Montanuniversität Leoben
Wirtschafts- und Betriebswissenschaften
Leitung: Univ.-Prof. Dipl.-Ing. Dr. mont. Wolfgang Posch

Franz Josef Straße 18 • A-8700 Leoben • Tel +43 (0) 3842 402 6001 • Mail wbw@unileoben.ac.at

WBW Insight

White Paper Nr. 3 (06/2026)

Titel	**Knowledge Graphs for Engineers. A Practical Guide for Energy Systems.**
Autor/en	**Dr. Nikolaos C. Poulios**
Inhalt / Abstract	This monograph provides a rigorous yet practical introduction to knowledge graphs for engineers, with a focus on energy and industrial systems.
Keywords	Knowledge Graphs, Graph Theory, RDF, SPARQL, Energy Systems, Data Integration, Decision Support Systems, Industrial Analytics

Knowledge Graphs for Engineers

A Practical Guide for Energy Systems

Dr. Nikolaos C. Poulios

Technical University of Leoben
Austria

June 8, 2026

Contents

List of Acronyms

AI Artificial Intelligence. 5

API Application Programming Interface. 5

CMMS Computerized Maintenance Management System. 5, 8

CSV Comma-Separated Values. 5

DB Database. 5

ERP Enterprise Resource Planning. 5, 8

ETL Extract, Transform, Load. 5

GNN Graph Neural Networks. 5

HVAC Heating, Ventilation, and Air Conditioning. 5

IoT Internet of Things. 5, 9

IRI Internationalised Resource Identifier. 5

ISWC International Semantic Web Conference. 5

JSON JavaScript Object Notation. 5, 9

KG Knowledge Graph. 5–7

KPI Key Performance Indicator. 5

ML Machine Learning. 5

OWL Web Ontology Language. 5, 6

. 5

PHM Prognostics and Health Management. 5

RDBMS Relational Database Management System. 5, 15

RDF Resource Description Framework. 5, 6

RDFS RDF Schema. 5

REST Representational State Transfer. 5

SCADA Supervisory Control and Data Acquisition. 5, 8, 9, 11

SPARQL SPARQL Protocol and RDF Query Language. 5, 6

SQL Structured Query Language. 5, 15

URI Uniform Resource Identifier. 5

URL Uniform Resource Locator. 5

XML eXtensible Markup Language. 5, 9

Chapter 1

Introduction to Knowledge Graphs

1.1 Why Knowledge Graphs matter for energy engineers

Modern engineering systems are increasingly shaped by digitalization, data integration, and complex interdependencies between technical, economic, and regulatory components. In domains such as energy, infrastructure, manufacturing, and industrial operations, decision-making depends on the ability to connect heterogeneous sources of information into coherent models. Knowledge graphs have emerged as a powerful approach to represent such interconnected systems in a structured and semantically meaningful way. Modern energy systems combine physical infrastructure, digital monitoring systems, market coordination mechanisms, and policy constraints into a highly interconnected operational environment. The distributed nature of these elements and their mutual dependencies creates structural complexity that cannot be fully understood through isolated data tables. Engineers and decision-makers must navigate data from sensors, control systems, market dynamics, regulations, and environmental metrics. Knowledge graphs offer a flexible, scalable solution to integrate these diverse sources into a unified, machine-readable format that supports analytics, diagnostics, and operational excellence. By adopting knowledge graph approaches, engineering and business-oriented research environments can enhance data integration, structural modelling, and the development of advanced decision-support systems. What do we mean by a Knowledge Graph? A knowledge graph (KG) is a structured representation of knowledge in the form of a graph. It consists of entities (nodes) and relationships (edges) that connect these entities in a meaningful way. Unlike traditional databases that store data in tables, knowledge graphs model real-world domains more intuitively by capturing the semantics of how different entities relate to one another. In practice, the term "knowledge graph" is used in two slightly different ways. The first interpretation, often referred to as the Semantic Web–style knowledge graph, is grounded in formal logic and standardized technologies such as RDF,RDFS,

OWL, and SPARQL. In this framework, entities are identified by globally unique IRIs, relationships are explicitly defined, and ontologies provide formal semantic constraints. Reasoning and inference mechanisms can derive new knowledge from explicitly encoded rules. The second interpretation, frequently called the Google-style knowledge graph, emphasizes large-scale information integration and retrieval. These systems focus primarily on linking entities across massive datasets to enhance search, recommendation, and ranking mechanisms. They often use graph databases or property graph models and may rely more on machine learning than on formal logical reasoning. Furthermore, Ontologies play a central role: they specify classes, properties, and constraints that give meaning to the data. The emphasis of this approach is semantic precision and interoperability. Because the data model follows formal logic principles, reasoning engines can infer new knowledge from existing triples. This makes Semantic Web–style graphs particularly suitable for domains where consistency, traceability, and structured integration across heterogeneous systems are essential. In contrast, Google–style knowledge graphs emerged primarily from large-scale information retrieval and search applications. Their focus is not formal logical reasoning but large-scale entity linking and contextual relevance. These systems connect entities across vast datasets to enhance search results, recommendation systems, and user experience. Although they may rely on graph structures internally, the design philosophy emphasizes scalability, performance, and practical usefulness rather than strict semantic formalism. Machine learning techniques often complement these graphs to improve ranking, recommendation, and entity disambiguation. Moreover, in this book, I primarily adopt a Semantic Web–oriented interpretation of knowledge graphs, grounded in RDF, ontologies, and formal graph representations. At the same time, we acknowledge practical industrial implementations that combine semantic modelling with scalable graph database technologies and advanced analytical approaches. In both cases, knowledge graphs aim to represent, integrate, and utilize data with rich semantics to support discovery, reasoning, and intelligent automation.

1.2 Historical Context and Evolution

The concept of knowledge graphs builds on earlier ideas in semantic networks, expert systems, and linked data. The Semantic Web movement in the early 2000s formalized the use of RDF and OWL, leading to large-scale efforts such as DBpedia and Wikidata. Commercial adoption surged in the 2010s with Google's launch of its own knowledge graph and industry recognition of the need for data integration across silos. In recent years, their adoption has expanded beyond web data integration toward engineering applications, industrial knowledge management, and infrastructure modeling.

1.3 Components of a Knowledge Graph

A typical knowledge graph consists of:

- Entities: The things or objects of interest (e.g., a person, product, sensor, or process).
- Relationships: Directed, labeled edges that connect entities (e.g., "produces", "locatedIn", "supervises").
- Attributes: Properties of entities (e.g., name, date, weight, height, category).
- Triples: The atomic unit of a KG, expressed as (subject, predicate, object), e.g., (`Pump30`, `hasStatus`, `Operational`).

These elements form a flexible and expressive structure that supports complex querying, inference, and integration.

The use of triples provides a semantic layer that makes knowledge graphs not only machine-readable but also interpretable by software agents for automated reasoning and decision support. The extensibility of this structure allows engineers to continuously evolve their data model without disruptive changes.

1.4 Why Knowledge Graphs?

The idea of a knowledge graph may initially appear technical. In practice, however, the need for it arises from very concrete problems. Engineering environments are shaped by fragmented information. Operational data is stored in one system, financial information in another, regulatory documentation somewhere else. Each system captures part of the reality, yet rarely the whole.

In energy systems, this fragmentation becomes particularly visible. Technical components such as transformers, turbines, and substations are connected not only physically, but also through contracts, maintenance schedules, safety regulations, and performance targets. These relationships exist in reality even when they are not explicitly modeled in a data system.

A knowledge graph addresses this gap. Instead of organizing information around isolated records, it organizes information around relationships. Entities are connected through meaningful links. When these links are made explicit, it becomes possible to ask questions that follow chains of dependencies rather than single data points.

Consider a simple example. If a transformer fails, the immediate technical consequences may be clear. Yet the broader implications—affected customers, contractual obligations, maintenance history, regulatory reporting requirements—are often distributed

across systems. In a graph-based model, these connections can be traced directly. The structure of the model mirrors the structure of the system.

Flexibility is another important aspect. Traditional database schemas require careful planning and are often difficult to modify once deployed. Engineering systems, however, evolve continuously. New asset types are introduced. Regulations change. Monitoring technologies improve. Knowledge graphs allow new relationships or entity types to be incorporated without redesigning the entire structure.

Knowledge graphs also support structured reasoning. When ontologies define classes and constraints, automated checks become possible. For example, inconsistencies between maintenance procedures and regulatory standards can be detected systematically. This does not replace expert judgment, but it supports it.

For modern engineering systems that are increasingly interconnected and data-intensive, this relational modeling approach is not simply convenient. It reflects the underlying structure of the domain itself.

1.5 Real-World Applications

Several major companies globally and energy-focused organizations use KGs:

- Siemens Energy: Uses KGs to model assets and automation systems in industrial power plants
- Shell and BP: Apply KGs to link technical specifications, operating procedures, and regulatory frameworks
- Enel and E.ON: Leverage KGs for grid asset management, outage response, and predictive maintenance

In these applications, KGs support:

- Better integration across SCADA, ERP, CMMS, and monitoring systems.
- Enhanced asset tracking and maintenance workflows.
- Risk analysis and compliance with environmental and safety regulations.
- Transparent, data-driven decision support for energy planning and operations.

In the energy sector, for example, knowledge graphs are being applied to map the connectivity of grid components, monitor compliance with safety standards, and integrate sensor data with operational models. In manufacturing, they facilitate product lifecycle management by linking design specifications with test results, supplier metadata, and customer feedback.

1.6 Knowledge Graph Architecture

A knowledge graph is more than a dataset stored in a graph database. It is part of a broader system that connects sources of information, conceptual models, storage technologies, and analytical applications. Looking at its architecture makes this clear.

Every knowledge graph begins with data. In engineering environments, these data sources are rarely uniform. Some are structured, such as relational databases. Others are semi-structured, including XML or JSON files. Many are unstructured, such as technical documentation, inspection reports, or maintenance logs. In energy systems, additional streams often come from SCADA platforms or IoT sensors. Each of these systems captures a fragment of reality.

Before the data can be connected, it must be aligned. Entities described in one system must be recognized in another. A turbine recorded in a maintenance tool should correspond to the same turbine measured in an operational database. This stage involves extraction, cleaning, and linking. Without it, a knowledge graph would simply reproduce fragmentation in a different format.

At the conceptual level, the ontology plays a central role. It defines what kinds of entities exist, how they relate, and what constraints apply. The ontology does not store data; it provides structure. It ensures that the graph is not only connected, but coherent.

The structured information is then stored in a graph database or triple store. These systems are designed to handle relationships as first-class elements. Instead of reconstructing connections through joins, engineers can navigate directly through the network.

Above this storage layer sits the query engine. In RDF-based systems, this is typically SPARQL. Queries describe patterns of relationships rather than combinations of tables. This reflects how engineers often reason about systems: by following connections rather than by assembling records.

