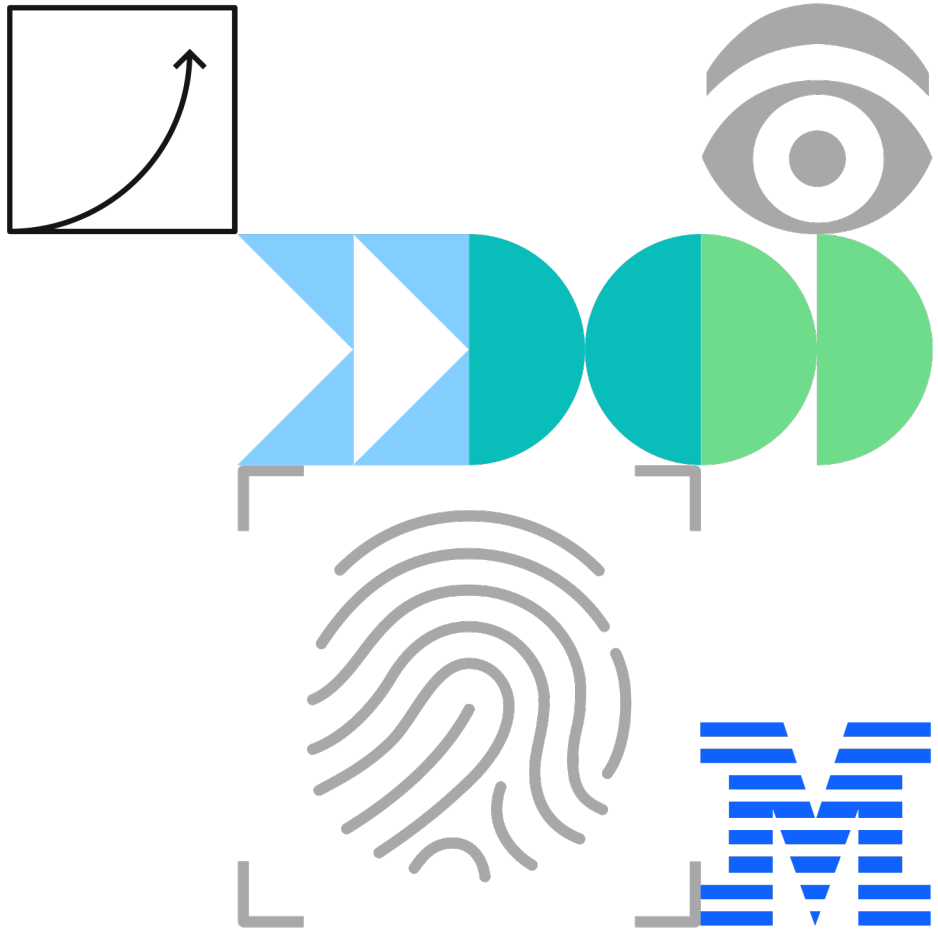
Central Canada Db2 Users Group

CCDUG | CenCan



Accelerating AI Workloads in Db2 v12.1.5.0: Introducing High-Performance Vector Indexes

May 13th 2026

Christian Garcia-Arellano
Zach Hoggard
Christopher Stojanovski

Notices and disclaimers

- This document is distributed “as is” without any warranty, either express or implied. In no event shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.
- Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.
- Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM.
- Not all offerings are available in every country in which IBM operates.
- Any statements regarding IBM’s future direction, intent or product plans are subject to change or withdrawal without notice.
- IBM, the IBM logo, and ibm.com are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at: www.ibm.com/legal/copytrade.shtml.
- Certain comments made in this presentation may be characterized as forward looking under the Private Securities Litigation Reform Act of 1995.
- Forward-looking statements are based on the company’s current assumptions regarding future business and financial performance. Those statements by their nature address matters that are uncertain to different degrees and involve a number of factors that could cause actual results to differ materially. Additional information concerning these factors is contained in the Company’s filings with the SEC.
- Copies are available from the SEC, from the IBM website, or from IBM Investor Relations.
- Any forward-looking statement made during this presentation speaks only as of the date on which it is made. The company assumes no obligation to update or revise any forward-looking statements except as required by law; these charts and the associated remarks and comments are integrally related and are intended to be presented and understood together.

Agenda

- Why vector indexing matters now
- Exact vs approximate similarity search
- Vector indexing in Db2
- Tuning speed vs recall
- DBA deployment considerations

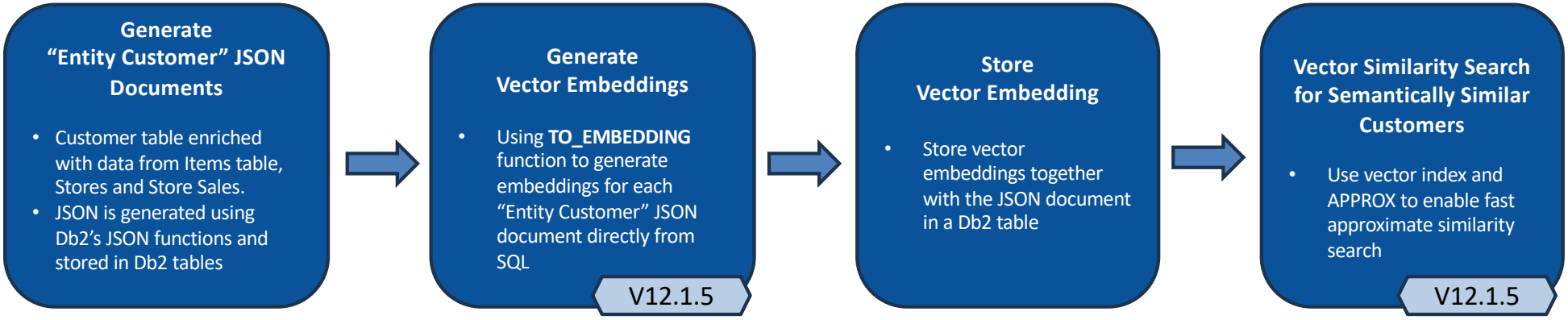
Why this matters now?

Embeddings are becoming a core building block for enterprise applications

Vector Similarity in Db2: End to End Use Case

Entity Customer Dataset

- The ”Entity Customer” data set is created from relational tables to enable semantic search
- Represents customers profiles enriched with comprehensive transactional and behavioral data to create a holistic "entity customer" representation, including:
  - Top 3 Purchased Items
  - Top 3 Stores
  - Monthly Spending Patterns
- Semantic search enables search of customers with specific buying pattern, spending patterns, store visit patterns, etc.



