ACTIVANT RESEARCH

Conversational Analytics

The Illusion of Chatting with your Data

March 2026

AUTHOR:

Scott Watson
Vice President, Research

About Activant

Activant is a research-led global investment firm that partners with high-growth companies. Since 2015, we have invested in category-defining businesses during their most critical phases of growth, partnering with founders who have won their initial battles and are ready for the next challenge.

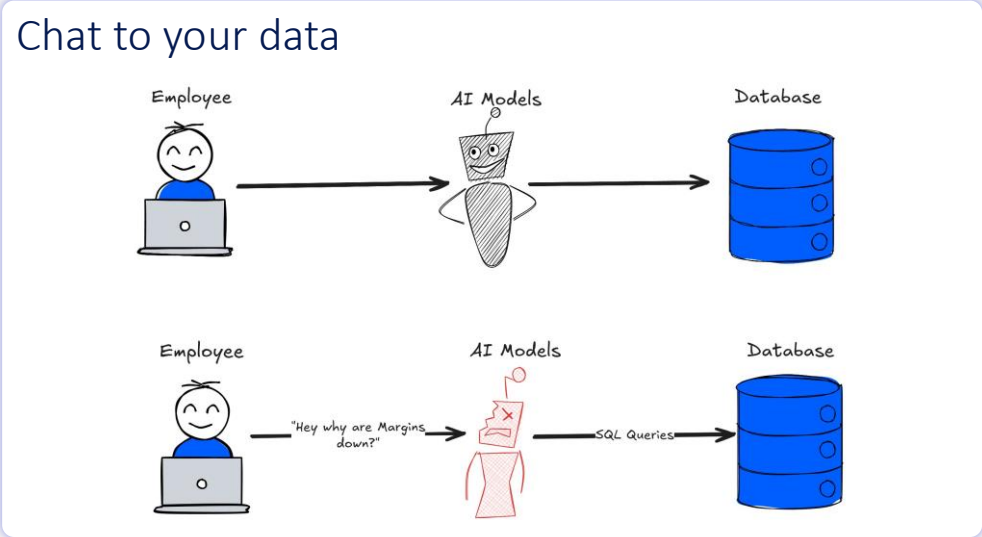
Our approach pairs deep, proprietary research with patient, flexible capital and hands-on operational partnership. We work alongside founders and leadership teams to refine strategy, strengthen operations, and accelerate sustainable growth.

Activant Research is dedicated to uncovering the most exciting emerging technologies, sectors, and companies we believe will shape the future. Our research-driven perspective informs everything we do, helping us invest at meaningful inflection points and support founders in building enduring, category-leading businesses.

You can find out more about Activant and our research at https://activantcapital.com/.

Text-to-SQL: The Promise and Reality

The enterprise data market has been chasing one goal for decades: conversational analytics. The objective is to allow any user, from the CEO to the junior analyst, to type a natural language question and instantly receive an accurate data visualization. Text-to-SQL is the technological engine built to make this happen.



In production environments, however, this bridge often fails. While LLMs like GPT and Claude can easily generate SQL syntax (the grammar of code), they routinely struggle with semantic resolution (the logic of the business).

The execution drop-off is stark. On isolated academic benchmarks (e.g., Spider 1.0 and 2.0), LLMs achieve approximately 86%–94% execution accuracy.[1] [2] However, when applied to realistic, complex enterprise schemas (e.g., the BIRD benchmark), baseline performance falls to roughly 54%–63%.[3]

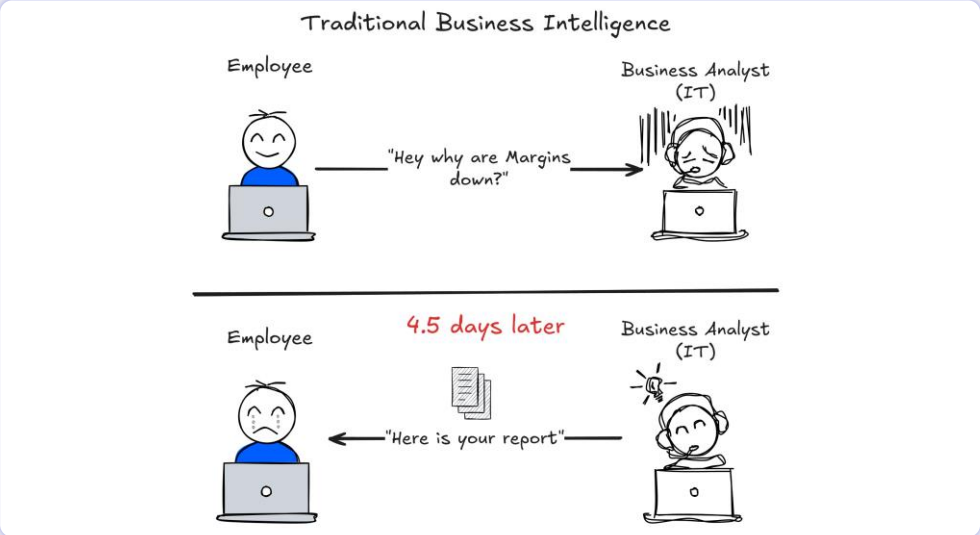
These models do not struggle to write code; they struggle to parse enterprise context. They are blind to the tribal knowledge buried in Slack threads, scattered across Google Docs, and locked inside the heads of internal data teams. The result is output that business users simply don't trust.

This trust gap explains why legacy dashboards like Tableau and Power BI still dominate the market. Cracking Text-to-SQL doesn't mean waiting for a smarter foundation model. It means changing the data architecture.

To understand why natural language hasn't effortlessly unlocked our databases, we first need to deconstruct why Text-to-SQL is so inherently difficult, and what human engineers do instinctively that AI still cannot.

