// FIELD GUIDE · 2026 EDITION

The LDM Field Guide

A Practical Reference for Large Database Models

Roughly **99% of enterprise data never reaches an LLM**. Large Database Models (LDMs) run AI directly inside your SQL database — learning semantic relationships from your own tables, with zero data movement and zero new compliance surface. This guide compresses the full operational playbook into six pages you can act on today.

INSIDE THIS GUIDE

PUBLISHED BY

Vertex Frontier Editorial

Frontier Intelligence for Enterprise Data

VF / LDM / 2026.08

v1.0 · 7 pages

Companion to the LDM longread

01 THE PROBLEM
Your AI Sees 1% of Your Data

Share of enterprise data reaching a large language model

Source: IBM Research estimate

Every enterprise AI roadmap eventually hits the same wall. The pilot succeeds on documents, wikis, and support tickets. Then someone asks the model a question about **actual customers, actual claims, actual transactions** — and it has nothing to say. Not because the model is weak. Because the data never left the vault.

That vault exists on purpose. Financial and insurance records sit behind encryption, row-level access control, retention policies, and audit trails. Extracting a copy isn't a technical inconvenience — it's a governance event. By IBM's estimate, between 32% and 40% of enterprise IT budgets are spent simply moving data around. The moment data leaves its regulated environment, it becomes harder to secure and harder to track. You didn't just spend money — you created a new copy to defend.

What Is a Large Database Model?

A Large Database Model (LDM) is a self-supervised neural network trained directly on the rows and columns of a relational table or view. It converts every distinct data value into a vector embedding and exposes those embeddings through standard SQL functions — letting you query by semantic similarity instead of exact-match WHERE clauses.

Three properties define the category:

Property	What it means	Why it matters
Narrow scope	You point the model at one table or view and select the columns. That selection is the feature engineering.	No "train on everything" fantasy. You commit to a domain.
Self-supervised training	Nobody labels anything. The model learns from which values tend to appear alongside which other values.	No data-scientist bottleneck. Co-occurrence is the signal.
SQL consumption	Output is a scalar function (AI_SIMILARITY, AI_ANALOGY, AI_SEMANTIC_CLUSTER) dropped into a SELECT.	Anyone who can write SQL can use it. No new tool to learn.

How it differs from the usual suspects

	LDM	LLM	Vector DB	Text-to-SQL
Trained on	Your selected tables / views	Public text corpora	External model embeddings	Public text + your schema
Best at	Similarity across records	Language & reasoning	Fast retrieval of pre-made embeddings	Translating questions to queries
Data movement	None — runs inside the DB engine	Prompt leaves perimeter	Requires embedding pipeline	Schema + results may leave
Hallucination risk	Very low — returns ranked real rows	High without grounding	None (stores only)	Moderate — wrong joins
Output	Ranked records + similarity scores	Generated text	Nearest vectors	SQL query

> **SECTION TAKEAWAY**
> An LDM is a small, focused embedding model for one table. Its narrowness is the whole point — and the four technologies above are complements, not competitors. The most interesting architecture uses an LLM for language, an LDM for structured similarity, and returns real rows the user can audit.

02 THE MECHANISM

The 5-Step LDM Pipeline

End-to-end, training an LDM is five steps. Step 2 is where it gets genuinely clever — and where most explanations go wrong. Understanding the binning trick is what separates teams that get useful similarity scores from teams that get noise.

STEP 1 · SELECT

1 **Pick a table and classify every column**

Label each column as Categorical (state, status, channel), Numeric (age, price, balance), or Key (customer ID, transaction ID). Getting this wrong produces a model that trains successfully and answers badly.

STEP 2 · TOKENIZE

2 **Turn every value into a token — with binning for numbers**

Numbers get binned first: 37 and 38 land in the same bucket and receive the same token ID. Every token gets tagged with its column name, so city:New_York is distinct from birthplace:New_York.