- For our tests we use the TPC-DS benchmark data set that represents a wholesale retail business
- We use the SLATE 125 model to generate 768-dimensional Float32 vectors that represent each “Entity Customer”

https://github.com/IBM/IBMDb2-Entity-Customer-Semantic-Search

Vector Similarity Search Queries

The base of similarity search queries have 4 elements:

1. **A table with vector embeddings generated using an LLM model**
2. **A query vector embedding generated using the same LLM model as the data set**
3. **The use of the VECTOR_DISTANCE function distance in the ORDER BY clause with the corresponding distance metric to compare the query vector with the data set**
4. **A FETCH FIRST clause to indicate the (K) number of similar vectors to return**

Here is an example:

```
SELECT
      id, text, metadata,
      VECTOR_DISTANCE(embedding, {embedding}, COSINE) as distance
FROM {self.table_name}
ORDER BY distance
FETCH FIRST {k} ROWS ONLY
```

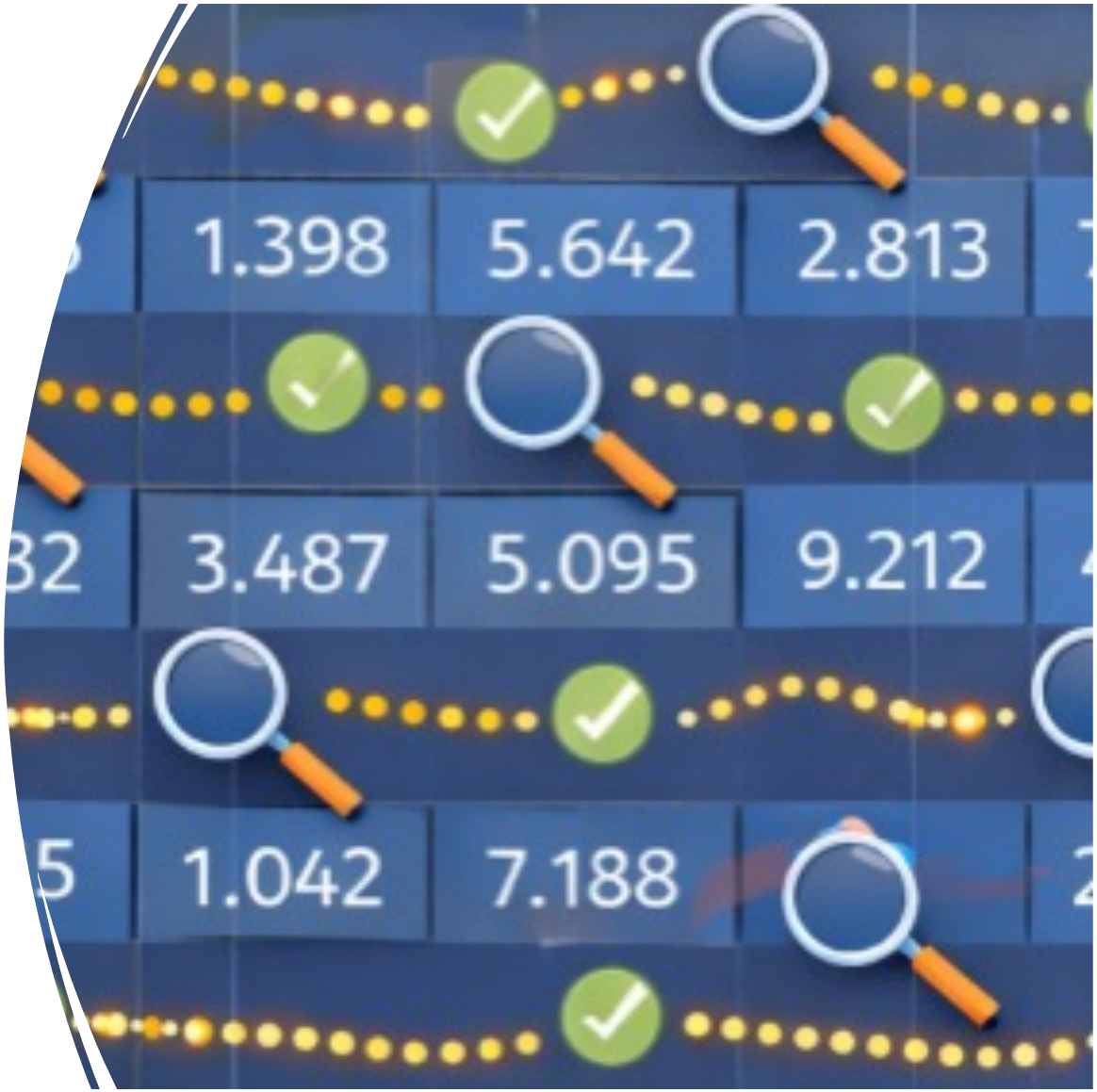
This is the syntax for the VECTOR_DISTANCE function:

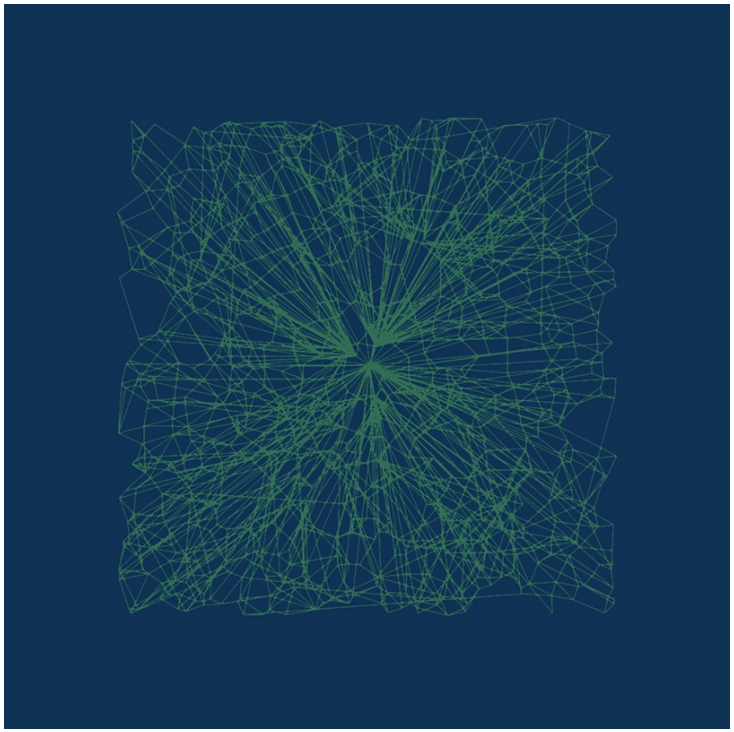
```
VECTOR_DISTANCE( <vector1>, <vector2>, <distance_metric> )
```

Where distance metric can be COSINE, EUCLIDEAN, EUCLIDEAN_SQUARED, HAMMING or MANHATTAN

The problem: exact vector search gets expensive at scale

- good for smaller or filtered workloads
- expensive for large candidate sets
- conventional indexes are not built for this problem

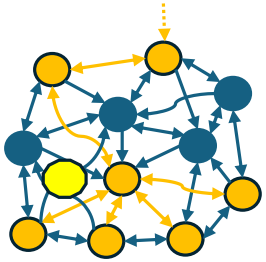




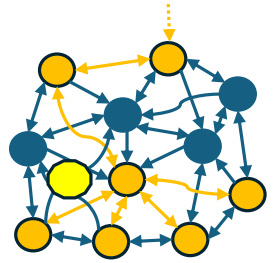
Vector Indexing in Db2: Powering Approximate Similarity Search

- Vector indexing helps reduce the latency of searches for high-volume datasets.
- Db2 vector indexing is a graph-based index – based on the DiskANN implementation.
  - Designed for efficiency in small to very large data sets.
  - Small memory footprint, leverages random disk accesses to navigate the graph.
  - Fast search at the expense of providing approximate answers.