Traditional Business Intelligence

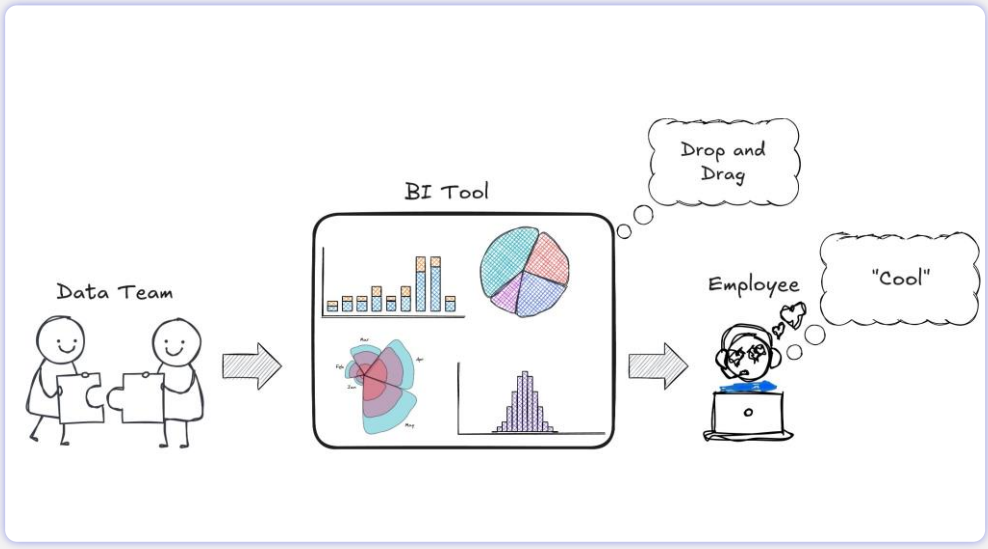
Before modern visualization platforms, the enterprise data supply chain was a rigid, manual slog. Data retrieval required a human gatekeeper, dragging business users into a disjointed feedback loop where the pace of the market far outstripped the speed of the SQL query.



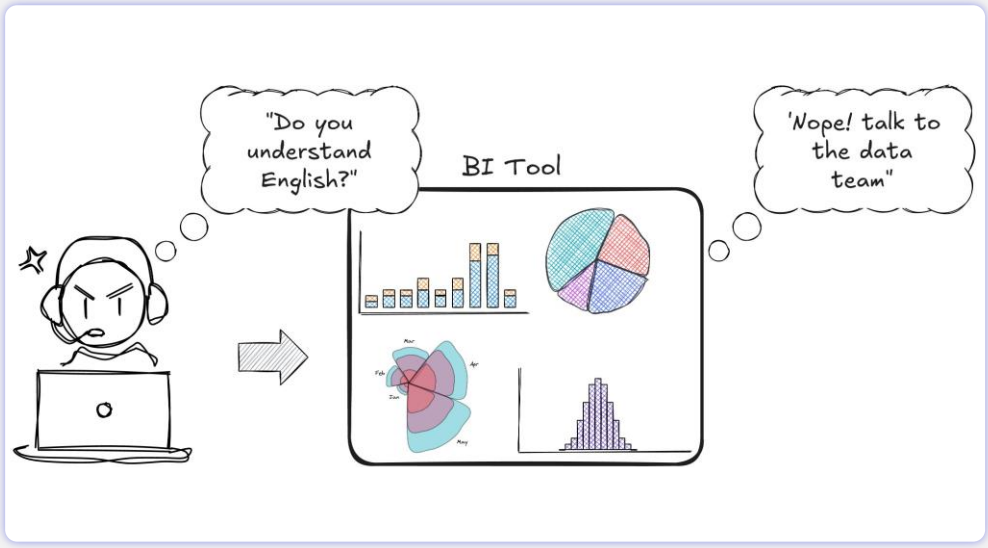
The mechanics were broken. An operational leader might submit a natural language query such as "analyze margin compression in the Northeast," only to watch that request disappear into an IT ticket queue. With average time-to-insight of 4.5 days (Day 0-1: triage, Day 1-2.5: context negotiation, Day 2.5-4.5: execution),[4] the decision window was often closed by the time the report landed. The data remained mathematically accurate, but economically obsolete.

To compress this lag, the industry pivoted to "Self-Service BI" via platforms such as Tableau and Power BI. The goal was autonomy: bypass the IT bottleneck by giving non-technical users direct database connections and a graphical interface.

The result? Despite reaching 87% enterprise adoption in North America, active usage stalled at approximately 26%.[5] These platforms provided the capability to connect to internal systems, but they weren't built for how non-technical people actually work. Instead, they simply shifted the technical burden onto the business user, turning expensive data initiatives into frustrating sunk costs.



Business users don't process information through dimensional models; they think in language. When you force a business operator to think like a database architect, the workflow breaks. They drag the wrong field, generate a broken chart, and retreat to the safety of legacy workflows, usually exporting raw data to Excel or submitting yet another ticket.

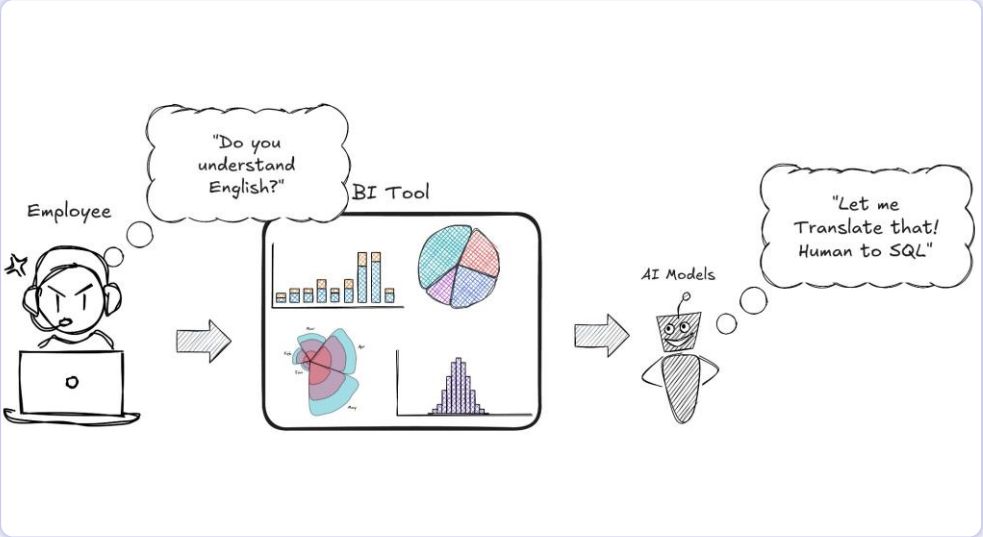


The market tolerated this high-friction status quo until LLMs normalized the natural language interface. Suddenly, conversational querying made the cumbersome drag-and-drop mechanics of legacy dashboards look archaic.