Finally, applications interact with the knowledge graph. These may include dashboards, reporting tools, optimization models, or AI components. The graph provides context. It does not replace operational systems; it connects them.

Seen as a whole, the architecture forms a pipeline. Raw data is transformed into structured relationships, which in turn support analysis and decision-making. For engineers, this architectural perspective clarifies that a knowledge graph is not an isolated technology. It is an integrating layer within a broader technical ecosystem.

1.7 Benefits for Business Engineers

Engineers who operate between technical infrastructure and economic decision-making often encounter a structural problem: information exists, but it does not exist together. Operational data, financial metrics, compliance records, and maintenance histories are

typically stored in separate systems. A knowledge graph offers a way to represent these relationships explicitly.

This becomes visible at the level of system-wide transparency. Energy systems involve generators, transmission assets, market participants, regulators, and service providers. These elements influence one another. When modeled in isolation, important dependencies remain hidden. When modeled as a graph, these dependencies become part of the structure itself.

Interoperability is another practical consequence. SCADA platforms, IoT devices, ERP systems, and maintenance software are rarely designed to communicate seamlessly. Rather than replacing these systems, a knowledge graph introduces a shared semantic layer. It allows information to be linked without forcing a redesign of existing infrastructure.

The benefits are not only operational. When a failure occurs, tracing its impact requires understanding how components depend on each other. In a graph structure, these dependencies are already present. Engineers can follow them directly instead of reconstructing them manually across multiple tables.

Over longer time horizons, the same structure supports strategic objectives. Emission data, asset performance indicators, and regulatory requirements can be connected within a single framework. This supports sustainability reporting and investment planning in a more consistent manner.

Resilience analysis provides another example. Energy networks are interdependent. A disturbance in one location may propagate elsewhere. By modeling these connections explicitly, engineers can analyze possible cascading effects. The graph does not eliminate uncertainty, but it provides a structured way to examine it.

Taken together, these aspects show that knowledge graphs are not merely a storage solution. They reflect the structural nature of modern engineering systems. For business engineers in particular, this integration of technical and economic relationships supports both operational improvements and long-term planning.

1.8 Summary

Knowledge graphs provide a framework for representing interconnected systems in a structured and meaningful way. By modeling entities and relationships explicitly, they move beyond isolated data storage and toward integrated system representation.

In engineering contexts, particularly in complex infrastructures such as energy systems, this approach aligns naturally with how components interact in practice. Technical assets, economic processes, regulatory constraints, and operational data form networks of dependencies. A graph-based model makes these dependencies visible.

The chapter has introduced the conceptual foundations of knowledge graphs, outlined

their historical development, and described their core architectural components. It has also examined their relevance for engineers who operate at the intersection of technical systems and strategic decision-making.

To understand how these structures function mathematically, the next chapter turns to the fundamentals of graph theory. These principles form the structural backbone upon which knowledge graphs are built.

Chapter 2

Graph Theory for engineers

In many areas of engineering, especially in the energy sector, we work with systems made up of interconnected components. Think of power grids, sensor networks, supply chains, or even the flow of information within an organization. These are all networks, and the most natural way to represent them is through graphs.

A **graph**, in its simplest form, is a collection of points (called *nodes* or *vertices*) and lines connecting them (called *edges*). Though simple in concept, graphs are powerful tools for modeling and analyzing complex systems.

Graphs help engineers understand structure, identify possible or potential vulnerabilities, simulate flows (like electricity or data), and design better infrastructure. They provide not only visual representations but also the computational foundation to explore and optimize systems.

Moreover, graphs serve as a common language across disciplines. Once you're familiar with the basic components, you can apply graph theory to mechanical systems, communication networks, data infrastructures, and logistics chains.

2.1 Basic Concepts in Graph Theory

Let's define the key terms and ideas used in graph modeling:

- **Nodes (vertices):** Entities in your system, such as a transformer, sensor, or data server.
- **Edges (links):** Connections between entities, such as a power line, communication cable, or control signal.
- **Directed vs. Undirected Edges:** A *directed edge* models a one-way relationship (e.g., control from A to B), while an *undirected edge* represents a mutual or bidirectional link.

- **Weighted Edges:** Gives special meaning to an edge, such as 'more' or 'less' important. Edges that carry a numerical value (e.g., cost, resistance, or capacity).
- **Paths and Cycles:** A *path* is a sequence of edges connecting nodes; a *cycle* is a path that returns to the starting node.
- **Connectivity:** A graph is *connected* if there is a path between every pair of nodes.

Other graph types include:

- **Trees:** Graphs with no cycles. Often used to model hierarchical distribution systems.
- **Bipartite Graphs:** Graphs with two sets of nodes, where edges only connect nodes from different sets.
- **Multigraphs:** Allow multiple edges between the same pair of nodes.

2.2 Examples from Energy Systems

Graphs naturally model many real-world systems in the energy domain:

- **Power Grids:** Nodes represent substations or transformers; edges are transmission lines. Graph analysis helps in fault tolerance, load balancing, and resilience planning.
- **SCADA Systems:** Nodes are sensors and control units; edges represent information flow. Graphs reveal bottlenecks and control hierarchies.
- **Energy Markets:** Entities like producers, regulators, and consumers are modeled as nodes; edges represent pricing relationships or contractual ties.
- **Hydrogen or Battery Infrastructure:** Storage points, converters, and loads form a network where graphs help optimize flow and predict failure points.

The above examples show that graph structures are not just another visualisation tool; however, they enable deep analysis and smarter decisions.

2.3 Important Graph Properties for Engineers

By applying graph theory as a tool engineers can extract meaningful insights from them, and usually compute specific properties. The most important are:

- **Degree:** The number of connections a node has. High-degree nodes are often hubs or critical components.
- **Centrality:** By centrality, we assign a number to a node in order to determine the importance of that node. In other words, measures a node's influence in the network. In centrality, we have different types such as:
  - **Betweenness centrality:** Nodes that lie on many shortest paths.
  - **Closeness centrality:** Nodes that can quickly reach all others.
  - **Eigenvector centrality:** Nodes that are connected to other well-connected nodes.
- **Clustering Coefficient:** Indicates how connected a node's neighbors are—helps detect modularity in systems.
- **Shortest Paths:** Algorithms such as Dijkstra's compute efficient paths for routing and restoration.
- **Connected Components:** Identifies isolated sub-networks—a red flag in reliability planning.
- **Flow and Coloring:** Used in tasks like network optimization, load balancing, or scheduling.

Understanding these properties allows engineers to move from descriptive diagrams to actionable insights.

2.4 A toy example

To better understand the practical value of these concepts, let's consider a small toy example based on an energy substation network.

Definition 2.4.1 *A graph is an ordered pair $G = (V, E)$ where:*

- *V is a finite set of vertices (or nodes),*
- *$E \subseteq V \times V$ is a set of edges.*

If $(u, v) \in E$ implies $(v, u) \in E$, the graph is called undirected. Otherwise, it is directed.

Remark. In the following toy example, although the system may represent directional flows (e.g., energy transfer), we adopt the undirected convention when computing clustering coefficients. This is standard practice, since triangles and local connectivity are typically defined for undirected graphs.

Imagine the following simplified system:

- Node A: Central transformer station
- Node B: Local substation 1
- Node C: Local substation 2
- Node D: Wind turbine input
- Node E: Battery storage unit

The connections (edges) between them are:

- A $\rightarrow$ B
- A $\rightarrow$ C
- D $\rightarrow$ A
- E $\rightarrow$ A
- B $\rightarrow$ C (backup path)

Now, let's apply some graph properties:

- **Degree**: Node A has the highest degree (4 connections), which means that it's a central hub.
  - In-degree (incoming edges) and out-degree (outgoing edges):
    - A: in = 2 (D, E), out = 2 (B, C)
    - B: in = 1 (A), out = 1 (C)
    - C: in = 2 (A, B), out = 0
    - D: in = 0, out = 1
    - E: in = 0, out = 1
  - Thus, node A is the hub, and C is a sink i.e. no outflow.
- **Betweenness centrality**: A lies on all paths between the generation sources (D, E) and consumers (B, C), so it's highly central.
  - By betweeness centrality we measure the frequency a node is on the shortest paths between other nodes.
  - The node A lies on all shortest paths from D or E to B or C.
  - The node B lies on the backup path from D,E $\rightarrow$ A $\rightarrow$ B $\rightarrow$ C, so it's sometimes involved.

  - In this case, we can say that node A has the high betweeness, while B has a moderate.

- **Clustering coefficient**: By cluster coefficient we are interested to know the frequency that two nodes that are connected are part of some kind of larger highly connected group of nodes. The formula to calculate the coefficient is

$$CC(v) = \frac{2N_v}{K_v(K_v - 1)} \tag{2.1}$$

  Where

  - v is a node
  - K_v is its degree
  - N_v is the number of links between neighbours of v.

  In the above toy example, B and C are connected via a backup link, forming a triangle with A. This adds redundancy and resilience.

Numerical Example for the Toy Graph

We now compute the clustering coefficient for vertex A in the undirected toy graph.

Vertex A is connected to the vertices B, C, D, and E. Hence, the degree of A is

$$k_A = 4.$$

The maximum number of edges that could exist among these four neighbors is

$$\binom{4}{2} = 6.$$

The possible pairs of neighbors are:

$$(B, C),\ (B, D),\ (B, E),\ (C, D),\ (C, E),\ (D, E).$$

From the structure of the graph, only the edge (B, C) exists among these neighbors. No other edges exist among these neighboring vertices.

Therefore, the number of existing links among the neighbors of A is

$$N_A = 1.$$

Using the clustering coefficient formula

$$C(A) = \frac{2N_A}{k_A(k_A - 1)},$$

we obtain

$$C(A) = \frac{2 \cdot 1}{4 \cdot 3} = \frac{2}{12} = \frac{1}{6}.$$

Thus, the clustering coefficient of vertex A is

$$C(A) = \frac{1}{6}.$$

This relatively small value indicates that most neighbors of A are not directly connected to one another. The only local redundancy arises from the connection between B and C, which forms a single triangle with A.

- **Shortest path**:

  In an undirected graph, the shortest path between two vertices is the path with the minimum number of edges connecting them. Since the toy graph is unweighted, the distance between two nodes corresponds to the number of edges along the path.

  In our example:

  - The shortest path from D to C is:

    $$D - A - C,$$

    which has length 2.
  - The shortest path from E to C is:

    $$E - A - C,$$

    which also has length 2.
  - The shortest path from D to B is:

    $$D - A - B,$$

    which has length 2.
  - An alternative path from D to C is

    $$D - A - B - C,$$

which has length 3 and is therefore not optimal.

Thus, the central position of vertex A ensures that most shortest paths between outer nodes pass through it. This reflects the structural role of A as a hub in the network.