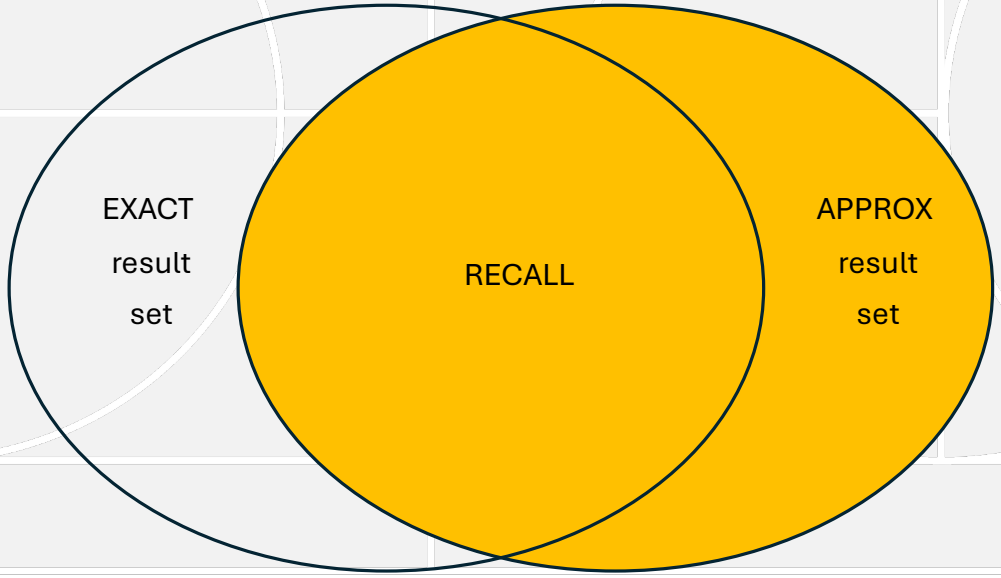
Speed vs Quality Trade Off: Exact vs Approx

	EXACT search 	APPROX search 
Search Method	Exhaustive (scan all vectors)	Graph traversal
Latency	High (scales with data size)	Low (sub-linear)
Result Quality	100% accurate (true nearest)	High recall (near neighbors)
Best-Fit Use Cases	Small datasets, strict accuracy, heavy filtering	Large-scale, real-time systems

What is Recall?



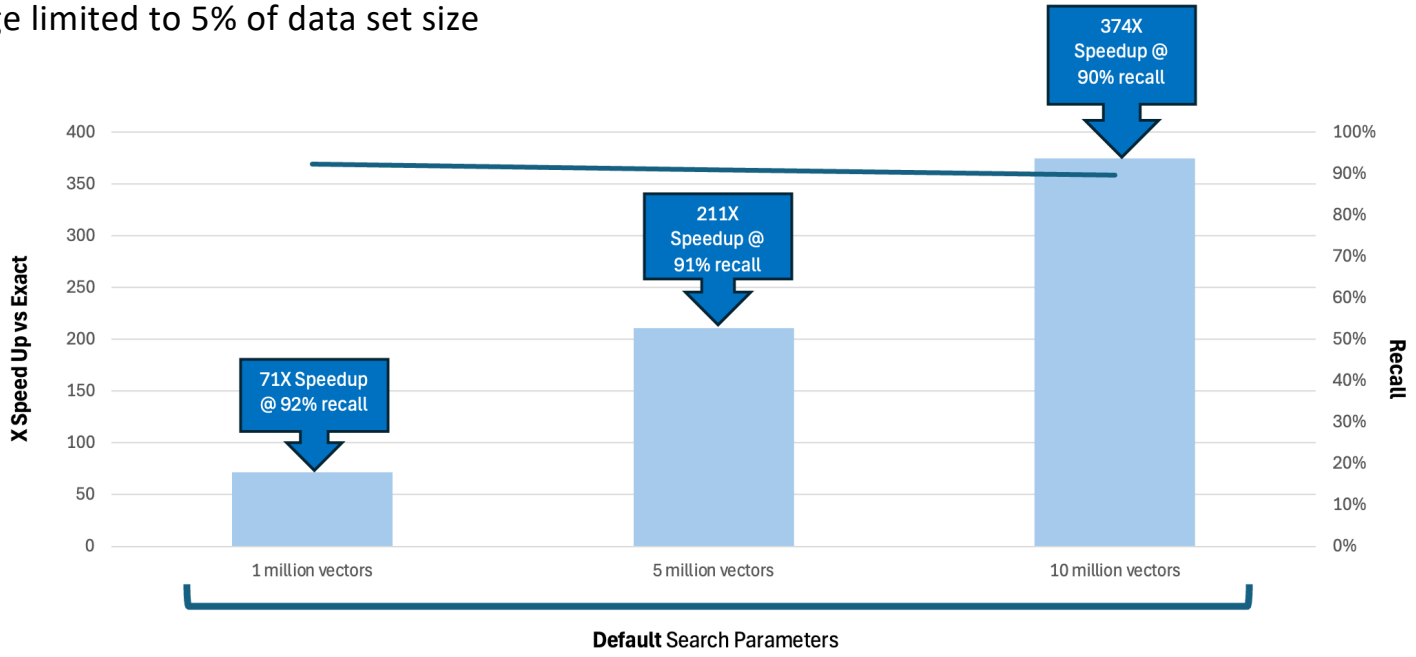
Recall tells us the percentage of true nearest neighbors were found by the approximate search



Vector Similarity Search Indexing Insights #1

- Db2 Vector Indexing **search performance differential improves as data set grows**
  - ✗ Exact search grows linearly with data set size, as it uses brute force approach that needs to scan all data and compute all distances

    Vector Search grows logarithmically with data set size, as graph navigation absorbs data set growth, reducing the number of random IO operations and distance computations.
- Approximate search accuracy remains stable, with minor degradation as data set grows
- Resource usage limited to 5% of data set size



Entity Customer data set: 1, 5 and 10 million vectors, 768 float32 dimensions, embeddings generated using SLATE-125 model from JSON documents sourced from TPC-DS relational tables

Cheat Sheet for Vector Indexing in Db2

- Index Creation

```
CREATE VECTOR INDEX annidx_1 ON tab1 (vector_column) WITH DISTANCE EUCLIDEAN
   COMPRESSED VECTORS IN compvecttablespace;

RUNSTATS ON TABLE tab1 AND INDEXES ALL;
```