Facing obsolescence, legacy incumbents (Salesforce, Oracle, Microsoft, and IBM) reflexively bolted AI features onto their existing rendering engines. However, because these tools weren't architected for native language comprehension, the integration triggered new issues. Instead of solving the usability bottleneck, grafting AI onto legacy frameworks created the Text-to-SQL hallucination problem: users got their charts instantly, but they couldn't trust the math.

AI Illusion: Why AI Text-to-SQL Breaks in Production

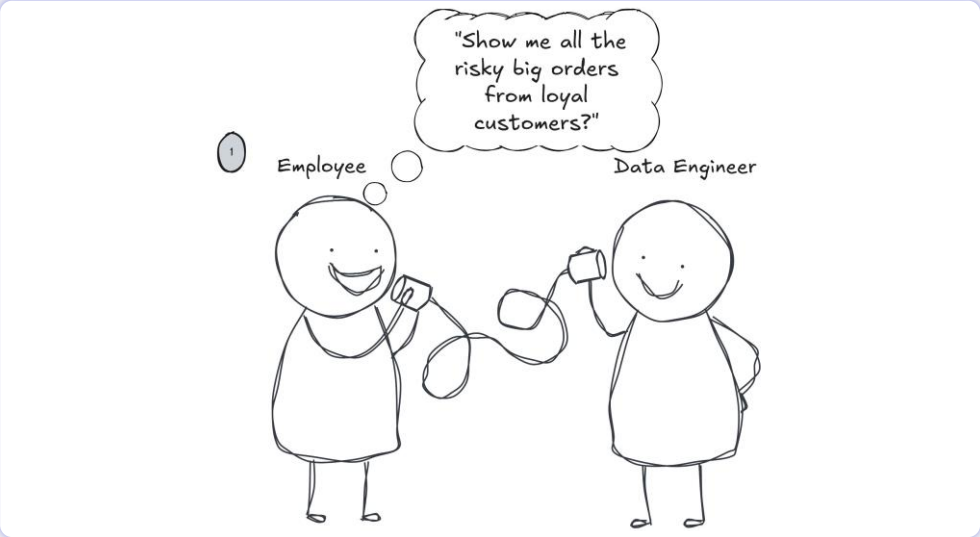
On paper, the AI analytics workflow looks flawless: an employee asks a question in plain English, the AI writes the SQL, the database executes the query, and a chart appears.



In reality, this workflow breaks at the translation layer. To understand why AI models fail to generate reliable SQL, we must first examine the cognitive workflow of the humans they are attempting to replace.

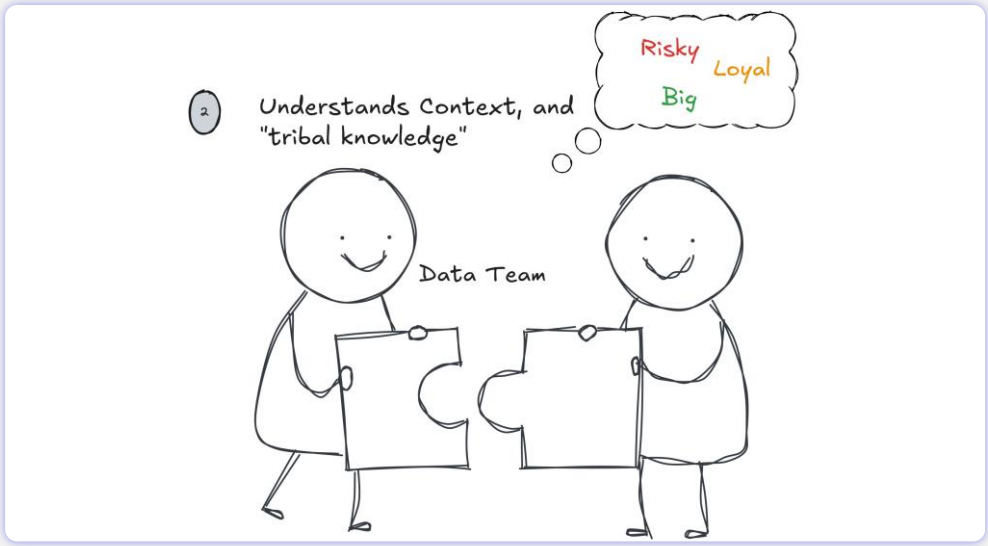
The Ambiguity Engine: Why Human Language Breaks Code

When an operational user submits a flexible natural language query such as *"Show me all the risky, big orders from loyal customers,"* a human data engineer does not immediately write code. Instead, they execute a multi-step semantic translation process:

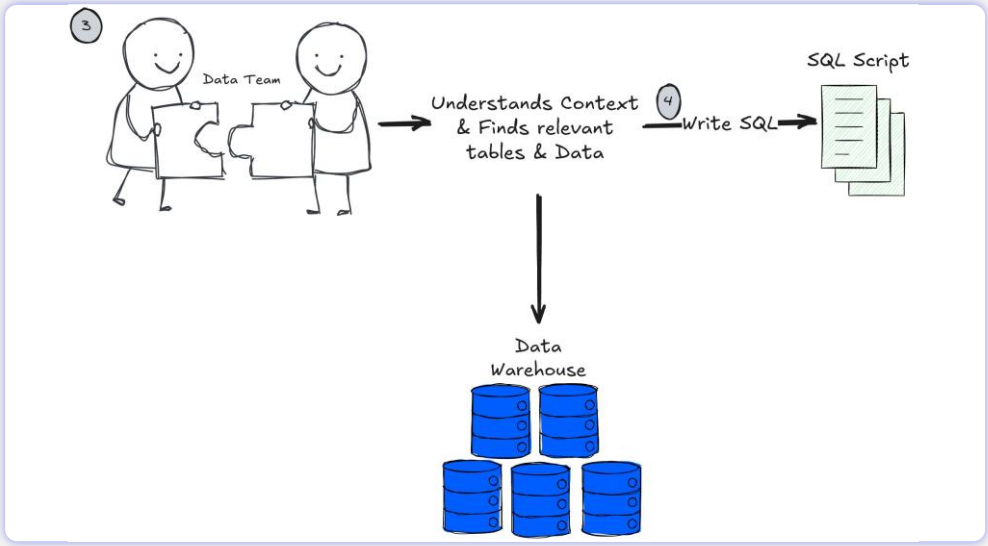


1. Humans naturally process business context, slang, and undocumented institutional rules. Before interacting with the database, the engineer defines the abstract terms:

    - What defines **"risky"**? (e.g., unverified shipping addresses).
    - What defines **"big"**? (e.g., order value exceeding $50,000).
    - What defines **"loyal"**? (e.g., active purchasing history >36 months).



2. Once the business logic is defined, the engineer maps these concepts to the highly complex, often tangled database schema (identifying the correct tables, columns, and join paths). Crucially, if the prompt is ambiguous, the human engineer initiates a clarification loop, returning to the employee to verify intent before proceeding.



Only after the semantic logic is entirely resolved and the schema mapped does the engineer write the SQL syntax.

Even with deep institutional context, internal communication channels, and the ability to ask clarifying questions, human data teams still struggle with the Text-to-SQL process.

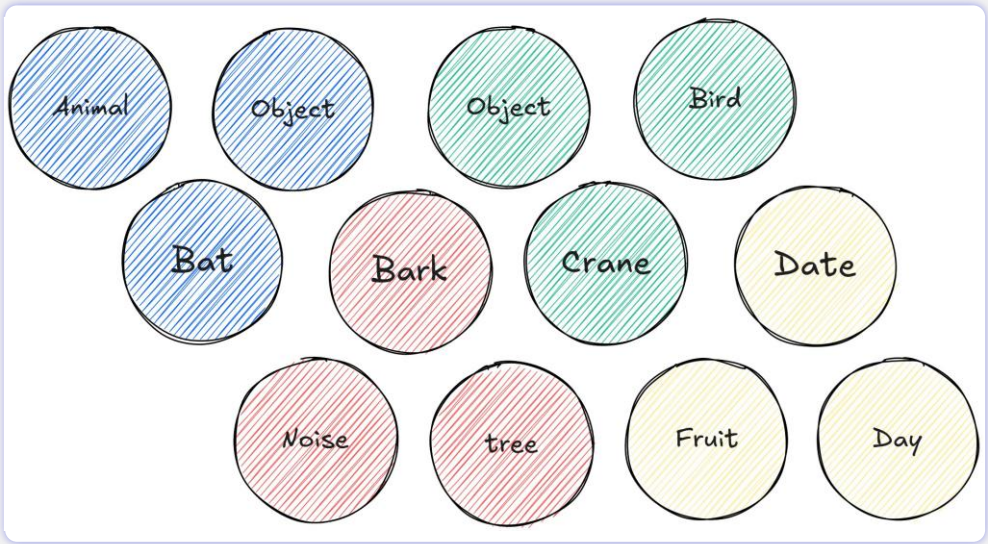
This difficulty highlights the core architectural problem. Translating flexible, highly ambiguous natural language into rigid, strictly structured SQL is inherently unstable. Expecting a zero-shot LLM, an entity completely isolated from a company's internal vernacular, lacking built-in feedback mechanisms, and blind to undocumented database complexity, to autonomously outperform human engineers is unrealistic.

Why Enterprise Language is Inherently Ambiguous

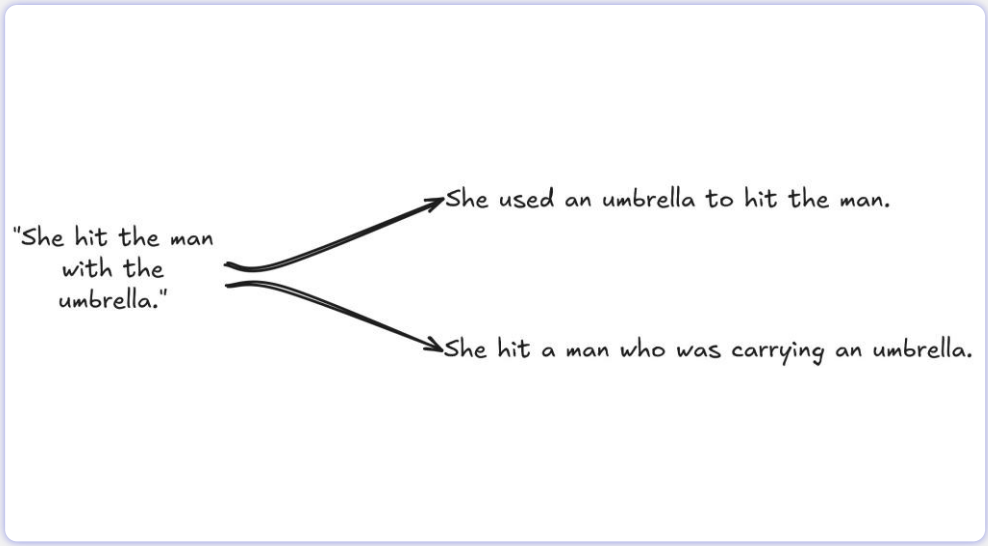
Human communication is messy. Unlike SQL, which follows strict rules, the way we ask questions constantly shifts. It is shaped by departmental assumptions and packed with ambiguity, often driven by a user's specific role and workflows as well as the unwritten rules of their department.