STEP 3 · REPHRASE

3 **Each row becomes an unordered sentence**

Every row is rewritten as a bag of words. Column order carries no meaning, so sequence models would waste capacity learning a pattern that isn't there.

STEP 4 · TRAIN

4 **Self-supervised network learns one vector per token**

The training signal is co-occurrence itself. Values that show up in similar rows end up near each other in vector space — no labels required.

STEP 5 · SERVE

5 **Vectors exposed as SQL functions**

Similarity, clustering, analogy — all scored in place. The model comes to the data, not the other way around.

The binning trick — and why it decides everything

You'd assume 37 and 38 behave like "cat" and "kitten" — close numerically, therefore close in vector space. They don't. To the model, 37 and 38 are just two arbitrary tokens with no built-in concept of numeric magnitude. Structurally, 37 is no closer to 38 than it is to "kitten". Binning fixes this: it groups numerically close values into shared buckets before training begins. You are **not asking the model to discover** that nearby numbers are related — you are **telling it**.

$$\text{bucket\_id} = \text{floor}\left((\text{value} - \text{min}) \div \text{width} \right) + 1$$

$$\text{width} = (\text{max} - \text{min}) \div \text{bins}$$

BINNING RULES OF THUMB

Too few buckets: you erase real distinctions — a 22-year-old and a 58-year-old share a token. Too many: you recreate the rare-value problem you were trying to solve. Aim for at least ~1,000 rows per bucket. Density beats granularity, every time.

TABLE READINESS

03 The C.O.R.E. Test — 15-Minute Filter

Most tables are bad LDM candidates. You can tell in about fifteen minutes — before you request a training job. Run the four checks below. If a table fails two or more, no amount of training configuration will rescue it. Fix the table, or pick a different one.

	What to check	Pass	Fail
C — Cardinality balance	Do categorical columns have a sane number of distinct values?	Roughly 10 to a few thousand distinct values per column.	4 values (no signal) or 800,000 near-unique values (email, free text, raw timestamps).
O — Overlap density	Do values genuinely recur across rows?	Many rows share the same triple of values across three columns.	Almost every row is a unique combination of unique values — no co-occurrence to learn from.
R — Row volume per token	After binning, does every bucket hold enough rows?	Every bucket and category has ≥ 1,000 rows for a stable vector.	Sparse tokens — the single most common cause of "the similarity results look random."
E — Encoded semantics	Do your columns actually mean what they say?	Each column stores one cleanly-scoped attribute.	Legacy overloaded fields — a status column that also stores a region prefix, or a notes field carrying structured data.

Binning calculator — worked example

You can sanity-check a numeric column before you train on it. Plug in the column minimum, maximum, bucket count, total row count, and a sample value to locate. The example below uses a customer age column from 18 to 95 with 16 buckets and 2 million rows:

Input	Value	Output	Value
Column minimum	18	**Bucket width**	4.81 years
Column maximum	95	**Bucket count**	16
Number of buckets	16	**Average rows/bucket**	125,000
Sample value to locate	37	**Falls in bucket**	B5 (range 35.24 – 40.05)
Total rows in table	2,000,000	**Verdict**	Density looks reasonable ✓

SECTION TAKEAWAY

LDM success is decided at table selection, not at training time. The C.O.R.E. test is a fifteen-minute filter that saves weeks. If your table fails two or more checks, walk away — and consider building a deliberately wide, denormalized view instead of training on a pure 3NF table.

QUERYING THE MODEL

04 Five Query Types & Decision Matrix

Once the vectors are live inside the database, five families of questions open up. Notice how much of classical analytics collapses into these five — segmentation, outlier detection, recommendation, entity resolution, and portfolio comparison have historically each required their own model. Here they are five ways of reading the same vector space.

