# Data Warehouse — Article

## Definition

A data warehouse is a centralized store of structured, historical data — typically modeled with facts and dimensions using a star or snowflake schema — organized specifically to support analytical querying and reporting, as opposed to the transactional operations a production database handles. It's not just "a database with reporting tables"; the storage engine itself and the data model are shaped around analytical access patterns.

## Real-life usage

The clearest way to see this is through fact and dimension tables. A retail warehouse might have `fact_sales` — one row per line item, carrying measures like quantity and revenue — joined to `dim_product`, `dim_store`, `dim_date`, and `dim_customer`. Anyone in finance can then slice revenue by region, by month, by category, in combinations nobody had to pre-build.

The same pattern shows up everywhere analytics matters at scale:

- **Banking/fintech**: regulatory reporting (daily risk exposure, AML) depends on consistent historical snapshots — "what was our exposure on March 31st" — not just current state, which is exactly what a warehouse's historical fact tables are built to answer.
- **Airlines**: `fact_bookings` / `fact_flights` joined to `dim_route`, `dim_aircraft`, `dim_time` let analysts study load factors, delays, and revenue per route across years of history.
- **Marketing**: `fact_impressions` / `fact_conversions` joined to `dim_campaign`, `dim_channel`, `dim_audience` answer "which channel actually drives conversions" across time — something no single operational system tracks end to end.

A useful analogy: a physical warehouse doesn't manufacture goods — it receives finished goods from many sources and organizes them onto labeled shelves by category (dimensions), with clear counts (facts), so anyone can walk in and pull exactly what they need without digging through the factory floor. A data warehouse does the same thing with data instead of goods.

# 5 Whys — Why Data Warehouses Exist as a Distinct Thing

The natural challenge is: you could build a star schema in any relational database — Postgres, MySQL — so why does the industry build dedicated warehouse engines (Snowflake, BigQuery, Redshift) at all?

**Root cause 1: columnar storage matches the analytical access pattern.** In a normal row-store database, a table's rows are stored together on disk — all of row 1's columns, then all of row 2's columns, and so on. A typical warehouse query like `SELECT SUM(revenue) FROM fact_sales WHERE region = 'EU'` only needs 2 of maybe 30 columns, but a row-store still has to pull every column of every matching row off disk into memory before it can discard the ones it doesn't need. That's wasted I/O and wasted cache space, proportional to the columns you don't care about.

Warehouse engines fix this with columnar storage: each column is stored contiguously on disk, so a query reads only the columns it actually references. This also unlocks two side benefits — far better compression (similar values stored together compress much more efficiently than mixed-type rows) and vectorized execution (operating on whole columns of values at once instead of row by row). This is the physical, mechanical reason a warehouse engine behaves differently from a row-store RDBMS running the exact same schema.

**Root cause 2: fact/dimension separation avoids redundancy and update anomalies.** Given columnar storage already solves read efficiency, it's fair to ask why not just put everything — customer name, product category, store region, revenue, quantity — into one big flat, columnar table. The answer is what happens to descriptive attributes under that design. If "customer address" lives inline in every sales row, and that customer has 500 transactions, the same address string is stored 500 times. At billions of rows, that's significant redundancy even under compression, and it creates an update anomaly: if the customer moves, do you rewrite all 500 (or 5 million) rows referencing them? Miss one, and the data is now silently inconsistent.

The deeper issue is conceptual: a "sale" and a "customer's current address" are different kinds of data that change at fundamentally different rates. Facts are an append-only event log; dimensions are comparatively static reference data that occasionally changes (which is exactly why "slowly changing dimensions" exists as a defined technique — dimensions do change, just rarely, and need an explicit strategy for handling it). Separating facts from dimensions keeps each kind of data managed according to its actual rate of change, instead of conflating an immutable event stream with mutable reference data in one structure.

Put together: **data warehouses exist because (1) columnar storage matches the "few columns, many rows" read pattern of analytical queries in a way row-stores structurally cannot, and (2) fact/dimension modeling avoids the redundancy and update anomalies that come from mixing high-volume immutable events with low-volume mutable reference data in a single flat structure.**

This root cause generalizes beyond SQL warehouses. A doc-oriented store like Elasticsearch actually does provide columnar-style reads for aggregations, via a structure called `doc_values` — a separate, purpose-built columnar layer distinct from its inverted index for text search. So root cause 1 can genuinely be satisfied outside a traditional warehouse engine. But root cause 2 is a modeling discipline, not a storage-engine feature — it has to be deliberately designed in. A medallion-style `agg_` index built from flat JSON documents that embed both facts and dimension attributes in every document reproduces the exact flat-table problem: dimension values are duplicated across every document that references them, and because a document store has no native join to factor dimensions out, a dimension change either goes unpropagated (silent inconsistency) or requires reindexing every historical document that referenced it.

# Value vs. Risk

**Value.** Understanding these two root causes changes how you evaluate any analytical data store, not just SQL warehouses. It tells you precisely what to check before trusting a design: does the storage layer read only the columns/fields a query needs (columnar or equivalent), and are high-volume facts kept separate from low-volume, referenced dimension data — or are they flattened together for convenience? For a system like a medallion-on-Elasticsearch setup, this means factoring dimension keys/IDs into fact-like documents and maintaining separate dimension indices, resolving them at query or ingest time, rather than embedding full dimension attributes into every aggregate document.

**Risk.** Keeping facts and dimensions flattened together, whether in a wide SQL table or in JSON documents, defers a cost rather than avoiding it. As data volume grows and dimension values change more often, the concrete failure looks like this: a report segmenting historical sales "by current product category" silently returns wrong numbers, because only some historical records were ever updated to reflect the new category — and there's no single source of truth to detect or correct the drift from, since nothing enforces referential consistency across the duplicated copies. This isn't a performance problem; it's a correctness problem that erodes trust in the numbers exactly the way inconsistent metric definitions do at the platform level.

---

Revision #2
Created 31 August 2026 09:01:40 by N8N
Updated 28 September 2026 14:39:35 by EMB