In enterprise operations, a single statement can generate multiple, conflicting interpretations. This friction usually stems from two main culprits:

Conflicting definitions: A single word often carries entirely different meanings depending on who is asking. In a business context, a word like "churn" might mean *voluntary account cancellation* to the Product team, but *credit card failure* to the Finance team.

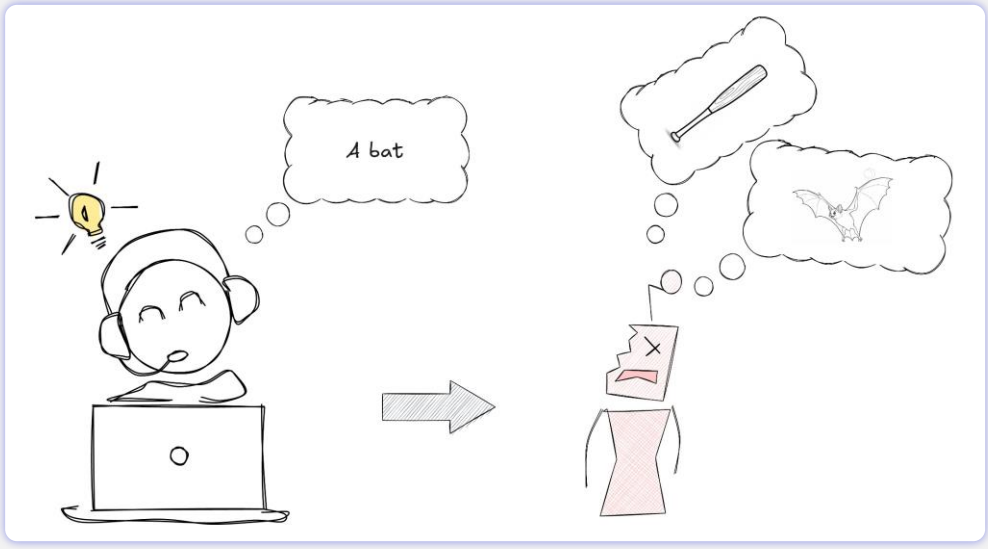


Tangled phrasing: Sentence structure can drastically change meaning. A seemingly simple prompt may contain overlapping clauses that make the underlying mathematical logic impossible to determine definitively.



This ambiguity is amplified in written enterprise communication such as Slack threads, emails, and ticketing systems. Text-based queries strip away vital context, resulting in the loss of tone, urgency, and shared understanding.

Human engineers fill these gaps through intuition and conversation. They rely on knowing the requester, understanding operational context, and engaging in iterative dialogue to verify the intent of the request before pulling data.



AI models lack these capabilities. They have no operational memory, no relationship with the requester, and no ability to pause and ask clarifying questions. So, what do they do? They guess.

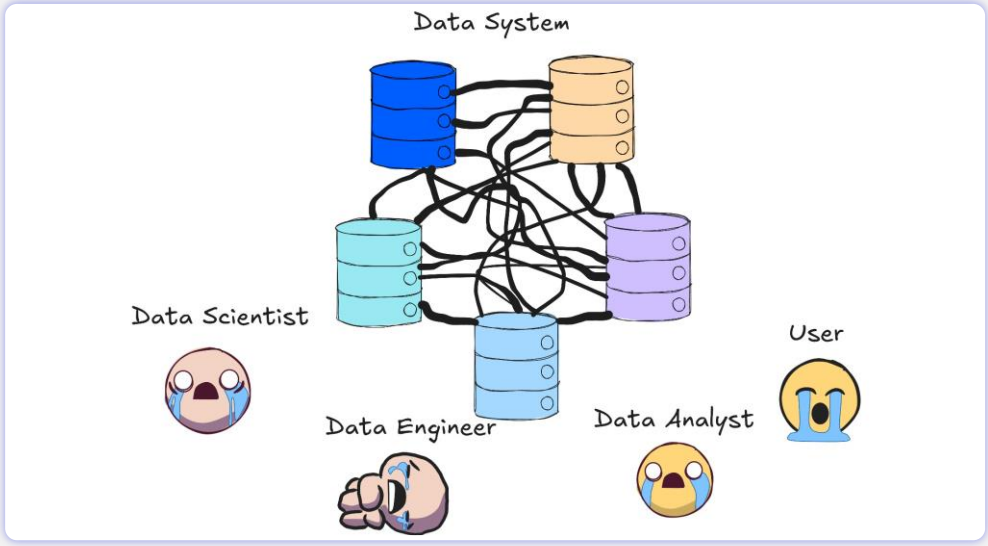
When an AI encounters linguistic ambiguity, it can't read between the lines. Instead, it executes a statistical guess based on its training weights. If that probability calculation is misaligned with the user's actual intent, the model hallucinates. It generates a syntactically flawless SQL query that returns the wrong data while presenting it as fact.

But linguistic ambiguity is only half the problem. The databases themselves are chaotic.

