
Querying the Vector Store

1. Overview of AI Workflows	2. Why Postgres as a vector store	3. Storing and managing vectors	4. Querying the vector store
• Look at high-level architecture - LLMs, vector stores and JSON • Look at key vocabulary and concepts (embeddings, vectors, hybrid queries, etc.)	• What is a vector store? Key concepts and use cases. • Why Postgres and how does it compare with other market tools • Setting up Postgres with vector capabilities (pgvector) • **Lab**: Install and configure Postgres using Docker	• Generating embeddings: Overview of tools and workflows • Storing and organizing embeddings in Postgres • Strategies for handling large datasets including chunking • Dense and sparse vectors • **Lab:** Generate embeddings for a dataset and store them	• Techniques for similarity search: k-NN, cosine similarity • Using indexes to optimize vector queries • Reranking results • **Lab:** Query stored vectors to retrieve similar items (document/image search)
5. Querying LLMs with retrieved data	**6. NoSQL with JSON in Postgres**	**7. Integrating Vector, Relational and JSON Data**	**8. Putting it all together**
• Recap on querying LLMs vis APIs • Best practices for combining vector retrieval with LLM prompts • Prompt configuration parameters (temperature, top-k, etc) • **Lab:** Build a pipeline where vector store results enhance LLM responses (context-aware Q&A, etc)	• Overview of JSON/JSONB support in Postgres • Querying JSONB data with SQL • Indexing JSONB data for performance • **Lab:** Design a schema mixing vector, relational and JSONB data for a sample project	• Building hybrid queries to power advanced workflows • **Case study:** Combining embeddings, metadata (relational) and configurations (JSON) • **Lab:** Implement a hybrid query to support a sample AI use case	• Full stack pipeline demo: Retrieve data, query the LLM and return results • Debugging and optimising the workflow • Spotlight on LLM frameworks • **Lab:** Build a working application combining all elements

Plan for the session

- Introduction to querying
- Cosine Similarity
- k-Nearest Neighbours (k-NN)
- Other similarity measures
- Using Indexes for Optimised Queries
- IVFFlat Index
- HNSW Index

Introduction to querying

- A vector query retrieves similar items based on vector distance or similarity
- Common use cases:
  - Semantic Search: Retrieve documents or data similar in meaning
  - Recommendations: Suggest similar products, songs or content
  - Clustering: Group items with similar embeddings
- There are four index operators:
  - **vector_cosine_ops**: Cosine similarity (lower = more similar) [<=>]
  - **vector_l2_ops**: Euclidean distance [<->]
  - **vector_inner_ops**: Inner product [<#>]
  - **vector_manhattan_ops**: Manhattan distance [<+>]

Cosine Similarity

5

Cosine similarity measures the cosine of the angle between two non-zero vectors in a multi-dimensional space. It quantifies how similar two vectors are, regardless of their magnitude, by assessing the orientation between them.

$$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

```
SELECT name, embedding <=> '[query_vector]'
  AS similarity
  FROM items;
```

•cosine similarity is widely used in various fields, including machine learning and information retrieval.
•Highlight its effectiveness in high-dimensional spaces, such as text documents represented by word embeddings.
•Mention that while cosine similarity measures orientation, it does not account for vector magnitude, which can be a consideration in certain applications.

k-Nearest Neighbours (k-NN)

- **Step 1:** Compute the similarity (or distance) between the query vector and all other vectors.
- **Step 2:** Sort by similarity/distance.
- **Step 3:** Return the top k results.

```
SELECT name, embedding ⟺ '[query_vector]'
  AS similarity
  FROM items
  ORDER BY similarity
  LIMIT 5;
```

Similarity Measure	Native Support in pgvector	SQL Example
Cosine Similarity	✅Yes	<=>
Euclidean Distance	✅Yes	<->
Inner Product	✅Yes	<#>
Manhattan Distance	✅Yes	<+>
Jaccard Similarity	✅Yes	<%>
Hamming Distance	✅Yes	<~>
Mahalanobis Distance	❌No (External computation)	Use Python/NumPy or other external tools

```
SELECT name, embedding <#> '[query_vector]'
  AS similarity
  FROM items
  ORDER BY similarity
  LIMIT 5;
```

```
SELECT name, embedding ⟷ '[query_ve
  AS similarity
  FROM items
  ORDER BY similarity
  LIMIT 5;
```