Query type	Question it answers	SQL pattern (IBM Db2)
Similarity	Which customers most resemble this one?	AI_SIMILARITY(customer_id, 'CUST4729')
Dissimilarity	Which transactions look least like the normal pattern?	AI_SIMILARITY(...) ORDER BY score ASC
Semantic cluster	Does this product belong with a reference group?	AI_SEMANTIC_CLUSTER(prod, ref_set)
Analogy	Is relationship A→B like relationship C→D?	AI_ANALOGY(a, b, c, d)
Commonality	Which records show unusually typical or rare combinations?	Aggregate similarity score against the corpus

A worked SQL example

Notice what's missing: nobody picked the fields. The model already learned, across every selected column and every row, which values tend to travel together. And a plain WHERE clause still works — semantic scoring and conventional filtering coexist in one statement.

SQL — the LDM approach

```
SELECT  customer_id,
        AI_SIMILARITY(customer_id, 'CUST4729') AS score
FROM    customers
WHERE   country = 'US'
ORDER   BY score DESC
FETCH   FIRST 100 ROWS ONLY;
```

When to use what — decision matrix

If your problem is…	Use this	Why
Find rows similar to a known good row	LDM	Native strength — similarity is the entire point of the model.
Generate a paragraph explaining a result	LLM	LDMs produce no language. They hand you ranked rows, not narratives.
Store pre-computed embeddings for fast retrieval	Vector DB	Vector DBs store embeddings; they do not train them. Use when your embeddings come from elsewhere.
Translate a natural-language question to SQL	Text-to-SQL	Useful for ad-hoc analyst queries, but fragile on multi-table JOINs and blind to data distributions.
Enforce a hard business rule (e.g. policy implies coverage)	Constraints / traditional SQL	LDMs do not reliably capture functional dependencies — peer-reviewed research confirms this. Use them for discovery, not enforcement.
Detect fraud / anomalies without a labeled training set	LDM (dissimilarity)	Anomaly detection is similarity search with the sort order reversed. If you have similarity, you have fraud detection.

SECTION TAKEAWAY

One trained model, five query types. That consolidation is where the real operational savings come from — not from any single query being faster, but from collapsing segmentation, outlier detection, recommendation, and entity resolution into one vector space.

MARKET REALITY

05 Vendor Landscape & Common Mistakes

IBM currently leads the commercial LDM category — it shipped the first LDM-based product in 2022 as SQL Data Insights in Db2 for z/OS, with SQL Data Insights Pro reaching general availability in March 2026. Other database vendors are approaching the same problem from different directions: most provide in-database vector search using externally generated embeddings, rather than training a dedicated model from relational tables.

Vendor / Product	Capability	True LDM?	Note
IBM — SQL Data Insights Pro	Trains embeddings directly from relational tables; AI_SIMILARITY / AI_ANALOGY / AI_SEMANTIC_CLUSTER; added unstructured text support (March 2026 GA).	□ Yes	Only commercial vendor marketing the LDM category as a distinct product.
Oracle — AI Vector Search	Semantic similarity search inside Oracle Database, alongside traditional SQL.	□ No	Relies on externally generated embeddings rather than training from tables.
Microsoft — SQL Server 2025	Native vector search and AI features; combine structured data with embeddings.	□ No	Focus is on retrieval and semantic search, not table-specific database models.
PostgreSQL + pgvector	Widely-adopted open-source vector similarity search extension.	□ No	Stores and queries embeddings efficiently but does not generate them.
Snowflake — Cortex AI	Native vector capabilities inside the data warehouse.	□ No	Integrates LLMs with enterprise data, not dedicated database models.
Databricks	Delta Lake + vector search + AI tooling for agentic workflows.	□ No	Broader AI infrastructure rather than a dedicated LDM architecture.