The Chaos of the Enterprise Data Warehouse

Enterprise data systems accumulate structural debt. Under the hood, this typically appears as:

- ambiguous or poorly named columns
- undocumented, complex relationships between table
- multiple, conflicting methodologies to calculate the same business metric

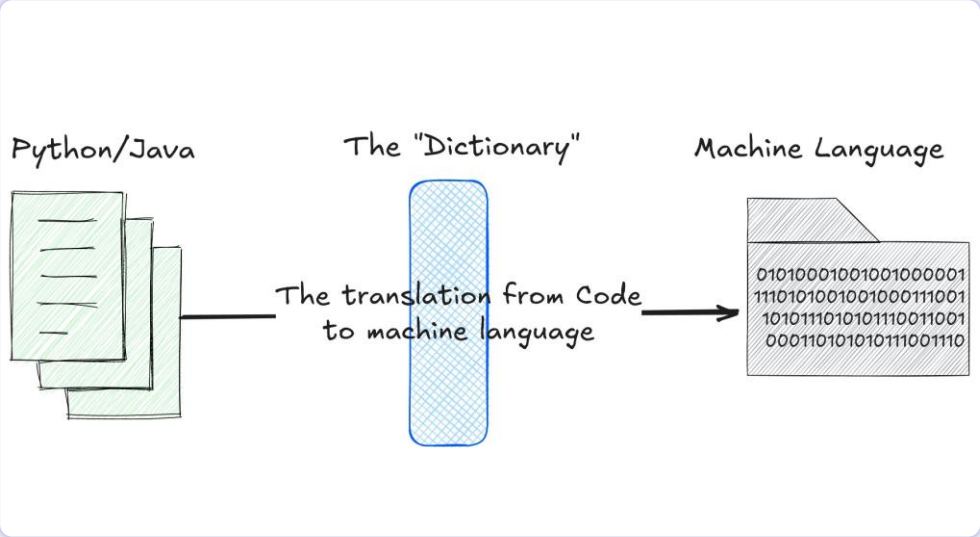


Figuring this out is never straightforward. Even human data engineers rarely write the correct SQL on the first attempt. Because enterprise schemas often lack robust modeling, engineers must chase down context across teams just to understand what the data actually means. They interrogate requesters, debug inaccurate reports, and rely on trial-and-error until the query actually returns the right numbers.

If human engineers with full institutional knowledge, direct access to colleagues, and years of experience still struggle to get the query right on the first try, expecting an isolated AI model to succeed immediately is naive.

The Translation Trap: English to SQL

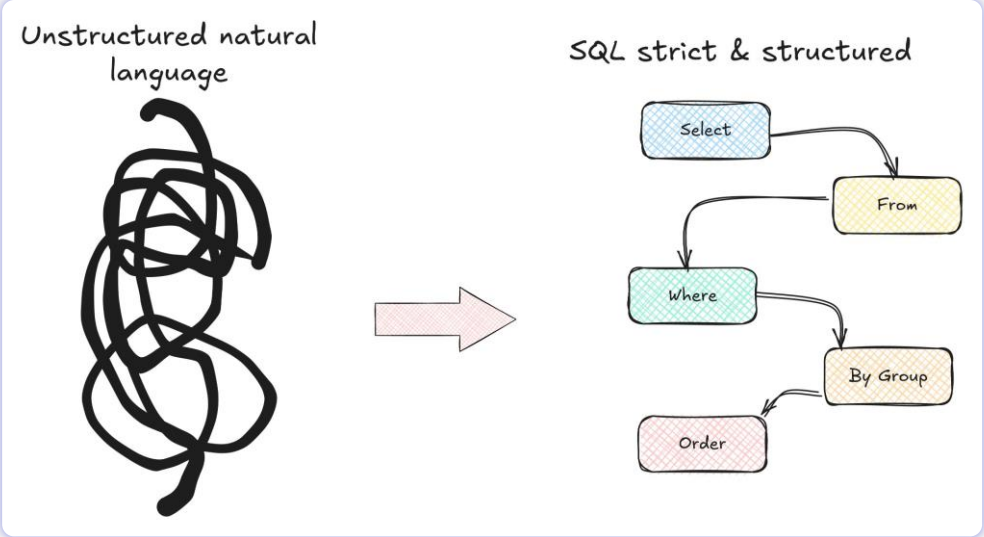
When a software engineer writes in Python or Java, the computer must translate that code into machine language. While technically complex, the rules are straightforward: it is a strict, *one-to-one mapping*. The programming language acts like a dictionary where every command has one exact, undisputed meaning.



Translating human language (English) into database language (SQL) breaks this rule entirely. It is a *one-to-many mapping*. A single English word in a prompt might connect to dozens of different tables, columns, or metrics depending on context.

This creates a mismatch between two different systems:

- **Human language is flexible:** We use slang, abbreviations, and ambiguous phrasing.
- **SQL is rigid:** It requires absolute precision and strict syntax. To complicate things, SQL isn't even universal, the rules change depending on which database you use (e.g., Snowflake vs. PostgreSQL).



In the real world, it isn't enough for SQL simply to *run*. Queries must also be highly optimized for speed, easy for a human to read, and fully reliable.

When we ask an AI model to take a structurally vague English question, figure out the business context, write strict code, and optimize it for a specific database all at once, the architecture collapses under the cognitive load. This results in queries that are painfully slow, mathematically flawed, or entirely impossible to debug. The AI needs a map.

The Solution

To fix the Text-to-SQL failure rate, we must stop forcing AI to guess the context. The architecture must introduce a translation layer between the language model and the raw database.

Recent academic benchmarking supports this shift. In a study evaluating LLM accuracy on a complex enterprise database, researchers tested ChatGPT using two distinct workflows:[6]

1. **Raw Text-to-SQL:** The AI was asked to generate SQL directly from the raw database tables.
2. **Context-Augmented SQL:** The AI was required to use a "Knowledge Graph," a pre-built map defining how the business entities relate to one another.