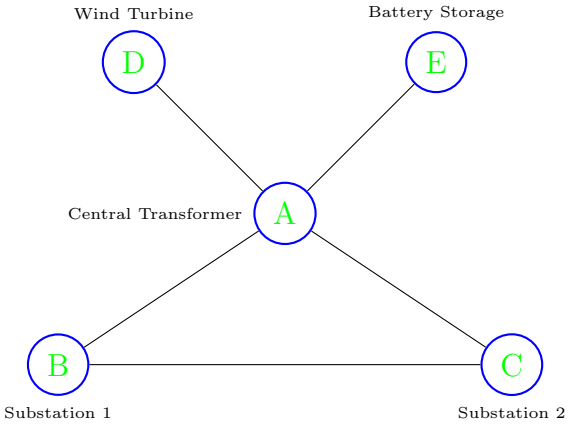

Figure 2.1: Simplified energy network graph with five nodes and undirected edges.

2.5 Why Engineers Should Care

Graph theory may appear abstract at first glance, but its relevance to engineering practice is direct and practical. Many systems that engineers design, operate, and analyze can be understood as networks of interconnected components. Power grids, communication systems, sensor networks, transportation infrastructures, and supply chains all exhibit relational structure. Graph theory provides a formal language to describe and analyze these structures.

One immediate benefit lies in identifying critical components. In networked systems, not all nodes play the same role. Some act as hubs, others serve as connectors between subsystems, and some may be peripheral. Measures such as degree and centrality allow engineers to quantify these roles. This is particularly important in energy systems, where failure of a highly central node can have widespread consequences.

Graph-theoretic concepts also support resilience analysis. By examining connectivity, shortest paths, and clustering, engineers can better understand how disturbances may propagate through a network. In an energy grid, for example, local redundancy can reduce the impact of a single failure. The ability to evaluate such structural properties provides insight into system robustness.

Beyond fault analysis, graph theory assists in optimization tasks. Problems such as routing, load balancing, scheduling, and resource allocation often reduce to finding

optimal paths or flows within a network. Algorithms like Dijkstra's method or maximum flow techniques translate theoretical concepts into operational tools.

Graph theory further supports visualization and conceptual clarity. Complex systems can be difficult to understand when represented only through tables or isolated diagrams. A graph representation highlights relationships explicitly, revealing structural patterns that may not be obvious otherwise.

For business engineers, this perspective is particularly valuable. Technical infrastructure, financial flows, regulatory constraints, and operational processes are not independent layers; they interact. Viewing these interactions through a graph-theoretic lens encourages systemic thinking rather than isolated optimization.

Ultimately, graph theory does not replace engineering expertise. Instead, it strengthens it by providing mathematical tools to reason about interconnected systems. In the following chapters, this structural understanding will serve as the foundation for knowledge graphs and their application in engineering environments.

2.6 Looking Ahead

Graph theory provides a formal framework for analyzing interconnected systems. In this chapter, we examined fundamental concepts such as degree, centrality, clustering, and shortest paths, and we applied them to a simple toy example. These notions are not isolated mathematical curiosities; they describe structural properties that appear in many engineering environments.

The purpose of this discussion has been to establish a structural mindset. Rather than viewing systems as collections of independent components, graph theory encourages us to think in terms of relationships. Dependencies, flows, redundancies, and bottlenecks become explicit once the system is modeled as a network.

In the next chapter, we shift from abstract graph structures to data modeling. We will examine how traditional relational databases represent information and where they encounter limitations when dealing with highly interconnected data. This will naturally lead to the introduction of graph-based data models and, ultimately, knowledge graphs.

The mathematical foundations developed here will serve as a reference point. As we move toward semantic technologies and data integration frameworks, the structural principles of graph theory will remain central.

Chapter 3

From Relational Databases to Graphs

3.1 Introduction

For decades, relational databases have been the backbone of data management in engineering and business. They offer a structured, well-understood model built around tables, columns, rows, and keys. But as systems become more interconnected and data becomes more contextual, engineers are often forced to jump through hoops to extract relationships that should be obvious.

This chapter explores why and how we move from the rigid world of relational databases to the more flexible and expressive world of graph-based models.

3.2 Relational Databases: A Pillar of Engineering

Before we critique, we must acknowledge its strengths. The relational model, as defined by E.F. Codd, organizes data into relations (tables). Each table has a fixed schema—a set of columns with defined data types. Rows represent individual records. Relational database management systems provide several mature guarantees. They support ACID transactions, ensuring that operations are executed atomically, consistently, and durably. They enforce a schema-on-write discipline, meaning that data must conform to predefined structures before being stored. Through SQL, engineers formulate declarative queries, specifying what results they need while the database engine determines how to compute them efficiently. Finally, normalization principles help reduce redundancy and preserve integrity across large and complex systems. For systems with well-understood, stable, and highly structured data, the RDBMS remains an excellent choice. The problem arises when the nature of the data and the questions we ask of it begin to chang

Relational databases are built around tables. Each table stores data of a certain

type—say, a list of energy assets, a list of technicians, or a schedule of maintenance checks.

A typical schema might include:

- A table of transformers (ID, capacity, location)
- A table of inspections (date, technician ID, transformer ID)
- A table of technicians (name, qualification level)

Tables are connected using foreign keys, and SQL is used to join them. This is fine for structured data, but real-world relationships are often more complex.

3.3 Where Relational Models Fall Short

Some major challenges include:

- **Complex joins:** Many-to-many relationships quickly become difficult to query.
- **Rigid schema:** Adding new relationships requires schema changes.
- **Context loss:** Tables capture data, but not meaning.
- **Poor performance for relationship queries:** Traversing relationships is slow and clunky.
- **Lack of visualization:** Tabular formats aren't intuitive for networks.

Relational databases are exceptionally good at handling structured and repetitive data. Difficulties arise, however, when relationships become complex or dynamic. The reason lies in their design philosophy. Each table is an isolated entity; relationships among tables must be expressed indirectly through keys and joins. This model works well for accounting systems or inventory lists, but it becomes cumbersome when data items are deeply interconnected or change frequently.

In engineering contexts, such complexity is the rule rather than the exception. Energy assets, maintenance actions, and environmental conditions form webs of dependency that are difficult to capture in a rigid schema. As soon as a system must represent interactions, influences, or hierarchical connections, the relational model begins to show its limits.

A simple example can illustrate this. Suppose an engineer wants to trace the chain of maintenance events that led to a particular transformer's failure. In a relational schema, this information might be distributed across several tables—`Transformers`, `Inspections`, `Technicians`, `Replacements`, and `SpareParts`. To reconstruct the sequence, the engineer must join all these tables on matching keys and then filter by date.

As the number of joins grows, the query becomes difficult to write, slow to execute, and almost impossible to interpret visually. The system answers the question, but at the cost of clarity and performance.

Another challenge emerges when relationships evolve over time. Suppose the organization introduces a new rule that each inspection must now reference not only the technician who performed it but also the supervisor who approved it. In a relational model, this seemingly minor change requires altering the schema, updating foreign keys, and rewriting existing queries. In a graph model, the new connection could simply be added as another edge between existing nodes without affecting the rest of the structure.

Relational databases also hide the meaning of relationships. A foreign key does not tell you *why* two tables are linked; it merely enforces referential integrity. For instance, a link between the `Inspections` and `Transformers` tables says that a given inspection involves a certain transformer, but it cannot express whether that inspection detected a fault, replaced a component, or performed a routine check. To store such semantics, the engineer must add more tables or columns, further complicating the schema.

Scalability presents yet another problem. Queries that involve multiple joins grow nonlinearly with data size. As datasets expand, performance degrades sharply. For example, retrieving "all substations within three hops of a transformer experiencing overload" would require recursive self-joins in SQL, a process that becomes prohibitively slow at scale. In contrast, a graph database would treat this as a simple traversal across three edges.

Visualization is another weak point. While relational tables excel at storing data, they do not convey how entities are connected. When troubleshooting a system, engineers often need to see dependencies rather than lists of records. Attempting to represent those relationships through tables or pivot charts quickly becomes confusing.

Finally, relational models are ill-suited to situations where partial or missing information must still be represented. If a transformer's manufacturer is unknown or a maintenance record lacks a timestamp, relational systems often force these gaps into placeholder values. Graphs, by contrast, can simply omit the edge or add a temporary node, preserving consistency without distortion.

These shortcomings do not mean that relational databases are obsolete—only that they were designed for a different type of question. They excel at "how many," "what value," and "when." They struggle with "how things relate," "what depends on what," and "what happens next." For modern engineering problems—where understanding connections and causality is as important as storing measurements—graphs offer a natural extension to the relational paradigm.

3.4 Thinking in Graphs

Instead of forcing relationships into table joins, we can represent them natively.

- Each object (technician, inspection, asset) becomes a **node**.
- Each relationship (performs, involves, located at) becomes an **edge**.

For example:

- Technician → Inspection
- Inspection → Transformer
- Transformer → Location

This structure lets you ask questions like:

- Which technicians worked on the same transformer?
- What's the sequence of inspections for asset X?

3.5 Toy Example: Maintenance Log

Relational schema:

- `Technicians (ID, Name)`
- `Transformers (ID, Location)`
- `Inspections (ID, Date, TechID, TransformerID)`

Data:

Technician ID	Name
T1	Alice
T2	Bob

Transformer ID	Location
X1	North Site
X2	South Site

Inspection ID	Date	TechID	TransformerID
I1	2023-06-01	T1	X1
I2	2023-07-15	T2	X1
I3	2023-08-10	T1	X2

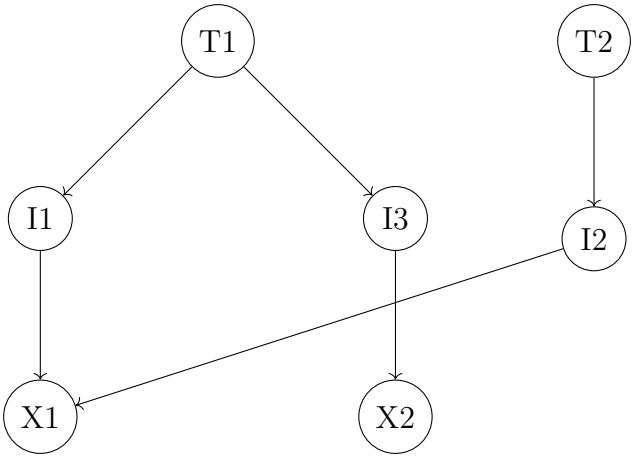

Figure 3.1: A toy example: relational data as a graph

3.6 Migrating from Relational to Graph

Transitioning from a traditional relational database to a graph structure is less about abandoning one technology for another and more about rethinking how information is modeled. In most engineering organizations, existing data are already stored in well-established SQL systems, often accumulated over years of operation. The challenge, therefore, is not starting from scratch but reshaping those data into a format that captures relationships explicitly rather than implicitly.

The migration process usually begins by identifying the key entities and relationships that define the system. In an energy context, these entities might include transformers, turbines, substations, technicians, and inspections. In the relational schema, these appear as separate tables, each with its own identifiers and attributes. The foreign keys linking these tables hint at relationships, but those connections exist only through matching IDs. The goal of the migration is to make those links explicit by turning them into edges in a graph.

