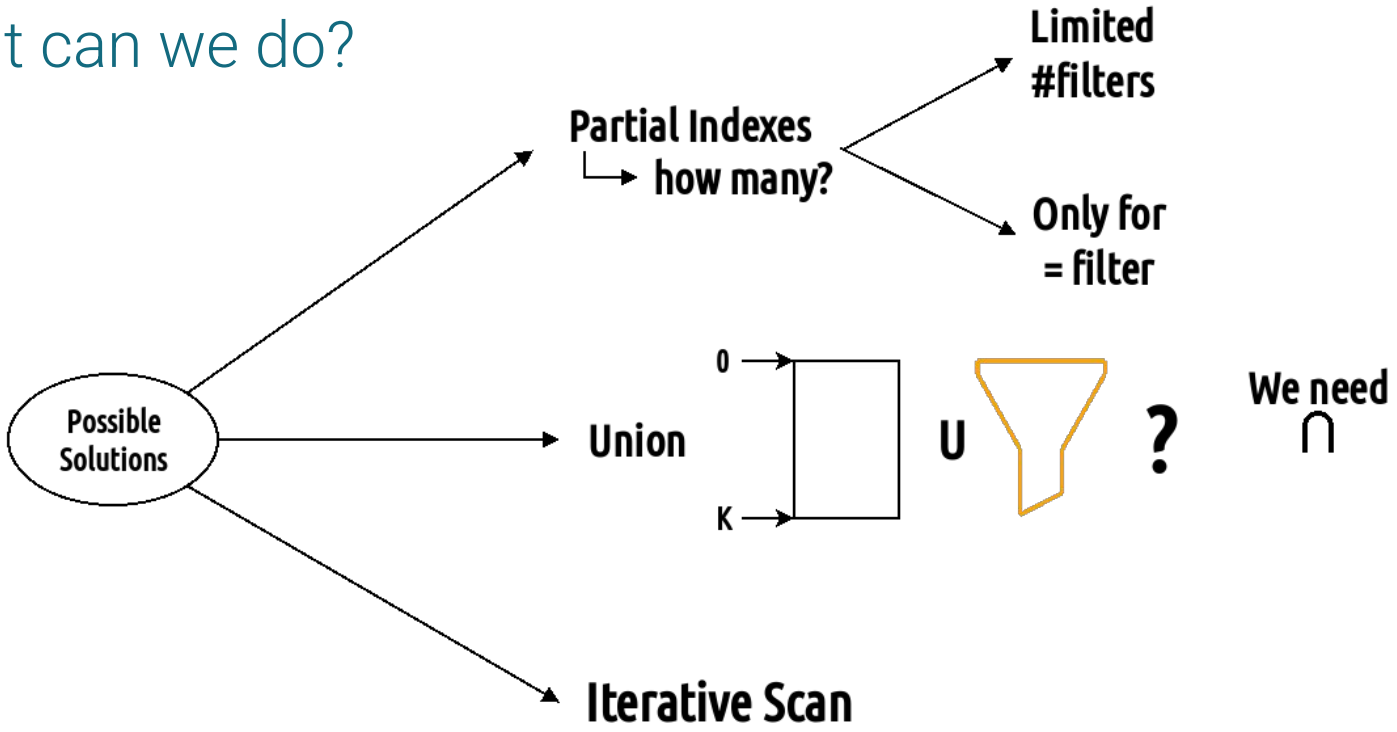
For example; “summer outfit for Thailand trip under €50”

Summer outfit for Thailand - requires semantic search because...

<€50 - is a filtering.

©EDB 2025 — ALL RIGHTS RESERVED.