- Similarity Search Query that can match the index – cost-based decision

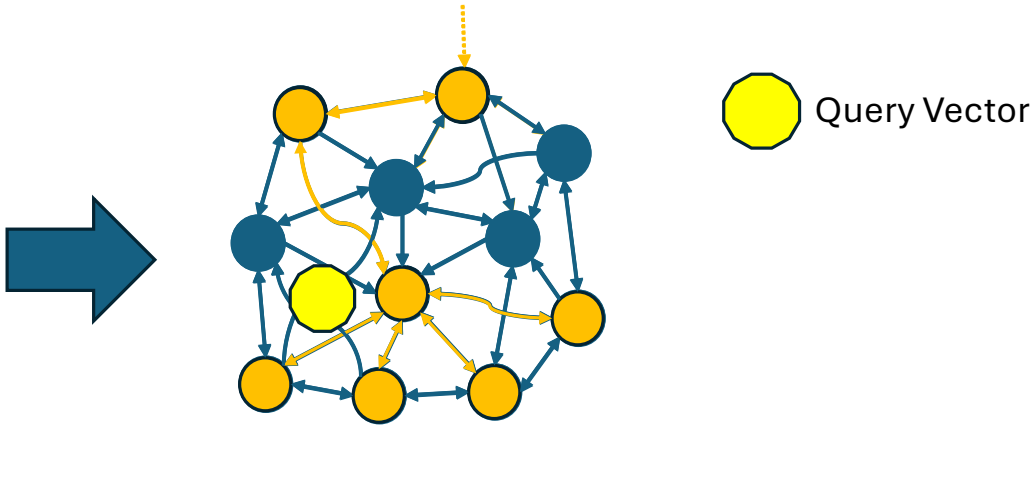
```
SELECT columnA, columnB,
       VECTOR_DISTANCE(vector_column, {embedding}, {dist_fcn}) AS distance
FROM {self.table_name}
ORDER BY distance
FETCH APPROX FIRST {k} ROWS ONLY;
```

- Similarity Search Query that skips the index

```
SELECT columnA, columnB,
       VECTOR_DISTANCE(vector_column, {embedding}, {dist_fcn}) AS distance
FROM {self.table_name}
ORDER BY distance
FETCH EXACT FIRST {k} ROWS ONLY;
```

How does the vector index search work conceptually?

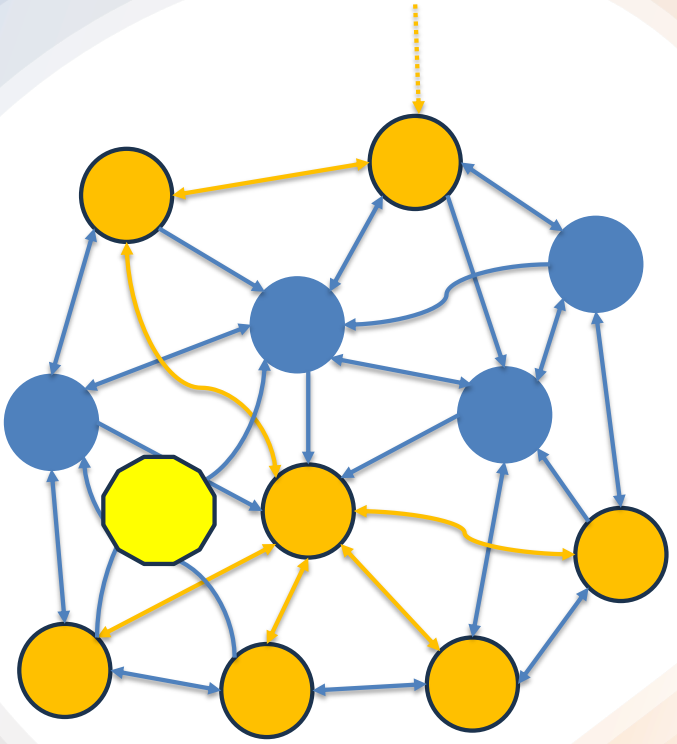
Base Table	
ID	**Vector**
1	[1.2, 2.3,3.1,...]
2	[4.5, 1.2, 0.9, ...]
3	[0.8, 3.5, 4.2, ...]
4	[2.9, 4.1, 1.5, ...]
...	...



Looking under the hood: *The Data Model*

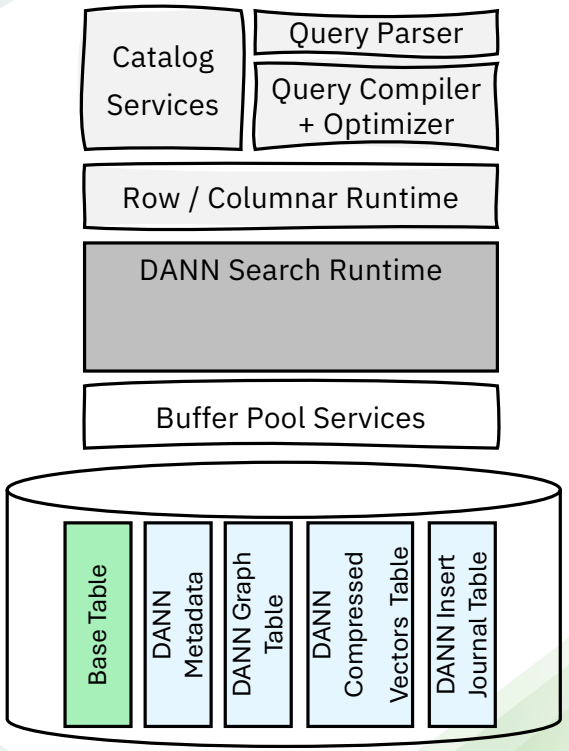
Three key components:

- **Graph Table**: Contains vectors, and neighbors list.
- **Compressed Vector Table**: Contains compressed versions of vectors. Can be stored in a separate tablespace to allow caching in its own buffer pool.
- **Insert Journal Table**: contains newly inserted rows not yet in the graph table.



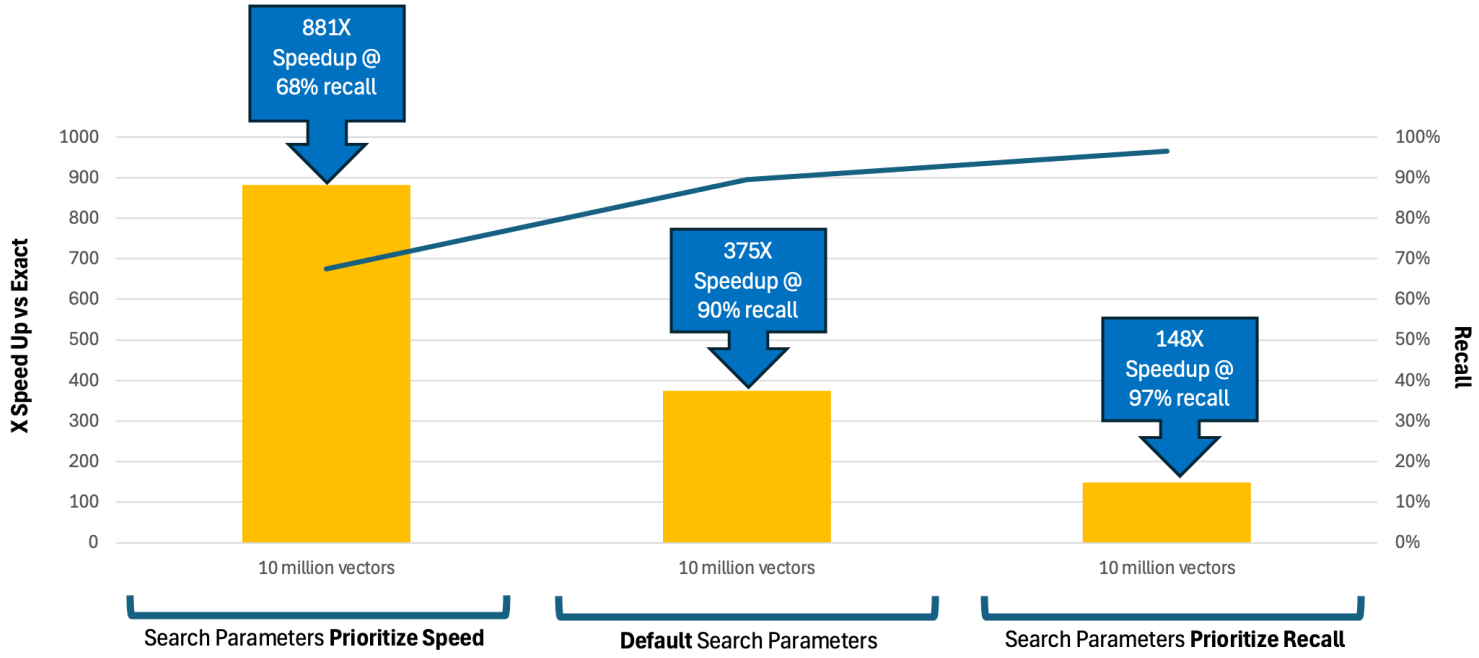
Looking under the hood: Engine Integration