For example, suppose a table called `Inspections` connects `Technicians` and `Transformers` through foreign keys. In a graph model, each technician and transformer becomes a node, and each inspection can be modeled either as a direct edge or, more appropriately, as a node of its own that links them, labeled with the inspection date or status. This approach reveals the structural patterns that were hidden behind SQL joins. What was once a complex join operation becomes a simple traversal: moving from one node to another along labeled edges. This modelling decision reflects an important principle in graph design. Whenever a relationship carries attributes of its own or may participate in additional relationships, it should be represented as a node rather than as a simple edge. This allows the relationship itself to become a first-class element of the model instead of being reduced to a mere connection. Technically, the migration can be achieved using an Extract–Transform–Load (ETL) pipeline. Data are extracted from the relational database,

transformed into a graph-compatible structure, and then loaded into a graph database such as Neo4j or GraphDB. During the transformation stage, entities are mapped to nodes, and foreign key relationships are converted into edges. Attributes that describe those relationships—such as timestamps, conditions, or quantities—become edge properties. Several open-source tools can assist in this process, but even a few lines of Python code using libraries like `pandas` and `networkx` can illustrate the concept effectively.

It is rarely necessary to migrate an entire relational database at once. A more practical strategy is to focus on the subset of data where relationships matter most. For instance, an energy company might keep numerical sensor data in a relational system but replicate only the asset relationships, maintenance histories, and technician networks in the graph layer. This hybrid approach preserves the efficiency of SQL for heavy numerical queries while unlocking the graph's ability to perform contextual reasoning and pattern discovery.

The advantages of even a partial migration are immediate. Once relationships become first-class citizens in the data model, queries that were previously complex or slow can be expressed intuitively. Engineers can follow connection chains—such as which technicians have worked on components that share suppliers or which assets are linked through fault propagation—without writing nested joins. Moreover, as the system evolves, new relationships can be added directly to the graph without altering any schema or disrupting existing operations.

Migrating from relational to graph is therefore less a technical hurdle than a conceptual one. It represents a shift from thinking about data as static records to viewing it as a living network of interactions. For engineers working in domains where connectivity defines system behavior—such as power grids, manufacturing processes, and logistics chains—this perspective offers both clarity and agility.

3.7 Hybrid Approaches

It is easy to assume that engineers must choose between a relational and a graph database, but in most real systems the two coexist. Modern data architectures rarely rely on a single paradigm. Instead, they combine the stability and transactional rigor of relational databases with the semantic richness and flexibility of graphs. The result is a hybrid environment in which both models contribute to a common analytical goal.

Relational databases continue to serve as the backbone for storing structured and high-volume numerical data. In energy engineering, these systems manage sensor readings, billing transactions, operational logs, and other well-defined records. Their strength lies in precision, reliability, and decades of optimization for queries that aggregate or filter values. Yet they lack the means to express how one entity influences or depends on another. Graph databases fill this gap. They model the connective tissue of a system—the network of relationships among assets, technicians, maintenance events, and

environmental factors. In a hybrid system, the relational layer handles the quantitative data, while the graph layer describes the qualitative context.

Historically, this division of labor has evolved naturally. The rise of Industry 4.0 and digital-twin concepts exposed the limitations of purely relational thinking. Engineers no longer analyze isolated tables; they examine interconnected systems of components. For example, a turbine's vibration reading is meaningful only when understood in relation to its gearbox, maintenance schedule, and the technician responsible for the last inspection. Storing these relationships directly in a graph allows such questions to be asked intuitively, without performing a series of complex joins across multiple tables.

A typical hybrid workflow might unfold as follows. A wind-farm operator collects gigabytes of sensor data every hour and stores them in a relational time-series database such as TimescaleDB. Periodically, a pipeline extracts selected records—metadata about turbines, maintenance events, and fault histories—and transforms them into a graph representation. In this graph, each turbine, technician, and inspection becomes a node, while the relationships among them form labeled edges. When a turbine fails, an engineer can navigate the graph to trace connections: which nearby turbines share components, which technicians have serviced both, and what environmental factors were present. This ability to traverse the data conceptually rather than numerically is what gives the hybrid model its analytical depth.

Technically, integration between relational and graph systems can be implemented in several ways. The most common is the *Extract–Transform–Load* process, which periodically synchronizes data between the two layers. More advanced configurations employ federated queries or hybrid engines—such as Apache AGE or ArangoDB—that natively support both SQL and graph traversal syntax. In some architectures, a semantic layer acts as an intermediary: it maps relational attributes to conceptual relationships. A field like `technician_id` becomes, in the semantic graph, a link labeled `hasTechnician`. This mapping enables interoperability without unnecessary duplication of data.

The benefits of hybrid architectures are manifold. Performance improves because each system handles the workload for which it was designed: relational databases process continuous numeric streams efficiently, while graph engines excel at contextual exploration. The approach also enhances scalability and flexibility. Adding a new relationship—say, linking a supplier to a specific asset—does not require altering the entire relational schema. Furthermore, hybrid systems reduce redundancy: only the essential relationships are replicated into the graph, while raw numerical data remain in the relational store. From a strategic standpoint, this design supports progressive digital transformation. Organizations can introduce a graph layer incrementally, without disrupting legacy SQL systems.

A concrete example illustrates how this works in practice. Suppose a distribution grid operator detects a voltage anomaly in a transformer. Through the relational layer, engi-

neers can immediately access sensor data to quantify the deviation. Through the graph layer, they can trace how that transformer connects to upstream substations, identify which technicians last inspected the affected components, and determine whether similar incidents occurred elsewhere. The combined insight—numerical precision from the relational side and causal understanding from the graph—enables faster and more informed decision-making.

Ultimately, hybrid approaches acknowledge that data in engineering systems are both structured and interconnected. Tables capture the measurable aspects of the world, while graphs capture its relationships. Used together, they allow engineers to move seamlessly from measurement to meaning, turning isolated datasets into actionable knowledge. This synergy represents not merely a technological choice but a conceptual shift: from storing data to understanding systems.

1. The Typical Division of Roles

Table 3.1: Typical data types and their optimal storage systems.

Type of Data	**Best Stored In**	**Reason**
Sensor measurements, time-series data	Relational DB (e.g., PostgreSQL, TimescaleDB)	Optimized for continuous numerical data and fast aggregation.
Asset metadata, maintenance records	Relational DB	Structured and transactional.
Relationships between assets, technicians, and faults	Graph DB (e.g., Neo4j, GraphDB)	Designed for complex relationships and traversals.
Ontologies, semantic links, and reasoning rules	Graph DB (RDF store)	Enables inference and context-aware queries.

2. Practical Workflow Example

   Imagine an engineer investigating why a power transformer failed unexpectedly.

   (a) Step 1 – Retrieve sensor data: The engineer queries a relational database to see recent voltage, temperature, and vibration readings.

   (b) Step 2 – Trace relationships: They use a graph database to explore connections:
   Which other transformers are connected to this one?
   Which technicians last serviced it?
   Were there similar faults in adjacent stations?

   (c) Step 3 – Combine results:

   The system blends both outputs in one dashboard, showing numerical data (from SQL) and contextual network insights (from the graph). This workflow reflects real operations in predictive maintenance, grid management, and reliability engineering.

3. Technical Integration Hybrid systems can communicate in several ways:

   - ETL Pipelines (Extract–Transform–Load): Periodically pull data from relational sources and update the graph with relevant relationships.
   - APIs or Federated Queries: Some tools (like Apache AGE, or Neo4j's SQL connectors) let you query relational and graph data in a single request.
   - Semantic Layers:

   Knowledge graphs can act as a semantic overlay, translating relational table structures into meaningful ontological relationships. For example, a SQL column $tech_id$ might be mapped to the concept "hasTechnician" in the graph ontology.

   This integration avoids duplication while giving the best of both worlds—fast relational operations and rich relationship discovery.

3.8 Summary

Relational databases remain essential in engineering environments, particularly for structured, transactional, and high-volume numeric data. However, as systems become increasingly interconnected, representing relationships explicitly becomes crucial. Graph models complement relational systems by making dependencies first-class elements rather than implicit join paths.

In practice, the most robust solution is often hybrid. Relational databases manage measurement data efficiently, while graph layers capture structural knowledge and contextual dependencies. The next chapter introduces the semantic technologies that transform a graph into a knowledge graph capable of reasoning and integration.

Chapter 4

Fundamental technologies: RDF, SPARQL, and ontologies

As we delve deeper into the world of knowledge graphs, it's essential to understand the core technologies that make these graphs not just possible but also powerful. In this chapter, we will explore three foundational components: RDF, SPARQL, and ontologies. Each of these technologies plays a crucial role in defining, querying, and enriching knowledge graphs, allowing engineers to unlock their full potential.

4.1 Understanding RDF (Resource Description Framework)

RDF is a standard model for data interchange on the web. Its primary purpose is to provide a framework for describing resources and their relationships in a structured way. At its core, RDF is based on the concept of triples, which consist of a subject, predicate, and object. This tripartite structure allows for a flexible representation of information.

Definition 4.1.1 (RDF Triple) *An RDF triple is an ordered tuple* (s, p, o) *where* s *(subject) and* p *(predicate) are IRIs or blank nodes, and* o *(object) is either an IRI, a blank node, or a literal. A collection of RDF triples forms a directed, labeled graph.*

To illustrate this structure, consider a simple example from the energy domain. The fact that transformer `T1` is installed in substation `SubstationA` can be expressed as the RDF triple

(`T1`, `locatedIn`, `SubstationA`).

This simple example anticipates the more detailed case study developed in Chapter 7.

While RDF defines how facts are represented, it does not specify what these facts mean. This motivates the introduction of ontologies. By chaining these triples together,

RDF creates a rich network of interconnected information, allowing for more sophisticated queries and insights.

4.2 The Importance of URIs in RDF

One of the defining features of RDF is the use of Uniform Resource Identifiers (URIs) to uniquely identify resources. A URI is a string of characters that acts as a reference to a specific resource, ensuring that there is no ambiguity in identifying entities. Unlike primary keys, however, URIs are globally scoped rather than restricted to a single database instance. This property enables cross-organisational data integration without additional mapping layers. For instance, instead of simply referring to "The Great Gatsby," you might use a URI like http://example.org/books/The_Great_Gatsby.

This practice is crucial for maintaining clarity and consistency within a knowledge graph. URIs enable RDF to integrate data from diverse sources seamlessly. When multiple datasets adhere to the same URI conventions, it becomes possible to link and query information across different domains. This interoperability is one of the reasons RDF has become a foundational technology for the Semantic Web.

4.3 From Data Storage to Knowledge Representation

Traditional information systems are designed to store data efficiently, often in relational tables. While this approach works well for structured records, it becomes limiting when relationships between entities become complex. Engineering systems such as supply chains, production networks, and energy infrastructures are inherently relational and interconnected.