Six mistakes that quietly wreck LDM projects

#	Mistake	Fix
1	Including the primary key as a trainable column. A unique ID appears once — its vector is meaningless and inflates vocabulary size.	Mark keys as keys, never as categorical features.
2	Treating a numeric column as categorical because it "looks like a code". Binning zip codes creates buckets like "10001–14999" — geographic nonsense.	Classify by meaning, not by data type.
3	Dumping free-text columns into a structured model. Millions of unique strings = millions of single-occurrence tokens.	Exclude free text unless your platform explicitly supports it (SQL Data Insights Pro does).
4	Choosing bucket counts by round numbers. "Ten buckets because ten is tidy."	Choose by row density per bucket — at least ~1,000 rows each.
5	Training once and forgetting. Customer behavior drifts; last year's model confidently returns last year's neighbors.	Schedule incremental refreshes. Modern LDM engines support delta updates without full retraining.
6	Reading similarity as causation. The model says two rows occupy the same region of vector space — not that one caused the other.	Treat LDM output as a hypothesis generator feeding an experiment, not a conclusion.

SECTION TAKEAWAY

Almost every LDM failure traces back to column classification or token sparsity — not to the model itself. Spend your debugging time on table selection and column classification, not on training hyperparameters.

06 ACTION PLAN

30-Day Rollout + Self-Audit + Glossary

Your first 30 days — a practical rollout plan

Days	Action	Why it matters
1–3	Pick one question, not one table. "Which accounts resemble the ones that churned last quarter?" beats "let's try AI on customer data."	A specific question gives you a measurable baseline to beat.
4–7	Run the C.O.R.E. test on candidate tables. Reject anything failing two or more checks.	Saves weeks of wasted training on bad tables.
8–10	Build a wide training view. Denormalize deliberately. Exclude keys and free text. Classify every column by meaning.	Clean schemas hurt LDMs; wide views with redundant context help.
11–14	Write the best rigid SQL segment you can. Record its outcome metric.	You will need this number to compare against the LDM segment.
15–20	Train two or three column-set variants. Score the same reference entities against each.	Column selection is your only real hyperparameter.
21–25	Blind-review the results with a domain expert. Show ranked neighbors with no explanation.	Experts spot broken embeddings faster than any metric.
26–30	Run the head-to-head. Same campaign, LDM segment vs baseline. Report the delta honestly.	Negative deltas are signal — they tell you the table or column set was wrong.

Self-audit: is your table LDM-ready?

☐ 01	My categorical columns each have between ~10 and a few thousand distinct values.
☐ 02	My numeric columns can be binned so every bucket contains at least ~1,000 rows.
☐ 03	I have a Key column I can mark as non-trainable (customer ID, transaction ID).
☐ 04	I have excluded (or separately handled) free-text columns like notes and comments.
☐ 05	When I pick three columns and count rows sharing the same triple, the count is non-trivial.
☐ 06	I have a specific business question — not "let's try AI on this table."
☐ 07	I have a baseline segment or metric to compare against before training.
☐ 08	I have a domain expert available for a blind review of similarity results in week four.

If you checked fewer than 6 of these — fix the gaps before requesting a training job. The model is not the bottleneck; the table is.

Glossary — nine terms you'll need on this journey

Embedding	A vector representing a value's meaning. Similar values land near each other.
Binning	Grouping numerically close values into shared buckets before training — the trick that makes numeric columns learnable.
Co-occurrence	The training signal for LDMs. Values that show up together in rows are pulled together in vector space.
HNSW	Hierarchical Navigable Small World — a graph index that lets the optimizer evaluate similarity across millions of rows without a full table scan.
RBAC	Role-Based Access Control — the existing DB permission model that LDMs inherit for free, since semantic queries are still SQL.
Semantic cluster	A query type that scores a candidate against a set of reference examples — useful for "does this product belong with this group?"
Analogy	Tests whether relationship A→B resembles C→D — the classic vector-arithmetic pattern applied to business entities.
Functional dependency	A hard rule "X always implies Y". LDMs do not reliably capture these — enforce them with constraints.
Vector drift	Silent degradation of an embedding space over time. Queries return plausible results, but known-good neighbors stop ranking near each other.

Read the full article at Vertex Frontier. This field guide is the operational companion to our LDM longread — covering documented deployments, the contrarian case for denormalized views, and what embeddings miss.