- Parser: Parse new APPROX and EXACT FETCH FIRST clauses
- Query Compiler + Optimizer: Match index and cost decision to determine whether to use ANN index or Brute Force/Exact.
- Catalog Services: index metadata and statistics
- DANN Search Runtime: Implements search algorithm
- Buffer pool: caching of Db2 pages.
- Storage of DANN components in internal Db2 tables



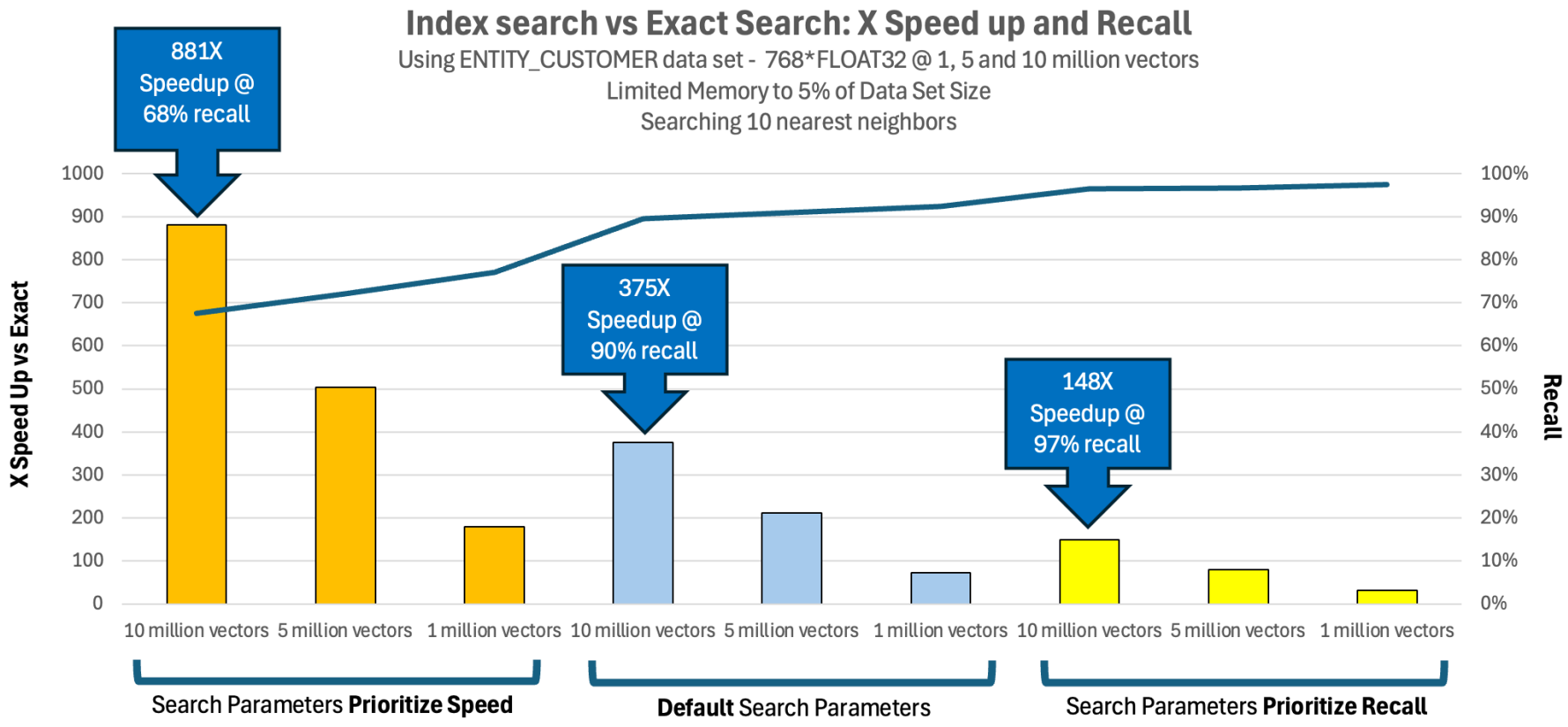
Vector Similarity Search Indexing Insights #2

- Search parameters allows end users to **tune the search behavior** for their needs using **query hints**:
  - 🏎️ Prioritize speed: through reducing the number of neighbors explored, speed improves at the expense of recall.
  - 🎯 Prioritize recall: through increasing the number of neighbors explored, recall improves at the expense of speed.
- Resource usage limited to 5% of data set size



Entity Customer data set: 10 million vectors, 768 float32 dimensions, embeddings generated using SLATE-125 model from JSON documents sourced from TPC-DS relational tables

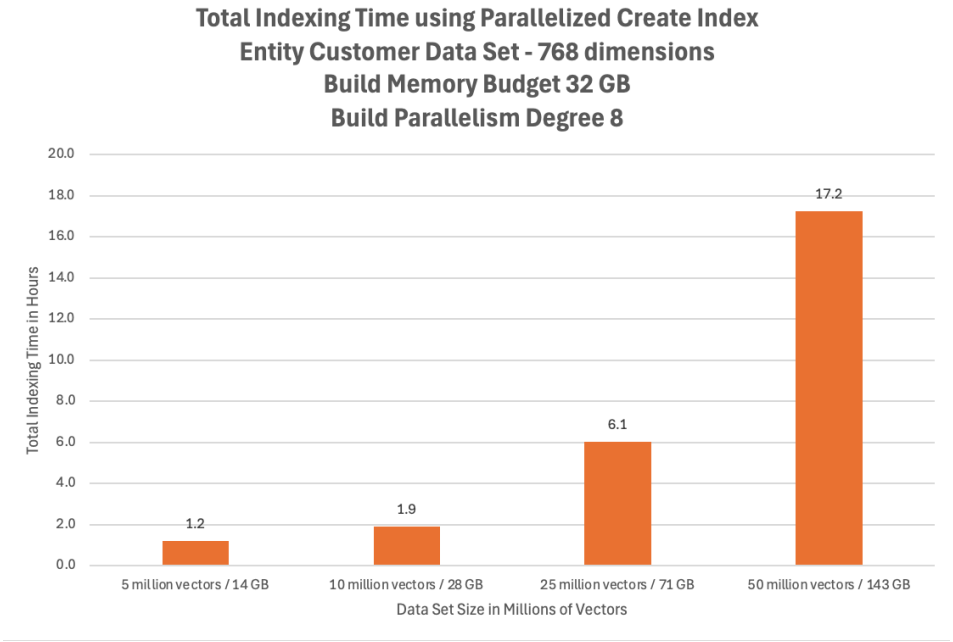
Vector Similarity Search Indexing



Getting to top performance: CREATE VECTOR INDEX

- Build-time tuning matters
  - **BUILD_PARALLELISM** let's you control the degree of parallelism during index creation. Default is **AUTOMATIC**.
  - **BUILD_MEM_BUDGET** let's you control how much memory is available for the index creation.