Knowledge Graphs address this limitation by modeling facts as explicit relationships between entities. Instead of focusing on rows and columns, the focus shifts to connections and semantics. This idea has been widely adopted in enterprise knowledge management and data integration (Hogan et al., 2021a).

4.4 RDF: A Graph-Based Data Model

At the core of every Knowledge Graph lies RDF, the Resource Description Framework. RDF provides a simple but powerful data model for representing information as a graph. Instead of tables and rows, RDF uses statements called triples. Each triple consists of a subject, a predicate, and an object, and can be read as a simple sentence describing a fact (Klyne & Carroll, 2004).

For example, the statement that a company produces a product can be expressed as a triple connecting the company entity to the product entity via the predicate *produces*. When additional statements are added—such as the materials used by the product or the location of the company—the graph grows naturally. Over time, many such statements form a dense network of interconnected knowledge. This representation closely mirrors real engineering systems, where components, processes, and organizations are linked through multiple relationships.

A crucial feature of RDF is the use of globally unique identifiers, known as Uniform Resource Identifiers (URIs). URIs ensure that every entity in the graph has an unambiguous identity. This is particularly important when integrating data from different systems, such as enterprise databases, external APIs, or public datasets (Heath & Bizer, 2011). For engineers, URIs play a role similar to globally unique identifiers or primary keys, but with the additional advantage that they support linking across organizational and system boundaries.

RDF is also highly flexible. Unlike relational databases, it does not require a rigid schema defined in advance. New facts can be added simply by introducing new triples, without modifying existing data structures. This makes RDF especially suitable for evolving systems, such as digital twins, long-lived industrial platforms, or enterprise knowledge management systems, where requirements change over time (Allemang & Hendler, 2011).

4.5 Ontologies and Semantic Modeling

While RDF provides the basic framework for representing relationships, ontologies add a layer of semantic richness to knowledge graphs. An ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. It defines the vocabulary and structure needed to describe the data in a meaningful way.For example, in the context of a book ontology, you might define classes such as Book, Author, and Publisher, along with properties like written_by, published_by, and has_genre. This structured approach enables you to model complex relationships and hierarchies effectively. By using ontologies, you can ensure consistency in how data is represented across different datasets. This consistency is vital when integrating data from various sources, as it helps avoid ambiguity and confusion. Moreover, ontologies enable reasoning and inference, allowing systems to derive new knowledge from existing data. Consider a scenario where you have an ontology representing literary works. If your knowledge graph includes the fact that "The Great Gatsby" is a type of Book, and you also have a rule stating that all Books have an associated Author, you can infer that F. Scott Fitzgerald is indeed the author of "The Great Gatsby" without explicitly stating it in your RDF triples. This reasoning capability enhances the power of knowledge graphs, enabling more intelligent data analysis.

While RDF defines how data is represented, it does not specify what the data means. Ontologies provide this semantic layer by defining the concepts of a domain, the relationships between them, and constraints on their use. An ontology can be understood as a formal and shared vocabulary that captures domain knowledge in a machine-readable form (Gruber, 1993).

In engineering and business domains, ontologies often define concepts such as companies, products, suppliers, machines, or locations. They also define relationships such as *produces*, *supplies*, or *locatedIn*. By making these definitions explicit, ontologies ensure that data is interpreted consistently across systems and teams. This is especially important in interdisciplinary environments, where different stakeholders may use the same terms with slightly different meanings.

Ontologies are commonly expressed using the Web Ontology Language OWL, which extends RDF with logical constructs (McGuinness & van Harmelen, 2004). These constructs allow engineers to define class hierarchies, domain and range restrictions, and equivalence relationships. As a result, ontologies support automated reasoning, enabling systems to infer new facts from existing ones.

Reasoning plays an important role in maintaining data quality and consistency. For instance, if an ontology specifies that every manufacturer is a type of company, and a particular entity is classified as a manufacturer, then it can automatically be inferred that this entity is also a company. Such inferences reduce redundancy and help detect modeling errors (Baader et al., 2007). In large-scale Knowledge Graphs, this capability becomes essential.

4.6 Querying Knowledge Graphs with SPARQL

Once we have structured our data using RDF, the next step is querying that data effectively. This is where SPARQL (SPARQL Protocol and RDF Query Language) comes into play. SPARQL is a powerful query language designed specifically for retrieving and manipulating data stored in RDF format. SPARQL queries resemble SQL queries in their structure, but they are tailored to work with the RDF data model. A typical SPARQL query consists of a SELECT statement, which specifies the variables to retrieve, and a WHERE clause, which outlines the patterns the query should match within the RDF graph. Once data has been represented using RDF and structured with an ontology, it must be queried effectively. SPARQL is the standard query language for RDF-based Knowledge Graphs (Prud'hommeaux & Seaborne, 2008). While SQL queries tables, SPARQL queries graph patterns.

A SPARQL query describes a set of relationships that should exist between entities. The query engine searches the Knowledge Graph for all subgraphs that match this pattern. This makes SPARQL particularly powerful for queries involving multiple re-

lationships or indirect connections. Questions such as tracing dependencies in a supply chain or identifying risks propagated through several layers of a network can be expressed naturally as graph patterns.

For engineers familiar with SQL, SPARQL represents a conceptual shift rather than a technical barrier. Instead of thinking in terms of joins, one thinks in terms of paths in a graph. This aligns well with how engineers reason about systems composed of interconnected components.

4.7 An Integrated Engineering Perspective

RDF, ontologies, and SPARQL are not isolated technologies. Together, they form a semantic layer that complements existing data infrastructures. RDF provides a flexible graph-based representation, ontologies define meaning and constraints, and SPARQL enables expressive querying. This combination allows engineers to integrate heterogeneous data sources, ensure semantic consistency, and extract insights that are difficult to obtain with traditional data models (Paulheim, 2016).

In practice, this approach is increasingly used in enterprise analytics, sustainability reporting, digital twins, and decision-support systems. Understanding these technologies is therefore a crucial step toward designing and deploying effective Knowledge Graph solutions. The next chapter builds on this foundation by focusing on the practical design of a business Knowledge Graph.

4.8 Real-world applications

In practical applications, RDF, SPARQL, and ontologies have proven their value across various domains. In the biomedical field, for instance, researchers use knowledge graphs to represent complex relationships between genes, diseases, and treatments. By integrating data from clinical trials, research studies, and patient records, healthcare professionals can gain insights into how different factors influence health outcomes.In the realm of e-commerce, companies leverage knowledge graphs to enhance product recommendations. By mapping relationships between products, customers, and their preferences, businesses can provide personalized suggestions that drive sales and improve customer satisfaction.In summary, RDF, SPARQL, and ontologies are the cornerstones of knowledge graph technology. Together, they enable engineers to build, query, and enrich knowledge graphs that provide meaningful insights and drive decision-making across various industries. Understanding these fundamental technologies is essential for anyone looking to harness the power of knowledge graphs effectively.

4.9 RDF and Property Graph Models

It is important to note that RDF-based knowledge graphs are not the only graph technology used in industrial practice. Property graph systems, such as those implemented in platforms like Neo4j, represent data as nodes and edges with attached key–value properties.

While RDF emphasizes semantic interoperability, global identifiers, and formal reasoning, property graph models often prioritize performance and application-oriented development.

In engineering environments, both paradigms coexist. RDF is typically preferred when interoperability, ontology-driven reasoning, and standardization are required. Property graphs are often chosen for operational analytics and system-specific implementations.

Chapter 5

Design a Business Knowledge Graph

Designing a business Knowledge Graph is an engineering task that goes far beyond selecting technologies or defining data schemas. It is a process that requires a deep understanding of how an organization creates value, how decisions are made, and how information flows across departments, systems, and time. Unlike traditional databases, which are often designed to support predefined transactions or reports, a Knowledge Graph is designed to support understanding, exploration, and reasoning. For this reason, its design must begin with business meaning rather than with data structures.

A business Knowledge Graph should be seen as a shared semantic backbone of the organization. It does not replace existing systems such as ERP, CRM, or data warehouses. Instead, it connects them by making explicit the entities they describe and the relationships that link them. When designed correctly, a Knowledge Graph allows engineers, analysts, and decision-makers to ask questions that cut across organizational silos, temporal boundaries, and levels of abstraction. Achieving this requires careful design choices, many of which are not purely technical but conceptual and organizational.

5.1 Design driven by business purpose

A successful knowledge graph design begins with a clear articulation of business purpose. General ambitions such as "improving analytics" or "using Artificial Intelligence (AI)" are not sufficient starting points. What is required is a concrete understanding of the decisions the organisation needs to support. These decisions often concern operational risk, asset reliability, regulatory compliance, sustainability targets, resource allocation, or long-term investment planning.

Unlike traditional database design, which typically focuses on storing transactions or generating predefined reports, knowledge graph design focuses on capturing the structure of understanding within the organisation. It aims to represent how entities are related in a way that reflects how experts reason about the system.

To make this discussion concrete, consider a regional electricity distribution operator responsible for substations, transformers, and maintenance teams. The organisation seeks to reduce unexpected equipment failures and improve maintenance planning. The relevant business purpose is therefore not "collecting more data," but enabling informed decisions about which assets require attention and why.

At this stage, engineers must resist the temptation to begin modelling existing databases. Legacy systems such as Enterprise Resource Planning (ERP), Computerized Maintenance Management System (CMMS), or data warehouses reflect historical constraints and local optimisations rather than a unified conceptual view of the business. If a knowledge graph simply mirrors them, it will inherit their fragmentation. Instead, the design process should begin with structured interviews and workshops involving domain experts, uncovering how decisions are actually made and which relationships are implicitly considered.

This phase often reveals inconsistencies in terminology and informal knowledge that has never been formalised. Rather than being an obstacle, this discovery process is one of the primary values of knowledge graph design: it forces conceptual clarity before technical implementation.

5.2 From Business Questions to Knowledge Structures

Once the business purpose has been clarified, it must be translated into explicit knowledge structures. Business questions are typically expressed in natural language and often contain ambiguity, assumptions, and implicit context. The role of the engineer is to unpack these questions and identify the entities, relationships, and constraints they presuppose.

Consider the electricity distribution operator introduced earlier. Suppose management asks:

"Which transformers present elevated operational risk due to repeated faults?"

At first glance, this appears to be a simple analytical question. However, it implicitly involves several conceptual elements: transformers, fault events, timestamps, maintenance actions, and a notion of risk classification.

To represent this question in a knowledge graph, we must identify:

- Core entities: *Transformer*, *FaultEvent*, *MaintenanceAction*, *RiskLevel*.
- Core relationships: *experiencesFault*, *repairedBy*, *classifiedAsRisk*.
- Relevant constraints: temporal ordering, frequency thresholds, and possibly regulatory limits.

In other words, the question is not answered by a single data field but by traversing a structured network of relationships.