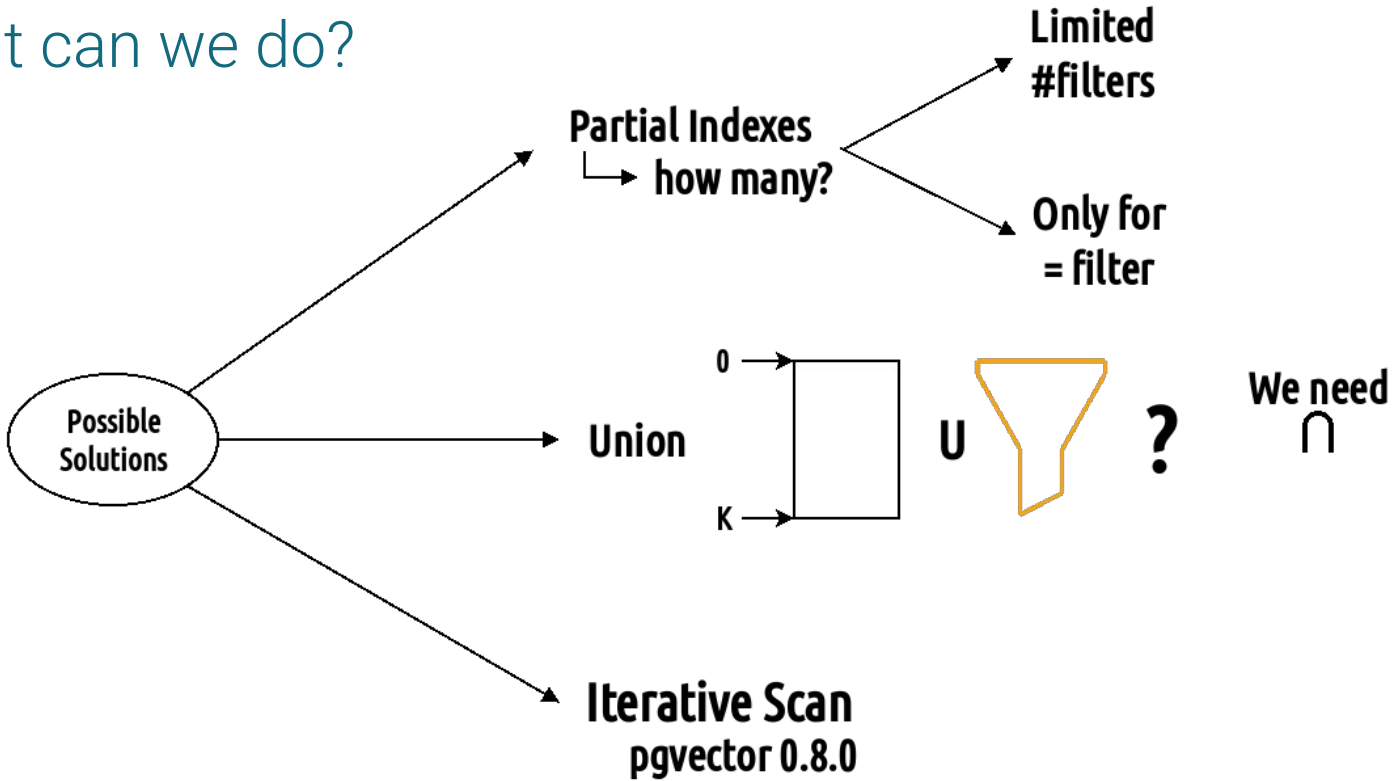
What can we do?



EDB Postgres for the AI Generation

©EDB 2025 — ALL RIGHTS RESERVED.

What can we do?



©EDB 2025 — ALL RIGHTS RESERVED.

Hybrid Search: using pgvector

“summer outfit for Thailand trip under €50”

Indexes: HNSW & IVFFlat are Approximate Nearest Neighbor (ANN) indexes.

Standard ANN index scans (especially HNSW in pgvector) often struggle to efficiently apply WHERE clause filters during the nearest neighbor search.

Common Issue: Retrieve Top K vector neighbours -> Then apply metadata filters.
Result: The desired k value didn’t meet as a result, it may not return an output at all.

©EDB 2025 — ALL RIGHTS RESERVED.