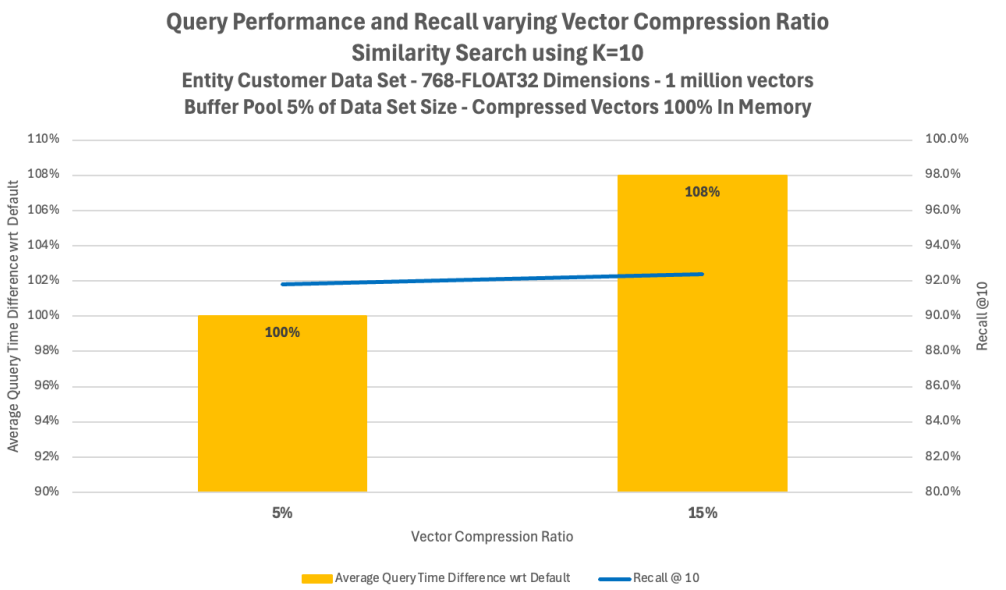
- Important scale message
  - at 50M vectors, the "Entity Customer" full data set size is 143 GB.
  - the 32 GB memory budget is only a fraction of that size.
  - **the index creation does not require the entire data set to fit in memory**.




```
CREATE VECTOR INDEX annidx_1 ON tab1 (vector_column) WITH DISTANCE EUCLIDEAN
   BUILD_PARALLELISM 8 BUILD_MEM_BUDGET 32
   COMPRESSED VECTORS IN compvecttablespace;
```

Getting to top performance: APPROXIMATE SEARCH

- Compressed vectors are key to search performance.
- There are two other build time tuning matters that relate to them:
  - **PCT_COMP_VECT_SIZE** let's you define the compression ratio for vectors. Default is 5 percent of full vector size.
  - **COMPRESSED VECTORS IN** let's you control tablespace and buffer-pool placement to minimize physical reads.




```
CREATE VECTOR INDEX annidx_1 ON tab1 (vector_column) WITH DISTANCE EUCLIDEAN
   PC_COMP_VECT_SIZE 15
   COMPRESSED VECTORS IN compvecttablespace;
```

Conclusion

Db2 offers an integrated end-to-end AI-powered search

Vector data type

- ✓ Zero-friction native integration
- ✓ Full leverage of Db2's query capabilities
- ✓ Enables **combined semantic and relational queries**
- ✓ **Semantic search ecosystem**

Vector indexing

- ✓ Graph-based ANN index
- ✓ Built-in vector compression
- ✓ Integrated with Db2 Buffer Pools
- ✓ Optimized for low latency ANN search

LLM integration

- ✓ Embedding generation directly from SQL
- ✓ Supports local and remote LLMs
- ✓ Seamless integration with Db2 workflows

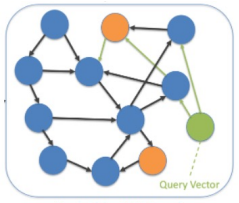
Base relational table

ID	Name	Type	Material	Color	Size	City
1	SportWalk	Walking	Synthetic	White	10	Toronto
2	RunMax	Running	Knit	Red	12	Toronto
3	SpeedFlex	Running	Synthetic	Blue	11	Vancouver

Vectorize data from SQL

ID	Name	Vector
1	SportWalk	[0.1, 0.2, 0.3, ...]
2	RunMax	[0.4, 0.1, 0.7, ...]
3	SpeedFlex	[0.2, 0.5, 0.1, ...]

Graph index for ANN search



Navigable ANN Graph

Vector + relational search

Top 3 Matching Shoe Records

Rank	Name	Type	Color	Score
1	RunMax	Running	Red	0.94
2	SpeedFlex	Running	Blue	0.87
3	SportWalk	Walking	White	0.72

Filtered by: City = 'Toronto', Size = 12

Thanks!

Semantic Product Recommendations in Db2: Pipeline to Search

🔷 Data Pipeline – Vectorizing Product Records in Db2

Product Table

ID	Name	Type ★	Material ★	Color ★	Size	City
1	SportWalk	Walking	Synthetic	White	10	Toronto
2	RunMax	Running	Knit	Red	12	Toronto
3	SpeedFlex	Running	Synthetic	Blue	11	Vancouver

Extract Features

Embedding Model

*e.g., **Granite-Embedding-30m***

Generate Vectors

Vectorized Table

ID	Name	Vector
1	SportWalk	[0.1, 0.2, 0.3, ...]
2	RunMax	[0.4, 0.1, 0.7, ...]
3	SpeedFlex	[0.2, 0.5, 0.1, ...]