Definition 5.2.1 *A business knowledge structure is a triple* $\mathcal{K} = (C, R, \Sigma)$*, where* C *is a set of conceptual classes,* R *is a set of relationships between classes, and* Σ *is a set of business constraints governing valid configurations of these relationships.*

This abstraction is intentionally simple. It emphasizes that knowledge graph design is not merely about storing data, but about explicitly defining how concepts are connected and under which conditions those connections are meaningful.

By systematically translating business questions into entities, relationships, and constraints, the organisation moves from informal reasoning to a shared, formal representation of knowledge.

5.3 Identifying Core Entities and Their Boundaries

Identifying core entities is not only about listing important objects in the organisation. It also requires defining their conceptual boundaries. A boundary determines where one entity ends and another begins, and it clarifies what belongs inside a concept and what does not.

In the electricity distribution example, a transformer is clearly a core entity. However, we must ask: does a transformer include its sensor measurements? Does it include its historical configuration? Does it include its maintenance history?

If measurements are treated merely as attributes of a transformer, then their temporal evolution may be difficult to analyse. If they are modelled as separate entities, connected through a relationship such as *hasMeasurement*, then each measurement can carry its own timestamp, unit, and quality indicator. The modelling decision depends on whether those measurements are central to decision-making.

Boundaries also clarify responsibility and identity. A maintenance team may change members over time. Should the team be represented as a stable organisational entity, or should each assignment be modelled separately? These decisions affect how accountability, workload, and performance indicators can later be analysed.

Poorly defined boundaries lead to ambiguity and inconsistent data interpretation. Well-defined boundaries, on the other hand, make the knowledge graph more stable over time and easier to maintain. They ensure that each entity represents a coherent unit of meaning within the organisation.

5.4 Relationships as First-Class Design Elements

If entities form the backbone of a knowledge graph, relationships form its connective tissue. A knowledge graph derives its analytical power not merely from the existence of entities, but from the explicit representation of how those entities interact.

In the electricity distribution scenario, relationships such as *locatedIn*, *experiencesFault*, *maintainedBy*, and *connectedTo* are not decorative additions. They encode causal structure, organisational responsibility, and physical topology. Without them, the graph would collapse into a collection of isolated records.

Designing core relationships requires clarity about direction, cardinality, and semantic meaning. For example, should the relationship between a transformer and a fault be expressed as *experiencesFault* or *causedBy*? The choice reflects how the organisation interprets operational events. Similarly, the relationship between a maintenance action and a technician may be *performedBy*, implying responsibility, or *assignedTo*, implying scheduling.

Direction also carries meaning. In a power network, *connectedTo* may be symmetric, whereas *suppliesPowerTo* is inherently directional. These modelling choices affect how queries propagate through the graph and how dependencies are analysed.

Relationships may themselves require attributes. For example, the connection between a transformer and a substation may have a commissioning date, capacity rating, or operational status. Such cases motivate modelling the connection as an entity in its own right, sometimes called reification or relationship modelling.

A carefully designed set of core relationships ensures that the knowledge graph captures not only static structure, but also the dynamic interactions that matter for decision-making.

5.5 Data Sources and Semantic Integration

A business knowledge graph does not operate in isolation. It integrates information originating from multiple heterogeneous systems, each designed for a specific operational purpose. In an electricity distribution environment, relevant sources may include Supervisory Control and Data Acquisition (SCADA) systems for real-time monitoring, CMMS platforms for maintenance records, ERP systems for asset inventories and procurement, and sensor streams from Internet of Things (IoT) devices.

These systems differ not only in format, but also in conceptual perspective. A SCADA system may represent a transformer through operational identifiers and telemetry channels, whereas an ERP system may represent the same transformer as a financial asset with cost and depreciation attributes. A maintenance system, in turn, may refer to it through work orders and inspection logs.

Semantic integration consists of reconciling these different representations into a unified conceptual view. This process is not merely technical extraction and loading. It requires identifying when two identifiers refer to the same real-world entity, resolving inconsistencies in naming conventions, and aligning terminologies that evolved independently.

For example, one system may record “TX-01” while another uses “Transformer 1”. The knowledge graph must establish a stable identity for the underlying asset and connect all related records to that identity. This typically involves entity resolution mechanisms and carefully defined mappings.

Semantic integration also requires addressing differences in granularity. Real-time telemetry may produce thousands of measurements per day, while maintenance logs are event-based. The knowledge graph must determine which data are stored as structured entities, which remain external but linked, and which are aggregated into higher-level indicators.

A successful integration strategy does not attempt to copy entire operational databases into the graph. Instead, it selectively incorporates the information necessary to support the defined business questions, while preserving traceability to the original systems.

5.6 Data Sources and Semantic Integration

A business Knowledge Graph almost always integrates data from multiple sources. These sources differ in structure, quality, update frequency, and ownership. Designing the Knowledge Graph therefore requires a clear strategy for semantic integration.

Semantic integration is not merely about mapping fields between systems. It involves aligning meanings. For example, the concept of a “supplier” may differ between procurement systems and financial systems. The Knowledge Graph must reconcile these differences and provide a coherent representation.

Engineers must also decide which sources are authoritative for which types of information. This decision affects trust, conflict resolution, and governance. In many cases, the Knowledge Graph should retain links to original sources rather than attempting to replace them.

5.7 Granularity and Abstraction Levels

Granularity refers to the level of detail at which information is represented in the knowledge graph. Every modelling decision fixes a resolution. If the model is too coarse, important relationships may be hidden. If it is too fine-grained, the graph may become difficult to maintain and computationally expensive to query.

In the electricity distribution example, a transformer may be modelled simply as an asset entity with basic attributes such as capacity and installation year. However, one could also represent individual components, sensor measurements, inspection records, and cooling subsystems. The appropriate level of detail depends on the decisions the organisation intends to support.

For financial planning, modelling at the asset level may be sufficient. For predictive maintenance or reliability analysis, event-level representation becomes necessary. A fault counter attached to a transformer provides limited insight, whereas modelling each fault as a separate entity enables analysis of frequency, timing, and correlation with environmental conditions.

Abstraction works in the opposite direction. It groups entities into higher-level categories. Transformers may be classified by voltage class, geographic region, or operational risk category. These abstraction layers allow strategic questions to be answered without navigating the full operational complexity of the graph.

A robust knowledge graph design often supports multiple levels simultaneously. Operational detail coexists with aggregated representations such as performance indicators or risk scores. The critical design principle is traceability: higher-level nodes must remain connected to the underlying operational data from which they are derived.

Choosing granularity is therefore a balancing act between expressiveness, clarity, and performance. It is a design decision that directly shapes the analytical capabilities of the organisation.

5.8 Designing for Change and Longevity

A business knowledge graph should be designed as a long-lived infrastructure rather than as a short-term project. Organisations evolve, regulatory frameworks change, asset portfolios expand, and operational processes are revised. If the knowledge graph cannot accommodate such changes, its usefulness will quickly decline.

In the electricity distribution example, new types of sensors may be installed, additional substations may be integrated, or regulatory requirements may introduce new reporting obligations. The model must allow these elements to be incorporated without requiring a complete redesign.

Designing for change means favouring extensibility over rigidity. New entity types should be introducible without disrupting existing structures. Relationships should be defined in a way that does not assume fixed hierarchies when the real-world organisation is dynamic.

Ontology versioning is particularly important. As the conceptual model evolves, changes should be documented explicitly. Deprecated classes or properties should not disappear silently but be marked and migrated carefully. This ensures traceability and prevents semantic drift.

Another important aspect is separation between conceptual design and implementation technology. The conceptual model should remain stable even if storage engines, query platforms, or integration tools change. This separation protects the organisation from vendor lock-in and preserves long-term flexibility.

Designing for longevity requires accepting that the first version of a knowledge graph will not be perfect. The goal is not to anticipate every future requirement, but to create a structure that can evolve gracefully.

5.9 Validation, Governance, and Responsibility

A knowledge graph that is not validated and governed will quickly lose credibility. Unlike isolated analytical tools, a business knowledge graph becomes a shared reference point across departments. Decisions may depend on it. For this reason, its correctness and consistency must be actively maintained.

Validation operates at multiple levels. At the structural level, the graph must respect the constraints defined in the ontology. Cardinality rules, domain and range restrictions, and logical dependencies must be checked regularly. Technologies such as SHACL, introduced earlier, provide a formal mechanism for expressing and enforcing such constraints.

At the data level, validation concerns completeness and plausibility. Are required relationships present? Do timestamps follow logical order? Are asset identifiers unique and stable? In an electricity distribution context, inconsistencies between maintenance records and operational logs may indicate modelling errors or integration failures.

Governance refers to the organisational processes that ensure responsibility for the graph. A knowledge graph should not be an unowned technical artifact. Clear roles must be defined: who is responsible for ontology evolution, who approves new entity types, who validates mappings from external systems, and who resolves semantic conflicts?

Responsibility also extends to transparency. If a risk score or performance indicator is derived from graph data, its derivation should be traceable. Users must be able to understand which entities and relationships contributed to a conclusion. This is particularly important when the knowledge graph supports regulatory reporting or safety-related decisions.

Without governance, a knowledge graph risks becoming another inconsistent data repository. With governance, it becomes institutional memory: a stable, evolving representation of how the organisation understands itself.

5.10 From Design to Implementation Readiness

A conceptual design is only valuable if it can be implemented in a controlled and sustainable manner. Once entities, relationships, constraints, and abstraction levels have been defined, the organisation must prepare the transition from modelling to operational deployment.

Implementation readiness does not mean that every possible data source must be connected immediately. Instead, it involves prioritising use cases, validating data mappings,

and testing the graph structure against real queries. Early pilot deployments are often useful for identifying unexpected modelling gaps or integration challenges.

In the electricity distribution example, implementation readiness might begin with a limited deployment focusing on transformer fault analysis. Core entities and relationships are instantiated using real operational data, and typical business questions are executed as test queries. This process verifies whether the conceptual model supports the intended decisions and reveals whether adjustments are required.

Performance considerations also enter at this stage. Query complexity, indexing strategies, and update frequency must be evaluated in light of expected workloads. Although the conceptual model should remain independent of specific technologies, its structure must be realistic with respect to computational constraints.

Equally important is documentation. Clear definitions of classes, relationships, and constraints ensure that future engineers can extend the graph without reinterpreting its original purpose. A knowledge graph that lacks documentation may function technically but will gradually lose semantic coherence.

Implementation readiness therefore marks the transition from design as an intellectual exercise to design as operational infrastructure. When this phase is approached systematically, the knowledge graph becomes a reliable foundation for querying and analysis, which is the focus of the next chapter.

5.11 Conclusion

Designing a business Knowledge Graph is a demanding but rewarding engineering task. It requires technical skill, domain understanding, and effective communication. When approached thoughtfully, it results in a system that supports insight, learning, and informed decision-making across the organization. The next chapter builds on this foundation by focusing on how such Knowledge Graphs can be queried to generate concrete business insights.