Hierarchical Navigable Small Worlds

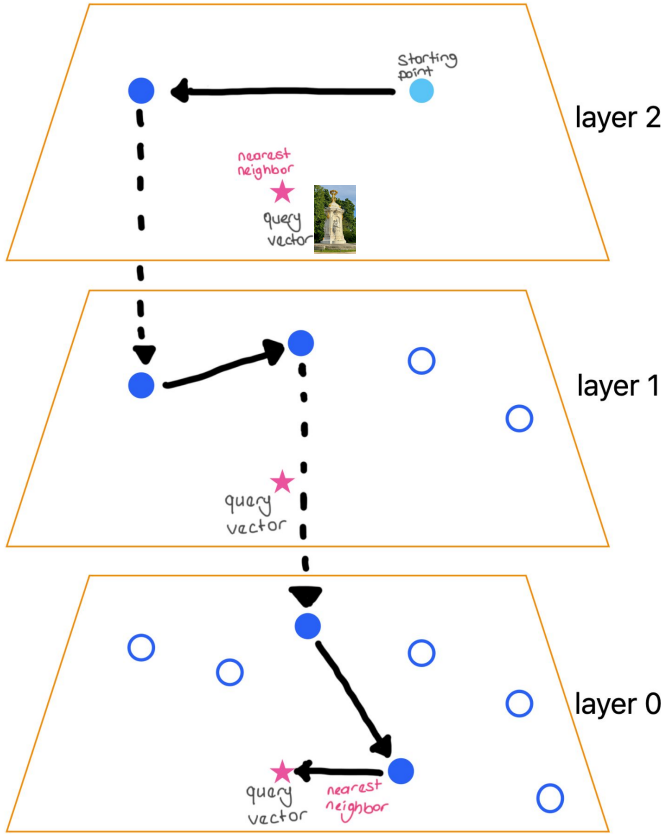
```
CREATE INDEX ON items
USING hnsw (embedding vector_cosine_ops)
WITH (m = 16, ef_construction = 64);
```

```
SET hnsw.ef_search = 100;
```

Hierarchical Navigable Small Worlds

```
CREATE INDEX ON items
USING hnsw (embedding vector_cosine_ops)
WITH (m = 16, ef_construction = 64);
```

```
SET hnsw.ef_search = 100;
```



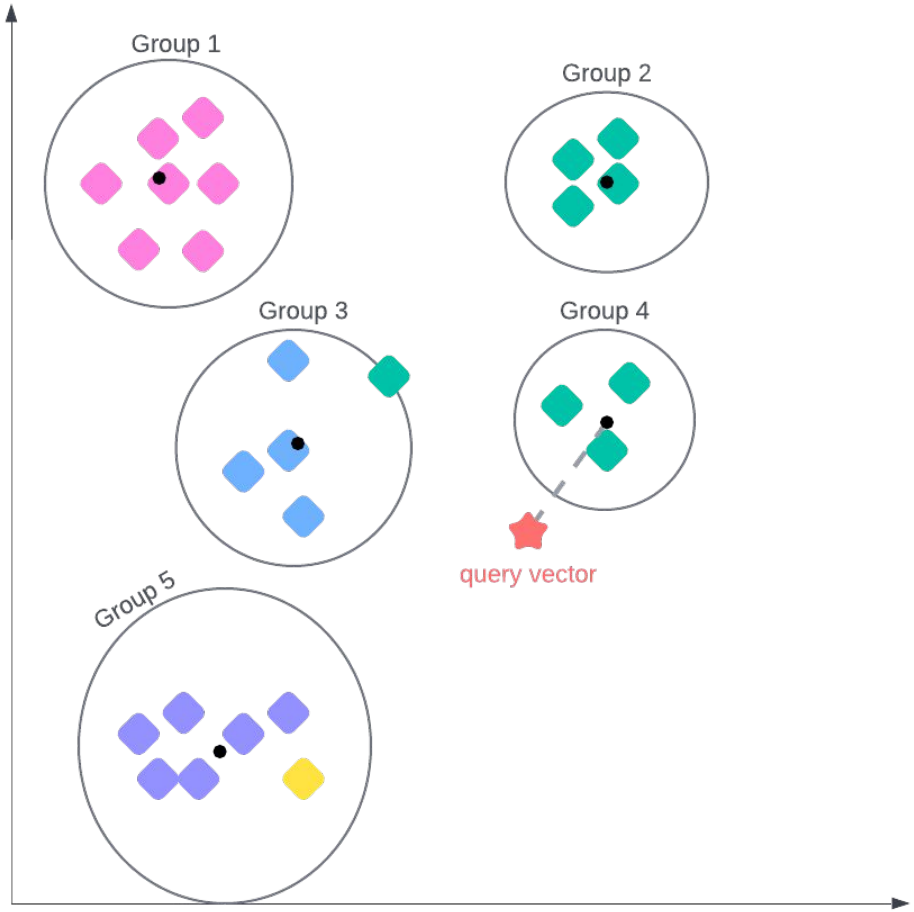
EDB Postgres for the AI Generation

©EDB 2025 — ALL RIGHTS RESERVED.