•**Common Usage:** Text embeddings (e.g., word/document similarity), high-dimensional spaces.
•**SQL Operator in pgvector:** <=>

2. Euclidean Distance

•**What it does:** Measures the straight-line distance between two vectors in a multidimensional space.
•**Formula:** Euclidean Distance=∑i=1n(ai−bi)2\text{Euclidean Distance} = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}Euclidean Distance=i=1∑n(ai−bi)2
•**Range:** [0, ∞] (Lower is more similar).
•**Common Usage:** Image processing, clustering, spatial data.
•**SQL Operator in pgvector:** <->

3. Inner Product (Dot Product)

•**What it does:** Measures the degree of overlap between two vectors, considering both magnitude and direction.
•**Formula:** Inner Product=∑i=1nai·bi\text{Inner Product} = \sum_{i=1}^n a_i \cdot b_iInner Product=i=1∑nai·bi
•**Range:** [-∞, ∞] (Higher is more similar).
•**Common Usage:** Recommendation systems, ranking items, scenarios where magnitude matters.
•**SQL Operator in pgvector:** <#>

4. Manhattan Distance (L1 Norm)

•**What it does:** Computes the sum of the absolute differences between vector components.
•**Formula:** Manhattan Distance=∑i=1n|ai−bi|\text{Manhattan Distance} = \sum_{i=1}^n |a_i - b_i|Manhattan Distance=i=1∑n|ai−bi|
•**Range:** [0, ∞] (Lower is more similar).
•**Common Usage:** Simpler datasets, when absolute differences are meaningful (e.g., tabular data).

5. Jaccard Similarity

•**What it does:** Measures the similarity between two sets by dividing the size of their intersection by the size of their union.
•**Formula:** Jaccard Similarity=|A∩B||A∪B|\text{Jaccard Similarity} = \frac{|A \cap B|}{|A \cup B|}Jaccard Similarity=|A∪B||A∩B|
•**Range:** [0, 1] (Higher is more similar).
•**Common Usage:** Sparse data, binary features, text comparisons (e.g., bag-of-words).

6. Hamming Distance

•**What it does:** Counts the number of positions where two vectors differ.
•**Range:** [0, n] (Lower is more similar).
•**Common Usage:** Binary data, error detection (e.g., DNA sequence comparison).

7. Mahalanobis Distance

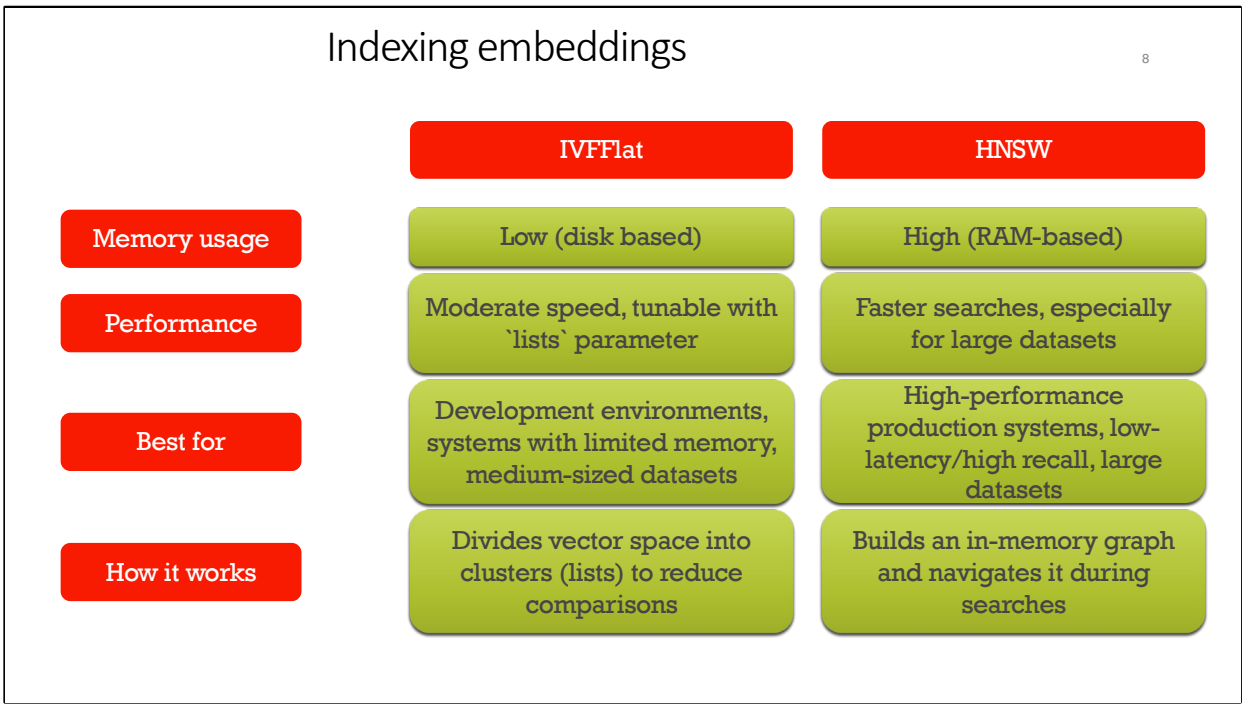
•**What it does:** Measures the distance between a point and a distribution, considering correlations between variables.
•**Formula:** Mahalanobis Distance=(x−μ)⊤Σ−1(x−μ)\text{Mahalanobis Distance} = \sqrt{(\mathbf{x} - \mathbf{\mu})^\top \mathbf{\Sigma}^{-1} (\mathbf{x} - \mathbf{\mu})}Mahalanobis Distance=(x−μ)⊤Σ−1(x−μ)
•**Range:** [0, ∞] (Lower is more similar).
•**Common Usage:** Anomaly detection, multidimensional data with correlations.