The results were decisive. When the LLM was grounded by the Knowledge Graph, execution accuracy jumped from 16% to over 54%[7] across all levels of

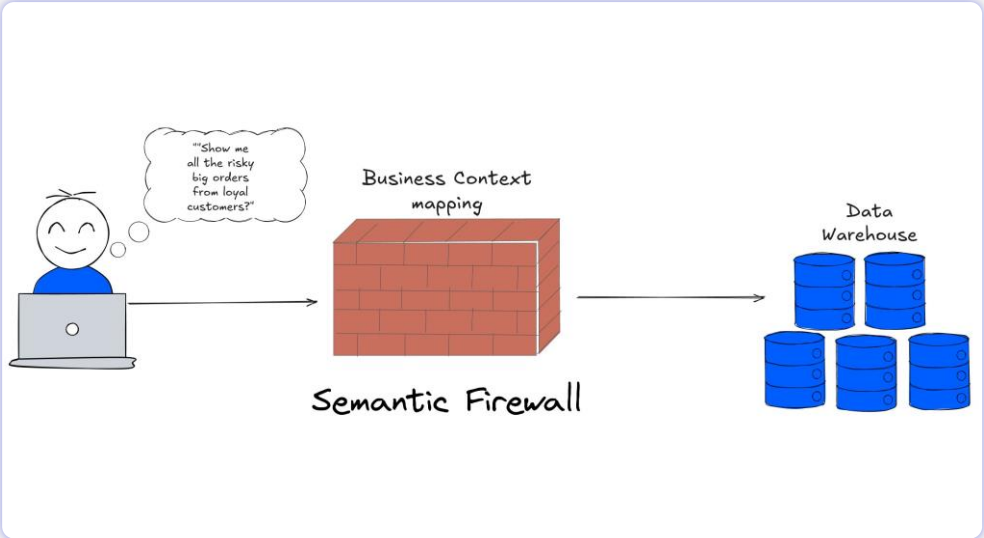
query complexity. By supplying the model with explicit business rules, the architecture removes AI's reliance on probabilistic approximation.

A Knowledge Graph is simply a structured map of business relationships (e.g., customers generate orders; orders contain revenue). In modern enterprise data infrastructure, this concept has evolved into a scalable and widely adopted standard: the Semantic Layer.

The Semantic Layer

As companies scale, their data volume and structural variety inevitably compound. This growth creates a usability bottleneck. No matter how efficiently the raw data is stored, operational users eventually hit a technical wall: they cannot query the raw database without help. To make the data usable, the modern stack needs a universal translator.

The solution is a *Semantic Layer*, which sits between the raw data (the cloud warehouse) and the tools people use (BI dashboards and AI agents).

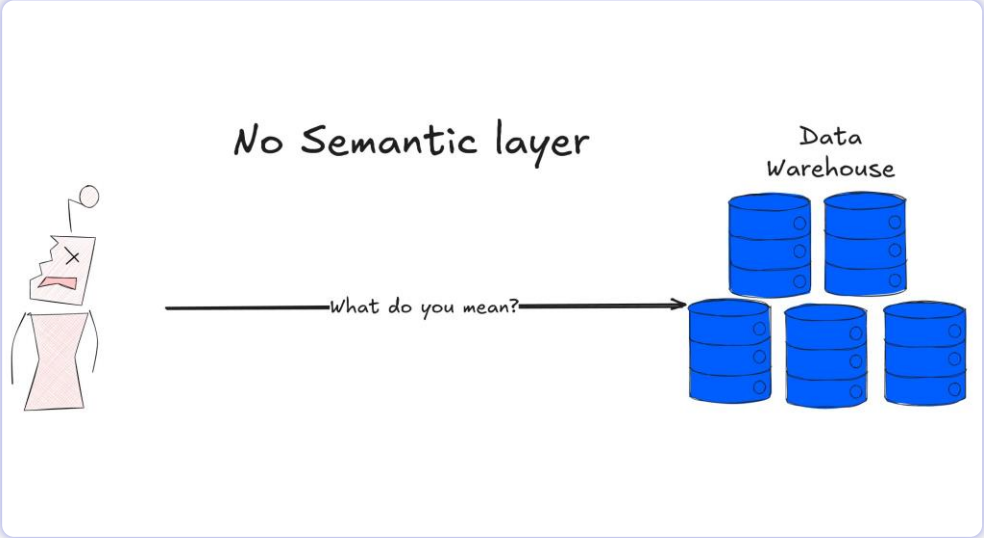


Instead of exposing users to messy, technical database schemas, the Semantic Layer maps raw data assets to governed, business-friendly concepts. It explicitly defines the relationships and mathematical rules behind metrics such as "net revenue" or "active users" in one centralized location.

By acting as a universal translator, the Semantic Layer abstracts the underlying SQL complexity. It ensures that business users and AI models interact only with trusted business metrics rather than raw tables.

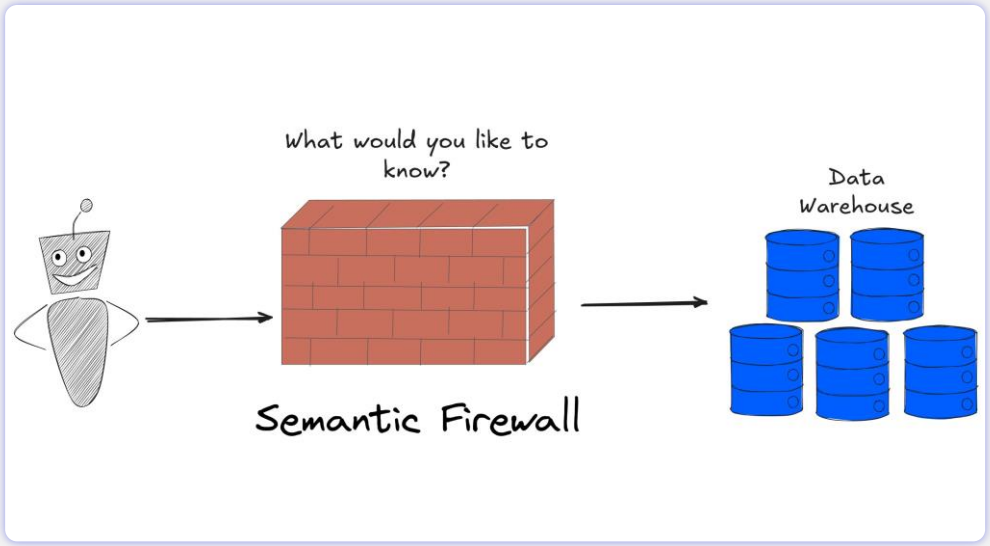
How the Semantic Layer Fixes Hallucinations

The primary cause of Text-to-SQL failure is the AI's inability to navigate chaotic database schemas. The Semantic Layer resolves this by acting as a structural shield between the AI and the database. The LLM no longer needs to calculate complex join paths, locate relevant tables, or parse abbreviated column names. All structural routing and data relationships are permanently hardcoded into semantic architecture.