Inverted File Flat

```
CREATE INDEX ON items
USING ivfflat (embedding vector_cosine_ops)
WITH (lists = 1000);

SET ivfflat.probes = 32
```



©EDB 2025 — ALL RIGHTS RESERVED.

Ideal Scenario

```
-- Select top 5 similar image IDs
SELECT
    img_id,
    (embedding <=> '{text_embeddings}') AS score
    -- Calculate similarity score
FROM
    products_embeddings
WHERE
    gender = '{gender}' -- Filter by gender
ORDER BY
    score
LIMIT 11;
```


©EDB 2025 — ALL RIGHTS RESERVED.

The Scenario I can Use ?

```
-- 1. Filter products by gender
WITH filtered_products AS (
    SELECT img_id, productdisplayname
    FROM products_pgconf
    WHERE gender = '{selected_gender}'
)
-- 2. Find top 100 similar embeddings for filtered products
SELECT
    r.img_id, r.score
FROM
    filtered_products fp
CROSS JOIN LATERAL (
    SELECT img_id, (embedding <=> '{text_embeddings}') AS score
    FROM products_embeddings_pgvector
    ORDER BY score
    LIMIT 100
) AS r
-- 3. Join and get the top 5 most similar
WHERE r.img_id = fp.img_id
ORDER BY r.score LIMIT 11;
```

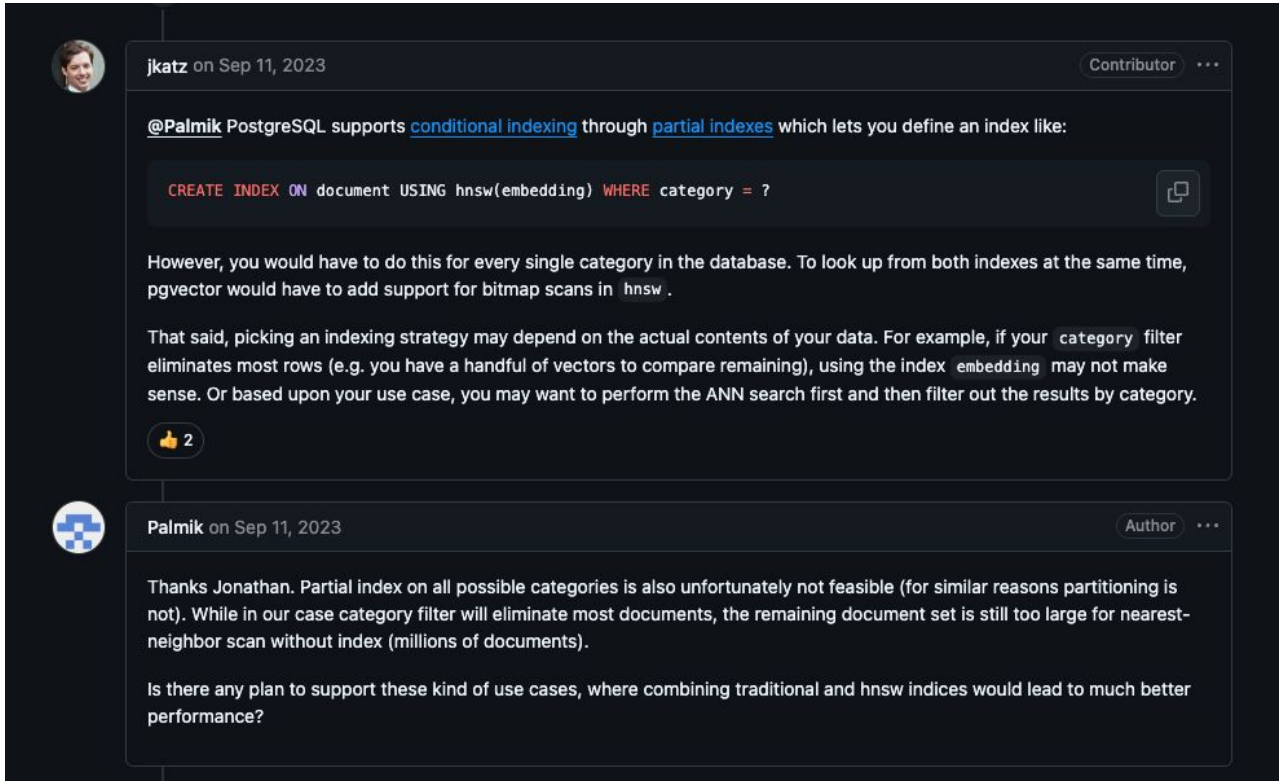

©EDB 2025 — ALL RIGHTS RESERVED.