Chapter 6

Querrying Knowledge Graphs for business insights

A knowledge graph that no one queries is just an expensive diagram. This chapter is about how engineers and analysts ask questions of a knowledge graph and how those questions become quantitative answers that support decisions.

The previous chapter described how a business knowledge graph is designed. Design alone, however, does not create value. Value emerges when the graph is interrogated and used to support decisions.

6.1 Thinking in Paths Instead of Joins

Engineers with experience in Structured Query Language (SQL) often attempt to transfer their existing mental model directly to SPARQL Protocol and RDF Query Language (SPARQL). This typically leads to unnecessarily complex queries and frustration. The essential difference is structural: SQL operates on tables connected by joins, whereas SPARQL operates on graph patterns that describe paths.

In SQL, a complex business question may require a sequence of explicit joins between tables. As the number of joins increases, queries become difficult to read and maintain. In a knowledge graph, the same question can often be expressed as a path that follows relationships between entities.

Consider the question: "Which raw materials indirectly affect our final products?" In a relational system, this would require joining product, bill-of-materials, supplier, and material tables. In a knowledge graph, the question becomes a path from products to materials through intermediate relationships. The query describes the path, and the graph engine identifies all matching structures.

Path-based thinking aligns naturally with how engineers reason about systems. Processes, flows, and dependencies are typically understood as chains of interactions rather

than as isolated records.

6.2 Basic Query Patterns

Although knowledge graph queries can be highly expressive, most business questions rely on a small number of recurring structural patterns. Recognising these patterns helps engineers design queries systematically rather than improvising ad hoc solutions.

Entity retrieval. The objective is to identify all entities of a given type that satisfy specific conditions. For example, retrieving all suppliers located outside the European Union involves specifying a type constraint and a geographic restriction. The graph pattern describes the relevant relationships, and filtering conditions refine the result set.

Relationship exploration. In many business contexts, the goal is not merely to list entities but to understand how they are connected. Relationship exploration queries map paths between entities. For instance, identifying which suppliers contribute materials to specific products requires traversing the path from product to material to supplier. Such queries reveal dependency structures that are difficult to express cleanly in tabular systems.

Aggregation and comparison. Business decisions often depend on quantitative summaries. Knowledge graphs support aggregation functions such as `COUNT`, `AVG`, or `SUM`. For example, calculating the average Carbon Dioxide (CO2) intensity of suppliers contributing to a product portfolio provides a basis for sustainability assessment. Aggregation queries combine structural traversal with numerical evaluation.

These patterns illustrate that graph querying is not conceptually complex. Most analytical questions can be reduced to combinations of entity retrieval, path traversal, and aggregation. The expressive power of SPARQL lies in allowing these patterns to be composed while remaining readable.

6.3 Using Semantics During Querying

One of the defining features of a knowledge graph is that queries can exploit not only explicitly stored data, but also the semantic structure defined in the ontology. This distinguishes graph querying from conventional database querying.

In a relational system, a query returns exactly the rows that match the specified conditions. In a semantically enriched knowledge graph, a query may return additional results that follow logically from class hierarchies or property definitions.

Consider the electricity distribution example. Suppose the ontology defines that every `HighVoltageTransformer` is a subclass of `Transformer`. A query requesting all entities of type `Transformer` should also return instances of `HighVoltageTransformer`, even if they are not explicitly declared as `Transformer`. This behaviour follows from RDFS subclass semantics.

Similarly, property hierarchies influence query results. If the ontology specifies that `suppliesPowerTo` is a subproperty of `connectedTo`, a query for connected assets may automatically include relationships originally asserted as `suppliesPowerTo`.

Semantic querying therefore relies on inference. Depending on the system, inference may be performed at query time or precomputed during data loading. In both cases, the result is that queries operate on the logical closure of the graph rather than on isolated triples.

For engineers, the practical implication is important: queries should be written at the appropriate level of abstraction. Instead of enumerating every specific asset subtype, it is often better to query a more general class and allow the ontology to expand the result set automatically.

This semantic layer reduces query complexity and increases robustness. When new subclasses or specialised relationships are introduced, existing queries continue to function correctly, provided they rely on the ontology rather than on hard-coded assumptions.

6.4 Querying at Different Levels of Abstraction

A well-designed knowledge graph supports queries at multiple levels of abstraction. The same underlying structure can answer highly operational questions as well as strategic, aggregated ones. The difference lies not in the data source, but in how the graph is traversed and how results are grouped.

At an operational level, a query may focus on specific assets or events. For example, an engineer might ask:

"Which transformers experienced more than three faults in the past six months?"

Such a query operates at event-level granularity and relies on explicit relationships between transformers and fault events.

At a strategic level, management might instead ask:

"Which regions exhibit elevated operational risk?"

This question abstracts away from individual assets and focuses on aggregated indicators derived from multiple entities. The graph structure allows fault events to be linked to transformers, transformers to substations, and substations to regions. An aggregation step then produces a regional risk metric.

The ability to move between levels is one of the defining strengths of graph-based systems. Because relationships are explicit, it is possible to trace aggregated indicators

back to the operational entities that generated them. This traceability is often difficult to maintain in systems where data are stored in disconnected tables or precomputed summaries.

Querying across abstraction levels therefore depends on two design principles introduced earlier: appropriate granularity and explicit relationships between conceptual layers. If these are well defined, the graph becomes a flexible analytical instrument rather than a static repository.

6.5 Dealing with Uncertainty and Incompleteness

Real-world engineering systems are rarely fully documented. Sensor failures, delayed reports, legacy systems, and manual data entry all contribute to incomplete or inconsistent data. A knowledge graph does not eliminate uncertainty, but it makes it explicit and manageable.

One important conceptual difference between relational databases and semantic knowledge graphs is the so-called *open-world assumption*. In a relational system, if a fact is not stored, it is typically treated as false. In contrast, semantic systems assume that missing information may simply be unknown. The absence of a relationship does not necessarily imply its negation.

For example, if a transformer is not linked to a recorded maintenance event, this does not automatically mean that no maintenance occurred. It may indicate that the data have not yet been integrated. Engineers must therefore interpret query results carefully, especially when absence of evidence might be mistaken for evidence of absence.

Knowledge graphs also allow uncertainty to be modelled explicitly. Confidence values, timestamps, or data provenance information can be attached to relationships. For instance, a connection between a supplier and a material may carry a reliability score derived from audit results. Queries can then incorporate these attributes to filter or rank results.

Temporal aspects are equally important. Many engineering facts are valid only during specific periods. A transformer may be connected to one substation today and to another after a reconfiguration. Including temporal qualifiers in the graph enables queries such as:

"Which assets were connected to this region in 2023?"

Handling incompleteness therefore requires both modelling discipline and cautious interpretation. A well-designed knowledge graph does not pretend that uncertainty does not exist; instead, it provides a structure in which uncertainty can be documented, queried, and gradually reduced as more data become available.

6.6 Performance and Scalability Considerations

As knowledge graphs grow, performance becomes a practical concern. Industrial graphs may contain millions of triples describing assets, events, suppliers, contracts, and operational data. Query design therefore influences response time and system scalability.

One important factor is path length. Queries that traverse many intermediate nodes may generate large intermediate result sets. While graph engines are optimized for relationship traversal, complex patterns involving multiple joins, filters, and aggregations can still become computationally expensive. Engineers should therefore aim for queries that reflect the actual business question without introducing unnecessary traversal steps.

Indexing strategies also play a central role. Most graph databases maintain indexes on subjects, predicates, and objects. Well-designed ontologies support efficient indexing by avoiding overly generic relationships. For example, using a broad property such as `relatedTo` everywhere reduces selectivity and slows down queries.

Reasoning introduces another performance dimension. If inference is computed at query time, response times may increase depending on the complexity of the ontology. Alternatively, systems may materialize inferred triples during data loading, trading storage space for faster query evaluation. The appropriate strategy depends on system size and update frequency.

Scalability is not only a technical issue but also an organisational one. As more departments contribute data, the graph grows in volume and complexity. Governance policies, naming conventions, and schema discipline become essential to prevent uncontrolled expansion.

In practice, performance optimisation in knowledge graphs follows the same engineering principle as system design: start simple, measure behaviour, and refine iteratively. Most business queries remain efficient when modelling decisions are consistent and path patterns are kept meaningful.

6.7 Interpreting Query Results as Business Insight

Executing a query is only the beginning. The true value of a knowledge graph lies not in producing lists of entities, but in transforming structured results into actionable understanding.

A query may return a set of transformers with repeated faults, a group of suppliers contributing high CO2 emissions, or a chain of dependencies linking a raw material to several critical products. These results are structural descriptions. To become business insight, they must be interpreted within an operational or strategic context.

For example, identifying suppliers with above-average emission intensity does not automatically imply termination of contracts. It may instead trigger further investigation,

renegotiation, or risk diversification strategies. The graph provides visibility; management determines action.

Similarly, tracing dependencies across multiple layers of the supply chain reveals systemic risk. If a single material affects several high-revenue products, the organisation may decide to diversify sourcing or increase inventory buffers. The graph exposes concentration risk that might otherwise remain hidden.

Knowledge graphs therefore function as analytical instruments. They do not replace human judgement, but they structure information so that judgement can be applied more effectively. Because the underlying relationships remain visible, decisions can be justified and audited.

An important characteristic of graph-based insight is traceability. Aggregated indicators can always be decomposed back to the entities and relationships that generated them. This transparency is essential in regulated sectors, where sustainability reporting, operational safety, or compliance requirements demand explainable results.

Ultimately, querying a knowledge graph is not a technical exercise but part of a broader decision-support cycle. Questions evolve, hypotheses are tested, patterns are discovered, and models are refined. The graph becomes a living representation of organisational knowledge rather than a static storage system.

6.8 Conclusion

This chapter examined how knowledge graphs are queried to generate business-relevant insight. The discussion moved from the structural difference between joins and paths to practical query patterns used in engineering and management contexts.

We explored how semantics enrich query results, how abstraction levels allow movement from operational detail to strategic indicators, and how uncertainty and performance considerations influence practical deployment. Most importantly, we emphasised that querying is not merely a technical operation but part of a broader decision-support process.

Knowledge graphs reveal dependencies, concentration risks, and structural relationships that remain hidden in tabular systems. When combined with careful interpretation, they become powerful analytical tools for engineering organisations operating in complex environments.

Chapter 7

Case Study: An Energy Knowledge Graph

7.1 Case Description: A Regional Electricity Network

Consider a regional electricity distribution operator responsible for maintaining transformers, substations, and associated equipment. The operator collects data from multiple sources: SCADA systems, maintenance logs, inspection reports, supplier records, and environmental performance indicators.

Each transformer is located in a specific region, connected to substations, monitored by sensors, and serviced by technicians. Fault events are recorded together with timestamps and severity levels. In addition, management seeks to evaluate operational risk and sustainability performance across regions.