By acting as the definitive source of truth, the Semantic Layer eliminates the need for the AI to fabricate business logic.

- The Raw SQL Risk: If a user asks for "total sales," a zero-shot LLM must guess which column joins, and filters apply, frequently resulting in a hallucination.
- Semantic Layer Fix: The AI simply maps the natural language request to the predefined Total Sales metric.



This architecture transforms the AI from a guessing machine into a deterministic execution engine. The AI is enriched with absolute business context: it strips out the ambiguity, guaranteeing every query returns an answer you can trust.

The Missing Link: Why Perfect Math Isn't Enough

The Semantic Layer successfully resolves the syntax and mathematical failure rate of raw Text-to-SQL. It guarantees that an AI model executes the correct calculation for governed metrics.

But enterprise analytics are rarely just about the numbers. They require human context. The Semantic Layer is a rigid calculator. It nails the *what*, but it is completely blind to the *why*.

Human data engineers survive on undocumented institutional memory. If an executive asks about "recent margin compression in the Northeast," the human engineer knows to reference a recent vendor contract (PDF) or a supply chain disruption currently being discussed in an operational Slack channel. A Semantic Layer cannot capture this unstructured reality. Without it, the AI remains locked out of the real, day-to-day business environment.

To bridge this final gap, the modern data stack is adopting a new piece of architecture: the Context Layer.

The Context Layer: Digitizing Tribal Knowledge

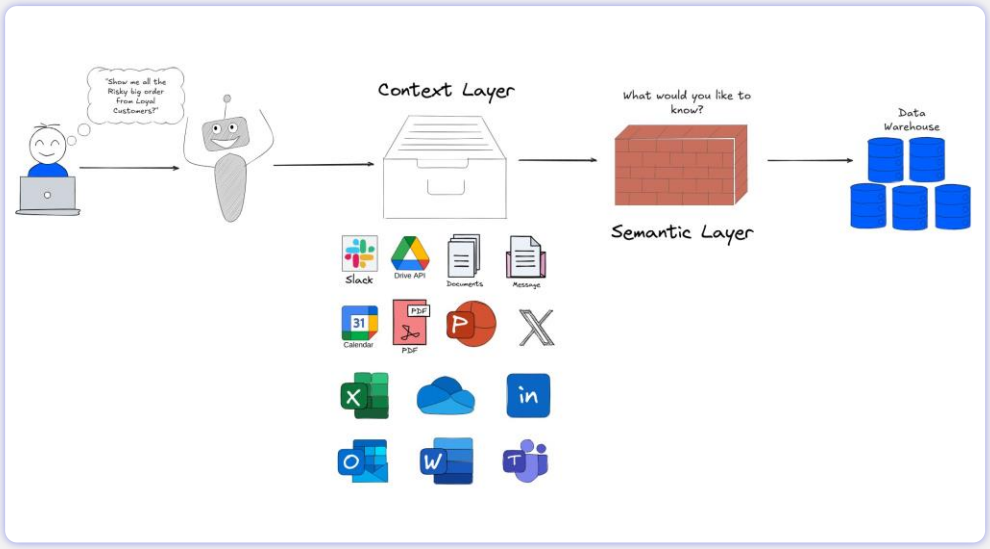
If the Semantic Layer is the enterprise's mathematical rulebook, the Context Layer is its digitized operational memory. It is an engine built exclusively to capture unstructured data and feed this directly to the LLM.

Rather than connecting to structured database tables, it ingests the messy, unstructured reality of how the organizations actually operate. This includes:

- **Communications:** Slack threads, Microsoft Teams logs, and email chains.
- **Documentation:** Confluence wikis, Notion pages, and data dictionaries.
- **Assets:** PDF contracts, Google Drive documents, and PowerPoint presentations.
- **Governance metadata:** Data lineage, ownership tags, and historical query logs.

Context Meets Semantics

True conversational analytics requires both layers working together before the query ever reaches the data warehouse.



The operational workflow transforms into a secure, two-step reasoning engine:

1. The Context Layer (Disambiguation): When a user asks, *"Why are the Big Apple accounts stalling?"*, the AI first queries the Context Layer. The LLM scans the ingested metadata, reading recent Slack threads and data dictionaries to determine that "Big Apple accounts" is internal slang for the New York enterprise segment, which recently underwent a pricing model change.

2. The Semantic Layer (Execution): The AI then submits a highly structured, governed request to the Semantic Layer, which deterministically maps the request to the exact SQL logic required to pull the revenue numbers for those specific accounts.

What Comes Next

Fixing Text-to-SQL is more than an engineering victory. It may be the catalyst for the end of traditional Business Intelligence as we know it. By transforming AI from a guessing machine into a deterministic execution engine, enterprises can finally deploy autonomous digital labor.

Next week, in Part II of our Cognitive BI series, we unpack the commercial implications: who wins, who loses, and why enterprise software spending is shifting from 'software access' to 'verified business outcomes.' We'll break down how Microsoft is gating its AI behind a $67,000 infrastructure tax, why Salesforce's Agentforce has only a 14% production deployment rate, and which AI-native startups are building the architecture designed to replace them.

---

[1] Yale, Spider 1.0 Yale Semantic Parsing and Text-to-SQL Challenge, 2025

[2] Yale, Spider 2.0 Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows, 2026

[3] Bird-SQL, A big Bench for Large-Scale Database Grounded Text-to-SQLs, 2026

[4] Endsight, IT Support Help Desk Metrics and Benchmarks, 2025

[5] Straits Research, Business Intelligence Market Overview, 2026

[6] data.world, A Benchmark to Understand the Role of Knowledge Graphs on Large Language Model's Accuracy for Question Answering on Enterprise SQL Databases, 2023

[7] Ibid