The Scenario I can Use ?

```
-- 1. Filter products by gender
WITH filtered_products AS (
    SELECT img_id, productdisplayname
    FROM products_pgconf
    WHERE gender = '{selected_gender}'
)
-- 2. Find top 100 similar embeddings for filtered products
SELECT
    r.img_id, r.score
FROM
    filtered_products fp
CROSS JOIN LATERAL (
    SELECT img_id, (embedding <=> '{text_embeddings}') AS score
    FROM products_embeddings_pgvector
    ORDER BY score
    LIMIT 100
) AS r
-- 3. Join and get the top 5 most similar
WHERE r.img_id = fp.img_id
ORDER BY r.score LIMIT 11;
```


©EDB 2025 — ALL RIGHTS RESERVED.

Hybrid Search with Intersection: using pgvector



jkatz on Sep 11, 2023 — Contributor

@Palmik PostgreSQL supports conditional indexing through partial indexes which lets you define an index like:

```
CREATE INDEX ON document USING hnsw(embedding) WHERE category = ?
```

However, you would have to do this for every single category in the database. To look up from both indexes at the same time, pgvector would have to add support for bitmap scans in `hnsw`.

That said, picking an indexing strategy may depend on the actual contents of your data. For example, if your `category` filter eliminates most rows (e.g. you have a handful of vectors to compare remaining), using the index `embedding` may not make sense. Or based upon your use case, you may want to perform the ANN search first and then filter out the results by category.

👍 2

Palmik on Sep 11, 2023 — Author

Thanks Jonathan. Partial index on all possible categories is also unfortunately not feasible (for similar reasons partitioning is not). While in our case category filter will eliminate most documents, the remaining document set is still too large for nearest-neighbor scan without index (millions of documents).

Is there any plan to support these kind of use cases, where combining traditional and hnsw indices would lead to much better performance?

©EDB 2025 — ALL RIGHTS RESERVED.

Hybrid Search: using pgvector v0.8.0

Iterative Index Scans

With approximate indexes, queries with filtering can return less results since filtering is applied *after* the index is scanned. Starting with 0.8.0, you can enable iterative index scans, which will automatically scan more of the index until enough results are found (or it reaches `hnsw.max_scan_tuples` or `ivfflat.max_probes`).

Iterative scans can use strict or relaxed ordering.

Strict ensures results are in the exact order by distance

```
SET hnsw.iterative_scan = strict_order;
```

Relaxed allows results to be slightly out of order by distance, but provides better recall

```
SET hnsw.iterative_scan = relaxed_order;
# or
SET ivfflat.iterative_scan = relaxed_order;
```

With relaxed ordering, you can use a materialized CTE to get strict ordering

```
WITH relaxed_results AS MATERIALIZED (
    SELECT id, embedding <-> '[1,2,3]' AS distance FROM items WHERE category_id = 123 ORDER BY
) SELECT * FROM relaxed_results ORDER BY distance;
```

For queries that filter by distance, use a materialized CTE and place the distance filter outside of it for best performance (due to the current behavior of the Postgres executor)

```
WITH nearest_results AS MATERIALIZED (
    SELECT id, embedding <-> '[1,2,3]' AS distance FROM items ORDER BY distance LIMIT 5
) SELECT * FROM nearest_results WHERE distance < 5 ORDER BY distance;
```


©EDB 2025 — ALL RIGHTS RESERVED.

