Knowledge Graph Thinking

Idea In Short

Most business data sits in tables, one per system, each with its own labels and rules. Questions that matter often cross those tables: which suppliers feed the products that our largest customers rely on, and what would a port closure do to them? Knowledge graph thinking answers by treating data as a network of entities and the relationships between them. Nodes stand for things such as customers, products, and places, and labeled edges say how they connect. The approach gained wide attention after Google described its own graph in 2012, though its roots lie in older work on semantic networks. This article explains the building blocks, shows how to model a domain, and covers the costs and risks.

A manufacturer learns that a fire has shut down a supplier's plant. The purchasing system lists the supplier. The product system lists parts. The sales system lists customer orders. No single table can say which customers are exposed. A graph that links supplier to part to product to order answers in seconds.

What a knowledge graph is

A knowledge graph is a structured way to describe how things connect. IBM describes it as a network of real-world entities, such as objects, events, situations, or concepts, that also illustrates the relationships between them. It is usually stored in a graph database and shown as a graph structure, and it has three main components: nodes, edges, and labels.1

The IEEE puts the same idea in slightly more technical terms. Entities appear as nodes, relationships appear as labeled edges, and a schema or ontology gives the labels defined meaning. The IEEE page adds that the term entered general use after Google described its own graph in 2012, while the underlying ideas descend from semantic networks and frame systems developed in artificial intelligence research from the 1960s onward.2

So the idea is not new, though its use in business has grown. Managers now meet it in

recommendation systems, search, fraud detection, and the AI tools that rely on context.

Entities, relationships, and rules

SAP's explanation breaks a business knowledge graph into three elements. Entities say what exists, such as customers, products, assets, and employees. Relationships say how they connect, for example buys, manages, supplies, or belongs to. Rules capture business logic and constraints that keep the graph consistent.3

Consider how this differs from a plain transaction record. A row that says a customer bought a product records an event. A graph models the meaning: this customer, in this region, belongs to this segment, bought this product, which contains this part, which comes from this supplier.

The richer picture lets you travel along connections. You can ask for all customers within two steps of a failing supplier, or for all products that share a part with a recalled one.

Why graphs beat tables for connected questions

Tables handle questions about one kind of thing very well. They struggle when a question needs many hops across kinds of things. Each hop needs a join, and long chains of joins become slow and hard to write.

A Neo4j explainer describes a knowledge graph as an organized representation of real-world entities and their relationships, typically stored in a graph database that natively stores the relationships between data entities. The relationships capture the context and meaning of how the entities are connected.4

Because relationships are stored directly, following them is natural. That is why graphs suit supply chains, organizations, customer networks, and any area where the connections carry the answers.

Connecting silos

Many firms hold data in separate systems for sales, finance, operations, and support. Loading everything into one store does not by itself make the data meaningful. A Siemens

blog on enterprise knowledge graphs argues that copying data into a common store does not deliver holistic integration, and that knowledge graphs can describe, contextualize, and link data across enterprise sources, structured and unstructured, providing a common model for enterprise data assets.5

The key word is common model. When every system calls a customer something different, a graph needs a shared definition of what a customer is. Building that definition is often the most valuable part of the work, because it forces departments to agree.

Start by linking identifiers. Match the customer in the sales system to the one in billing and support. Even this basic step lets you answer questions that no single system can.

Thinking in graphs without software

You can practice the mindset on a whiteboard. Pick a question, such as "Which of our projects depend on the same few people?" Draw the people, the projects, and the lines between them.

Name the lines with verbs: leads, contributes to, depends on, approves. Look at the picture. Hubs, the nodes with many connections, show concentration risk. Isolated nodes may show neglected assets or missing links.

This kind of drawing resembles a concept map, though a graph aims at data that software can store and query. Use the whiteboard version to decide what you would need to capture before investing in a system.

Designing the model

Begin with a use case. A good one has a concrete question and a clear owner, such as supply risk, customer 360, or compliance tracing. Broad ambitions to graph everything usually stall.

Define entities and relationships carefully. Use clear names and one meaning per name. Decide which attributes belong on each entity and which relationships matter for your questions.

Keep the first model small. Add entity types only when a question needs them. A graph that grows without discipline becomes a tangle that nobody trusts.

SAP's guidance emphasizes the same sequence: start with a key use case in a domain such as customers, products, or supply chains, then define entities and relationships and build or adopt an ontology that reflects the business.

Knowledge graphs and AI

Knowledge graphs have drawn attention because they give AI systems context. SAP argues that AI is only as good as the data it understands, and that without context models can produce errors. A graph can ground such systems in the business by supplying how things relate.

The practical benefit is traceability. When an assistant answers a question using graph facts, you can follow the path of entities and relationships it used and check it. That is harder with a pile of unlinked documents.

Treat these claims with care. A graph improves context only if the data in it are accurate and the definitions are right. A graph fed with bad data spreads errors along every edge.

Questions a graph can answer

The value of a graph shows in the questions it makes cheap. Which customers share the same supplier chain? Which employees hold knowledge that nobody else has? Which contracts reference a clause that a regulator has just changed? Each of these crosses several kinds of records.

Write your own list of five such questions before you build anything. If a table or a simple report already answers them, you may not need a graph. If the answers today take days of manual work across teams, a graph deserves a trial.

Measure the result in time and confidence. Compare how long a cross-system question took before with how long it takes on the graph, and track how often the answers needed correction. Those two figures show whether the effort pays off.

A hypothetical example

Imagine a hypothetical hospital network that wants to know which patients may be affected by a faulty batch of infusion pumps. The equipment records sit in one system, the maintenance logs in another, and the ward schedules in a third.

A small team builds a graph with pumps, batches, wards, shifts, and patients, linked by relationships such as installed in, used during, and treated with. Within an hour of the alert, the team lists the wards and patients who had contact with the batch. The case is invented, but it shows how connected data supports fast answers.

Cautions

Do not build a graph without a question. Collecting everything in a network feels productive and delivers little.

Avoid vague relationship labels. If every edge says "related to," the graph holds links but no meaning, and neither people nor software can use it well.

Finally, plan for governance. Assign owners to entity types, set quality checks, and track where each fact came from. A graph is only as trustworthy as the data and the rules behind it.

- 1What is a knowledge graph?
- 2Knowledge graphs, IEEE Technology Navigator
- 3What is a knowledge graph?
- 4What is a knowledge graph?
- 5Enterprise knowledge graphs to the rescue

Summary

Knowledge graph thinking means representing information as entities and labeled relationships, organized by a schema or ontology that gives the labels meaning. The structure lets people and software follow connections across systems, ask questions that

cut across silos, and infer facts that no table states directly. It works best when you start with a concrete use case, define entities and relationships carefully, and keep the model as simple as the questions allow. The approach also demands governance, since a graph that merges wrong data spreads errors quickly. Choose one cross-system question your team struggles to answer, list the entities and relationships involved, and sketch them as a graph on a whiteboard.