The challenge is not the absence of data, but fragmentation. Operational data reside in control systems, maintenance data in a CMMS, supplier information in an ERP, and sustainability metrics in reporting tools. The objective of this case study is to integrate these sources into a unified Knowledge Graph (KG) that supports both operational and strategic analysis.

7.2 Conceptual Model and Ontology Design

The conceptual model identifies the core entities:

- Transformer
- Substation
- Region

- FaultEvent
- Technician
- MaintenanceAction

Key relationships include:

- `locatedIn(Transformer, Region)`
- `connectedTo(Transformer, Substation)`
- `experiencedFault(Transformer, FaultEvent)`
- `performed(Technician, MaintenanceAction)`
- `resolvedBy(MaintenanceAction, FaultEvent)`

The ontology defines hierarchical relationships. For example,

- `HighVoltageTransformer` $\subseteq$ `Transformer`
- `PreventiveMaintenance` $\subseteq$ `MaintenanceAction`

These class hierarchies allow semantic querying at different levels of abstraction without modifying query structure.

7.3 RDF Representation

The conceptual model is encoded in Resource Description Framework (RDF). A simplified Turtle fragment is shown below.

```
@prefix ex: <http://example.org/energy#> .

ex:T1 a ex:Transformer ;
      ex:locatedIn ex:RegionA ;
      ex:experiencedFault ex:F1 .

ex:F1 a ex:FaultEvent ;
      ex:severity "High" ;
      ex:occurredOn "2024-03-15" .

ex:Tech1 a ex:Technician ;
         ex:performed ex:M1 .
```

```
ex:M1 a ex:MaintenanceAction ;
      ex:resolved ex:F1 .
```

Each triple explicitly represents a relationship, making dependencies transparent and machine-interpretable.

7.4 Graph Structure and Mathematical View

Formally, the knowledge graph can be represented as a graph $G = (V, E)$, where V is the set of entities and E the set of directed relationships.

If the graph is simplified to focus on transformer connectivity, we may consider an undirected abstraction for network analysis.

Let A denote the adjacency matrix of G, defined as

$$A_{ij} = \begin{cases} 1 & \text{if an edge exists between } v_i \text{ and } v_j, \\ 0 & \text{otherwise.} \end{cases}$$

Centrality measures can then be computed. For instance, degree centrality of transformer $T1$ corresponds to the number of directly connected entities.

Clustering coefficients provide insight into local redundancy. If two neighbouring transformers share a common substation, the network exhibits structural resilience. Such structural indicators help identify critical nodes whose failure would significantly affect the network.

7.5 Querying and Analytical Use

Once integrated, the knowledge graph supports both operational and strategic queries.

An operational query may ask:

"Which transformers experienced more than two high-severity faults in the past year?"

A strategic query may instead aggregate risk at regional level:

```
SELECT ?region (COUNT(?fault) AS ?faultCount)
WHERE {
  ?t a ex:Transformer ;
     ex:locatedIn ?region ;
     ex:experiencedFault ?fault .
}
GROUP BY ?region
```

Such queries reveal concentration of operational risk. If a single region exhibits disproportionately high fault frequency, management may allocate additional resources or review maintenance strategies.

The graph also supports dependency tracing. If a specific supplier provides components to multiple transformers within the same region, supply disruption may generate cascading operational effects.

7.6 Discussion and Lessons Learned

This case study illustrates how a knowledge graph unifies heterogeneous data into a coherent analytical structure. The graph makes dependencies explicit, supports semantic abstraction, and enables traceable decision support.

However, several challenges remain. Data quality must be continuously monitored, ontology evolution must be managed carefully, and performance must be evaluated as the graph grows.

Despite these challenges, the case demonstrates that knowledge graphs are not theoretical constructs. They provide tangible value in complex engineering environments where interconnected systems, regulatory constraints, and sustainability targets intersect.

Chapter 8

Challenges and Trends

8.1 Data Quality and Governance

The effectiveness of a knowledge graph depends fundamentally on the quality of the underlying data. In industrial environments, data often originate from heterogeneous systems, including legacy databases, SCADA platforms, maintenance logs, and external reporting tools. Inconsistencies in naming conventions, missing attributes, or incomplete integration pipelines may propagate into the graph.

A knowledge graph makes structural relationships explicit, but it does not automatically correct inaccurate or outdated information. Governance frameworks are therefore essential. Clear ownership of entities, version control of ontologies, and validation mechanisms must be defined at organisational level.

Without governance, a knowledge graph risks becoming a large but unreliable network of loosely connected facts. With proper governance, it becomes a trusted analytical infrastructure.

8.2 Ontology Evolution and Maintenance

Engineering systems evolve continuously. New asset types are introduced, regulatory requirements change, and operational practices are refined. The ontology must adapt accordingly.

Uncontrolled modifications, however, may break existing queries or introduce semantic inconsistencies. Versioning strategies and change documentation are therefore critical.

A practical principle is to design ontologies incrementally. Core concepts should remain stable, while extensions capture domain-specific innovations. This balance between stability and adaptability determines the long-term sustainability of the knowledge graph.

8.3 Performance and Industrial Scale

As knowledge graphs grow to millions or billions of triples, performance becomes a strategic consideration. Complex path queries, large aggregations, and inference mechanisms may introduce latency.

Industrial deployments often rely on distributed graph stores or hybrid architectures combining relational databases with graph components. Optimisation strategies include selective materialisation of inferred triples, query rewriting, and careful schema design.

Scalability therefore depends not only on hardware capacity but also on modelling discipline and realistic expectations regarding query complexity.

8.4 Integration with Machine Learning and Large Language Models

Recent developments in Machine Learning (ML) and large language models have renewed interest in knowledge graphs. Graph-based features can enhance predictive models by incorporating relational context, while embeddings derived from knowledge graphs support recommendation and anomaly detection tasks.

At the same time, large language models increasingly use knowledge graphs as structured backbones for retrieval-augmented generation. In such systems, the graph provides verifiable facts, while the language model generates natural-language explanations.

For engineering organisations, this integration suggests a complementary relationship rather than replacement. Knowledge graphs provide structured, explainable representations of domain knowledge. Machine learning models exploit patterns within that structure.

8.5 Standardisation and Validation

As knowledge graph adoption grows, standardisation becomes increasingly important. Technologies such as RDF, Web Ontology Language (OWL), and SPARQL provide interoperability across systems. In addition, constraint languages such as SHACL (Shapes Constraint Language) allow validation of graph structure.

SHACL enables organisations to define rules that specify which properties an entity must possess, cardinality restrictions, and datatype constraints. This formal validation mechanism strengthens data reliability and supports governance frameworks.

Standardisation and validation ensure that knowledge graphs remain consistent, interoperable, and auditable as they scale.

Conclusion

Knowledge graphs represent a powerful paradigm for integrating complex engineering data. Yet their success depends on disciplined modelling, governance structures, and realistic performance strategies.

Future developments will likely deepen the interaction between knowledge graphs, machine learning, and intelligent decision-support systems. For engineers operating in dynamic and interconnected environments, understanding both the strengths and limitations of this technology is essential.

Bibliography

Allemang, D., & Hendler, J. (2011). *Semantic web for the working ontologist*. Morgan Kaufmann.

Baader, F., et al. (2007). *The description logic handbook*. Cambridge University Press.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, *284*(5), 34–43.

Bollobás, B. (1979). *Graph theory: An introductory course* (Vol. 63). Springer-Verlag.

Bollobás, B. (1998). *Modern graph theory* (Vol. 184). Springer-Verlag.

Bollobás, B., & Morris, R. (2026). *Basic graph theory*. Cambridge University Press.

Bondy, A., & Murty, U. S. R. (2008). *Graph theory* (Vol. 244). Springer. https://link.springer.com/book/9781846289699

Diestel, R. (2017). *Graph theory* (5th, Vol. 173). Springer. https://doi.org/10.1007/978-3-662-53622-3

Ehrlinger, L., & Wöß, W. (2016). Towards a definition of knowledge graphs. *SEMANTiCS*.

Euzenat, J., & Shvaiko, P. (2021). *Ontology matching*. Springer.

Fensel, D., Simsek, U., Angele, K., Huaman, E., Kärle, E., Panasiuk, O., Toma, I., Umbrich, J., & Wahler, A. (2020). *Knowledge graphs - methodology, tools and selected use cases*. Springer. https://doi.org/10.1007/978-3-030-37439-6

Gruber, T. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, *5*, 199–220.

Heath, T., & Bizer, C. (2011). *Linked data: Evolving the web into a global data space*. Morgan & Claypool.

Hogan, A., et al. (2021a). *Knowledge graphs*. ACM.

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutiérrez, C., Kirrane, S., O'Sullivan, D., Navigli, R., Neumaier, S., et al. (2021b). *Knowledge graphs* (Vol. 54). ACM Computing Surveys. https://doi.org/10.1145/3447772

Kejriwal, M., Knoblock, C. A., & Szekely, P. (2021). *Knowledge graphs: Fundamentals, techniques, and applications*. The MIT Press.

Klyne, G., & Carroll, J. (2004). Resource description framework (rdf): Concepts and abstract syntax. *W3C Recommendation*.

Leventides, J., & Poulios, N. (2020). *Diffusion on dynamical interbank loan networks* (A. M. Raigorodskii & M. T. Rassias, Eds.). Springer International Publishing. https://doi.org/10.1007/978-3-030-55857-4_13

McGuinness, D., & van Harmelen, F. (2004). Owl web ontology language overview. *W3C Recommendation*.

Nickel, M., Murphy, K., Tresp, V., & Gabrilovich, E. (2016). A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, *104*(1), 11–33. https://doi.org/10.1109/JPROC.2015.2483592

Paulheim, H. (2016). Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, *8*(3), 489–508. https://doi.org/10.3233/SW-160218

Paulheim, H. (2017). Knowledge graph refinement: A survey of approaches and evaluation methods (P. Cimiano, Ed.). *Semant. Web*, *8*(3), 489–508. https://doi.org/10.3233/SW-160218

Prud'hommeaux, E., & Seaborne, A. (2008). Sparql query language for rdf. *W3C Recommendation*.

Trudeau, R. (1994). *Introduction to graph theory*. Dover Books on Mathematics.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph attention networks. *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1710.10903

Wang, J., Yang, B., Zhang, H., Wang, L., & Fang, X. (2022). Digital twin–driven smart manufacturing: Connotation, reference model, applications and research issues. *Robotics and Computer-Integrated Manufacturing*, *73*, 102266.

Wang, Q., Mao, Z., Wang, B., & Guo, L. (2017). Knowledge graph embedding: A review of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, *29*(12), 2724–2743. https://doi.org/10.1109/TKDE.2017.2754499

West, D. B. (2001). *Introduction to graph theory (classic version)* (2nd). Pearson.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(1), 4–24.

Xu, L., He, W., & Li, S. (2019). Enabling predictive maintenance in the era of industry 4.0. *IEEE Access*, *7*, 173730–173743.