8. Pearson Correlation

•**What it does:** Measures the linear correlation between two vectors.
•**Range:** [-1, 1] (Higher absolute values are more correlated).
•**Common Usage:** Time series data, when relationships between variables matter.

Choosing the Right Similarity Metric:

1. **High-Dimensional Data:** Cosine similarity works well as it ignores magnitude differences.
2. **Spatial Data:** Euclidean distance is intuitive for geometric contexts.
3. **Sparse Data:** Jaccard similarity or Manhattan distance are preferred.
4. **Dense Representations (e.g., embeddings):** Cosine similarity, inner product, or Euclidean distance.
5. **Correlated Data:** Mahalanobis distance is more accurate when variables are interrelated.



Indexing vector data improves the performance of similarity searches by reducing the number of comparisons needed. Instead of performing a brute-force linear search across all vectors, an index allows the database to narrow down the search space efficiently.

IVFFlat (Inverted File Index with Flat Quantization)
HNSW (Hierarchical Navigable Small World Graph)

- For HNSW the index is persisted to disc and loaded to ram on db restart

Do I need to rebuild the index when new data is inserted?
Answer:
No, the index is automatically updated when new rows are inserted, updated, or deleted. However:
•Frequent updates may slow down operations if the dataset is very large.
•For bulk inserts, consider creating the index after the data is loaded to optimize index creation time.

Are indexes persistent across database restarts?

Answer:
Yes, both IVFFlat and HNSW indexes are stored on disk and persist across database or container restarts. However, HNSW indexes are loaded into memory on first access after a restart, which may introduce a slight delay.

How can I tune IVFFlat for better performance?
Answer:
•lists Parameter:

- Specifies the number of partitions (clusters) in the vector space.
- More lists improve search accuracy but increase memory usage and index creation time.

How do I measure the performance of my index?
Answer:
•Use EXPLAIN ANALYZE to measure query execution time.
•Monitor resource usage (CPU, memory) during queries.
•Compare query performance with and without the index to validate improvements.

Are there any limitations to using vector indexes?
Answer:
•Vector indexes can consume significant memory (especially HNSW) for large datasets.
•Index tuning (parameters like lists, M, and ef_search) requires experimentation based on the dataset.
•Indexes add overhead during write operations (inserts, updates, deletes).

How can I tune HNSW for better performance?
Answer:
•M Parameter:

- Controls the number of edges per node in the graph. Higher M improves accuracy but increases memory usage.

•ef_construction Parameter:

- Determines the effort during index creation. Higher values improve recall but slow down index building.

•ef_search Parameter:

- Controls search accuracy. Higher values improve recall at the cost of slower searches.

How can I check if an index is being used in a query?

Answer:
Use the EXPLAIN or EXPLAIN ANALYZE command to check the query execution plan:

Can I drop and recreate an index without impacting the data?
Answer:
Yes, you can safely drop and recreate an index without modifying the underlying data:

Demo:
Generating indexes

Lab


Questions
?