VectorChord Vchord+Vchord_BM25

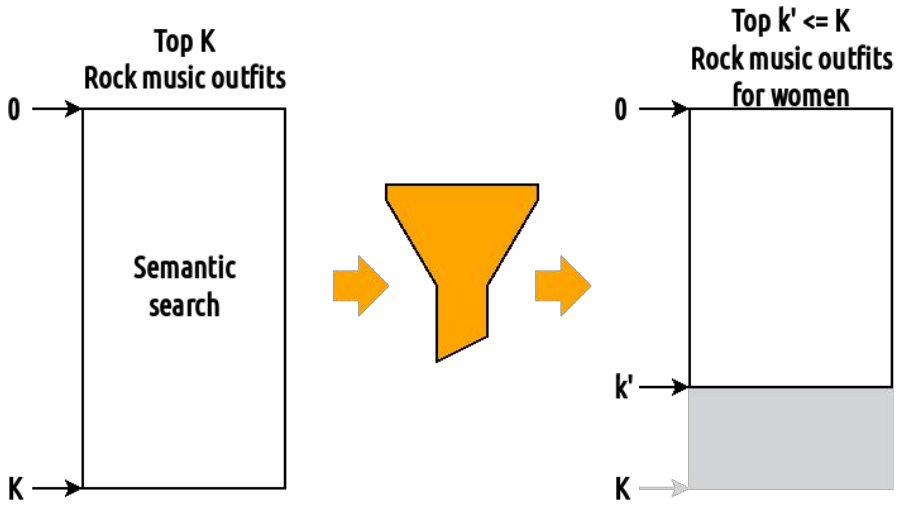
- VectorChord-BM25 is a PostgreSQL extension for keyword search. It not only implements BM25 ranking but also includes a tokenizer and a Block-WeakAnd index to improve speed.
- Semantic Search is utilizing RabitQ algorithm.
  - RabitQ is a quantization algorithm for high-dimensional spaces, designed to improve the storage and retrieval efficiency of high-dimensional data, such as embedded vectors.

©EDB 2025 — ALL RIGHTS RESERVED.

Iterative Scan

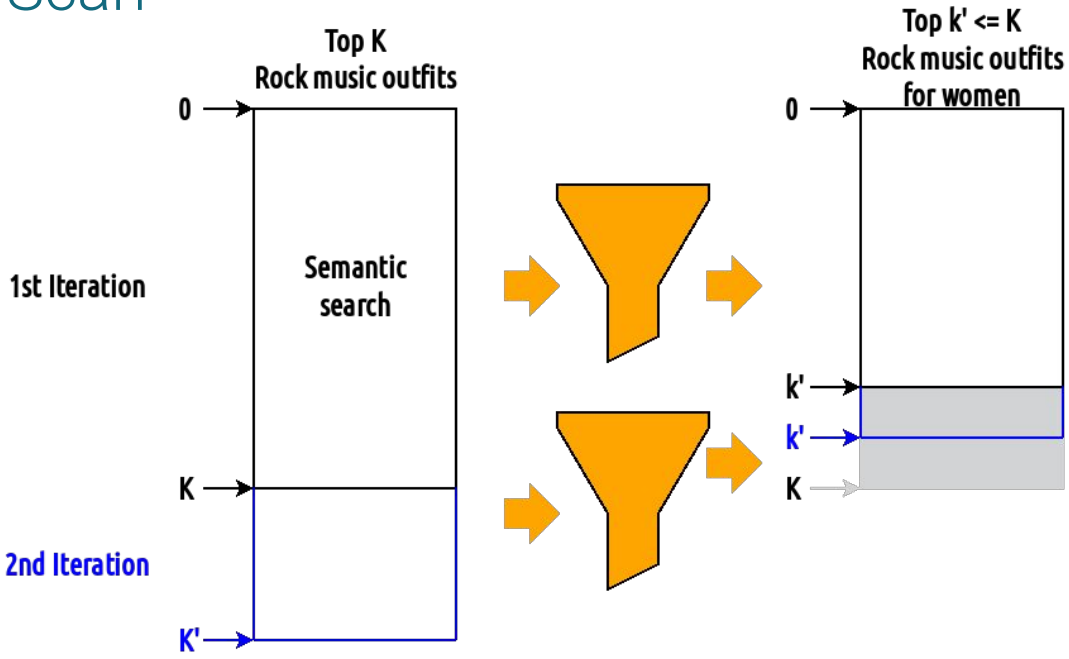
©EDB 2025 — ALL RIGHTS RESERVED.