Vectorized Shoe Records

Stored in Db2

🟩 Search – Querying with Vector Search in Db2

Query Shoe

"Running, Black, Size 12, Toronto"

Vectorize Query

Query Vector

`[0.2, 0.1, 0.6, ...]`

Via Same Embedding Model

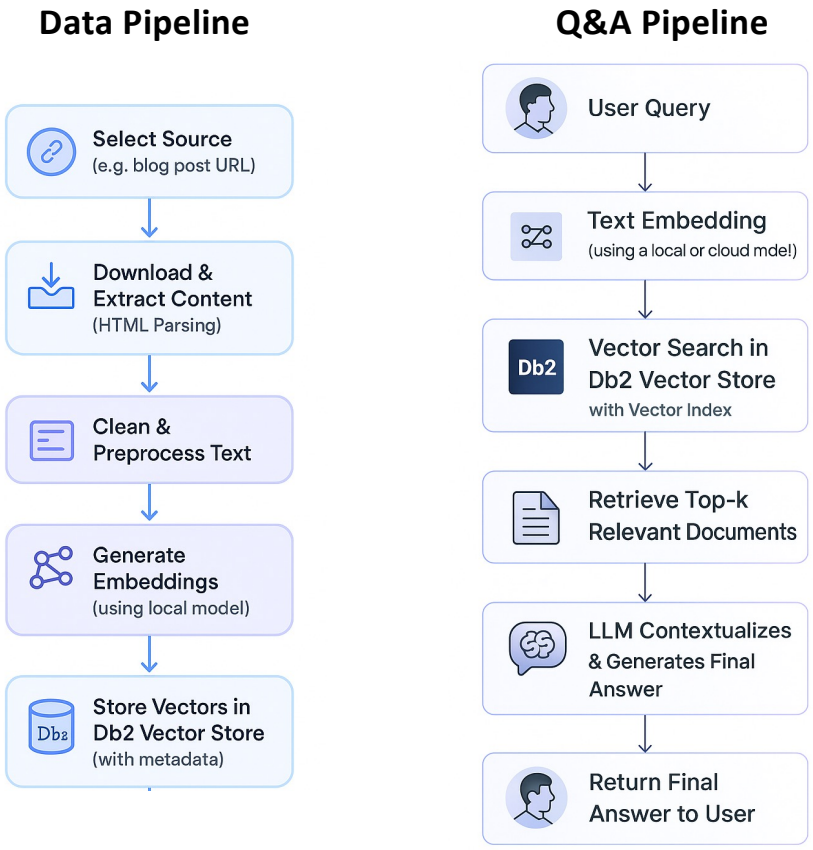
Vector Search in Db2 + Relational Filters (City = 'Toronto', Size = 12)

Top 3 Matching Shoe Records

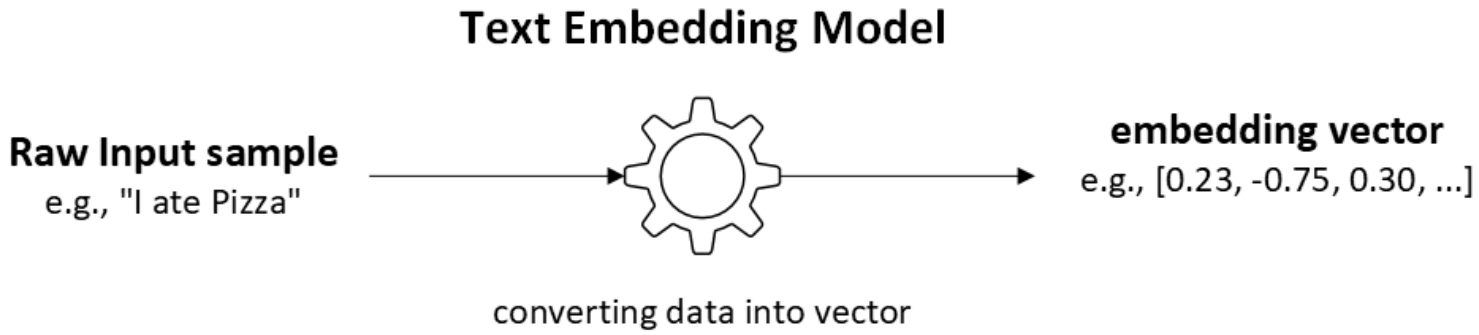
Rank	Name	Type	Color	Score
1	**RunMax**	**Running**	**Red**	**0.94**
2	SpeedFlex	Running	Blue	0.87
3	SportWalk	Walking	White	0.72

Filtered by: City = 'Toronto', Size = 12

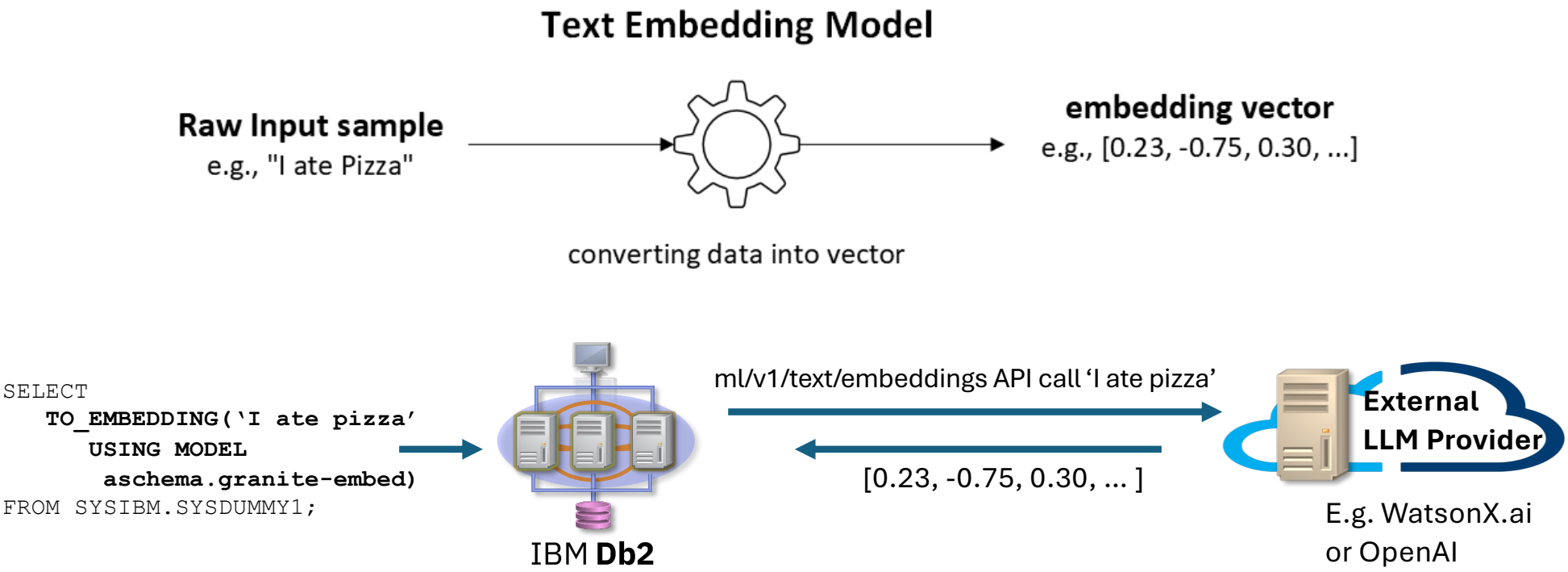
RAG with Db2


Data Pipeline
Select Source
(e.g. blog post URL)
Download &
Extract Content
(HTML Parsing)
Clean &
Preprocess Text
Generate
Embeddings
(using local model)
Store Vectors in
Db2 Vector Store
(with metadata)
Q&A Pipeline
User Query
Text Embedding
(using a local or cloud mdel)
Db2
Vector Search in
Db2 Vector Store
with Vector Index
Retrieve Top-k
Relevant Documents
LLM Contextualizes
& Generates Final
Answer
Return Final
Answer to User