Iterative Scan



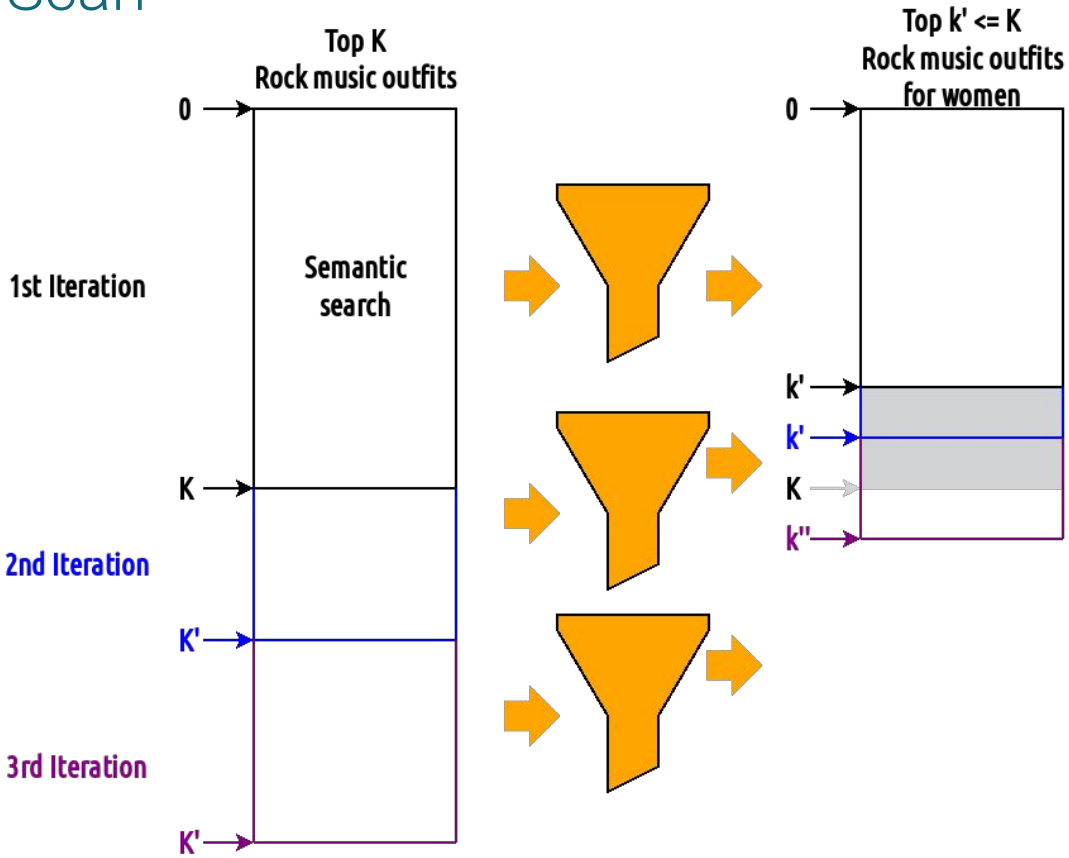
©EDB 2025 — ALL RIGHTS RESERVED.

Iterative Scan



©EDB 2025 — ALL RIGHTS RESERVED.

Iterative Scan



EDB
Postgres for the AI Generation

©EDB 2025 — ALL RIGHTS RESERVED.

Closing words

- Full text is not enough
- Vector search is not enough
- Existing tools only solves part of the problem or too expensive (and complicated)
- Iterative scans looks promising yet not cost-efficient

©EDB 2025 — ALL RIGHTS RESERVED.

References

1. Hands on LLMs - Jay Alammar & Maarten Grootendorst
2. https://github.com/pgvector/pgvector/issues/259
3. https://github.com/pgvector/pgvector?tab=readme-ov-file#iterative-index-scans
4. https://github.com/pgvector/pgvector/issues/301
5. https://blog.vectorchord.ai/hybrid-search-with-postgres-native-bm25-and-vectorchord

©EDB 2025 — ALL RIGHTS RESERVED.

Any Questions?

Bilge Ince

@bilge.bsky.social

bilge.ince@enterprisedb.com

Boriss Mejías

@tchorix.bsky.social

boriss.mejias@enterprisedb.com

©EDB 2025 — ALL RIGHTS RESERVED.