LLM Integration #1
Vector Embedding: Turning Data Into Vectors



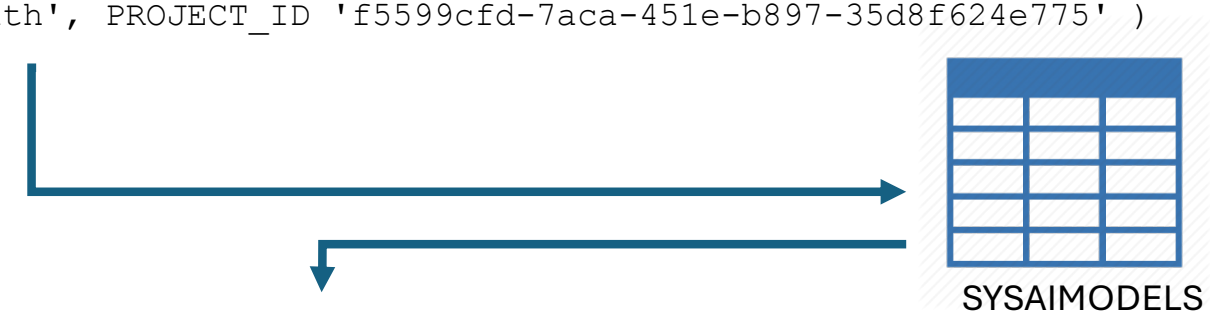
LLM Integration #1
Vector Embedding: Turning Data Into Vectors



LLM Integration #1

Registering an External Model Provider to use for Embedding Generation from SQL

```
CREATE EXTERNAL MODEL aschema.granite-embed
    PROVIDER WATSONX KEY 'api-keyxxxxx' ID 'ibm/slate-30m-english-rtrvr'
    TYPE TEXT_EMBEDDING
    RETURNING VECTOR(1024, FLOAT32)
    URL 'https://us-south.ml.cloud.ibm.com/ml/v1/text/embeddings'
    OPTIONS ( REGION 'us-south', PROJECT_ID 'f5599cfd-7aca-451e-b897-35d8f624e775' )
```




```
SELECT TO_EMBEDDING('I ate pizza' USING MODEL aschema.granite-embed )
FROM SYSIBM.SYSDUMMY1;
```

[0.23, -0.75, 0.30, ...]

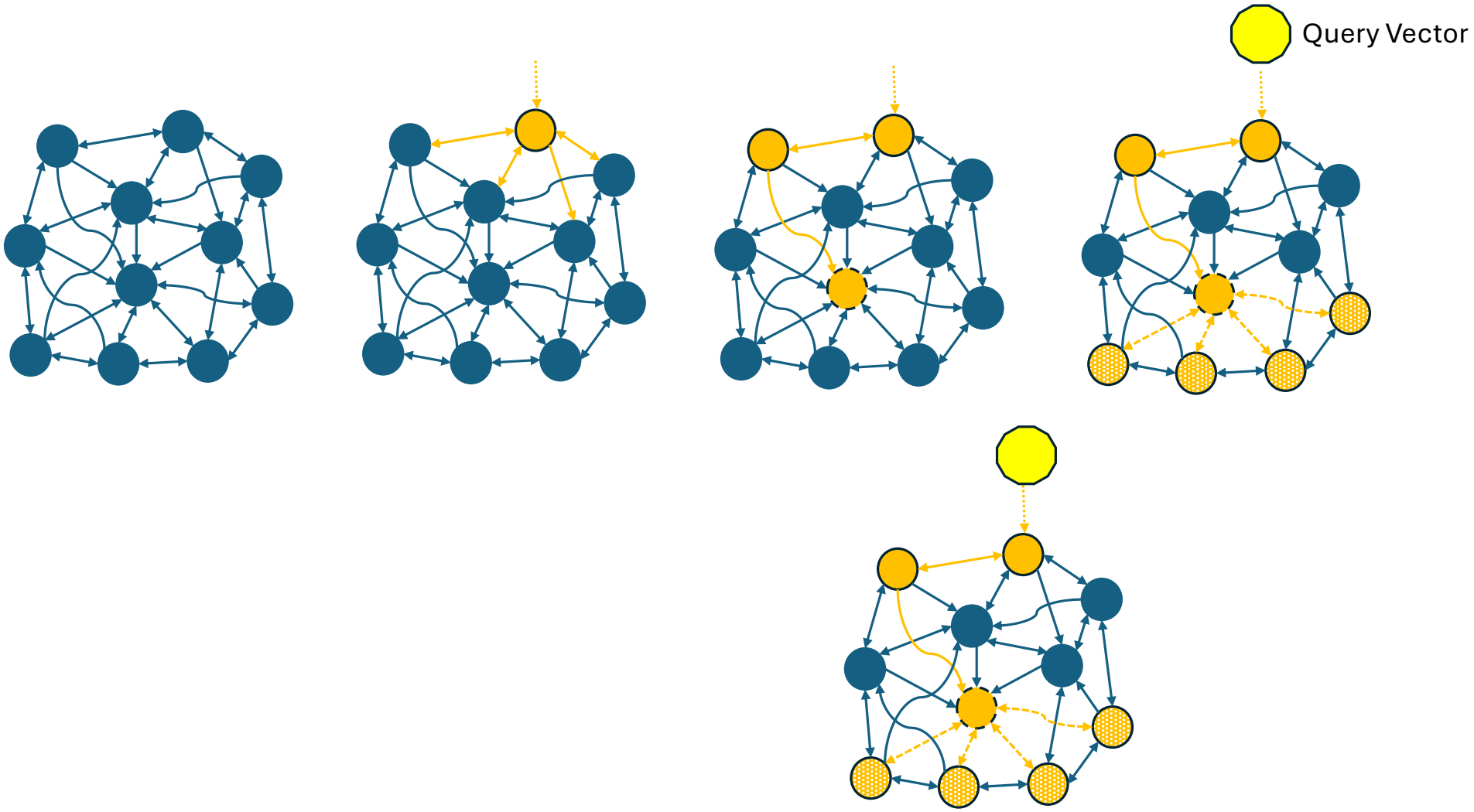
LLM Integration #2

Registering an External Model Provider to use for Text Generation from SQL

```
CREATE EXTERNAL MODEL myschema.granite-instruct
    PROVIDER WATSONX KEY 'api-keyxxxxx' ID 'ibm/ granite-13b-instruct-v2'
    TYPE TEXT_GENERATION
    RETURNING VARCHAR(1024)
    URL 'https://us-south.ml.cloud.ibm.com/ml/v1/text/embeddings'
    OPTIONS ( REGION 'us-south', PROJECT_ID 'f5599cfd-7aca-451e-b897-35d8f624e775' )
```


```
SELECT TEXT_GENERATION('How far is USA from India?' USING MODEL myschema.granite-instruct)
FROM SYSIBM.SYSDUMMY1;
```

"The distance between the USA and India is approximately **8,000 to 10,000 miles (12,860 to 16,090 kilometers)."


Query Vector