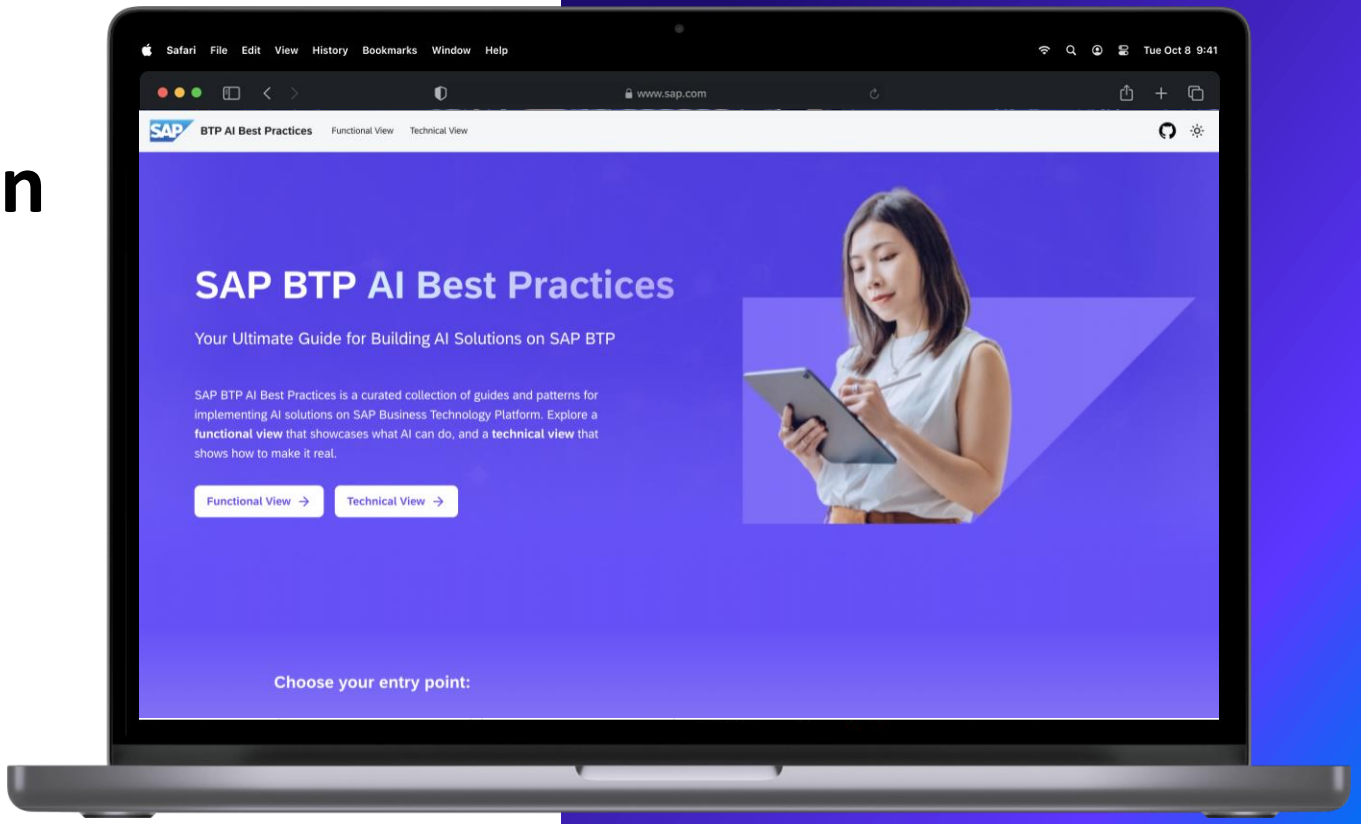
SAP BTP AI Best Practices

Graph-based RAG KG Creation

A best-practice guide to grounding LLMs using structured knowledge graphs in SAP HANA Cloud. Unlock smarter, more trustworthy Retrieval-Augmented Generation (RAG) for enterprise AI.



BTP AI Services Center of Excellence

05.08.2025

Steps

1 Overview

2 Pre-requisites

3 Illustration of the process through an example

4 Expand your knowledge

Why Knowledge Graphs?

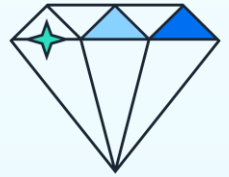
Large Language Models are powerful but can hallucinate. Knowledge graphs provide grounding and context.

Retrieve

Fetch relevant facts from the knowledge graph (RDF or property graph in SAP HANA Cloud)

Augment

Enrich LLM input with structured domain context



Generate

Produce grounded, accurate, and business-aware responses

Expected Outcome

Accurate, explainable responses
Scalable GenAI grounded in SAP HANA Cloud enterprise knowledge
Improved trust, reduced hallucinations, and faster time-to-value

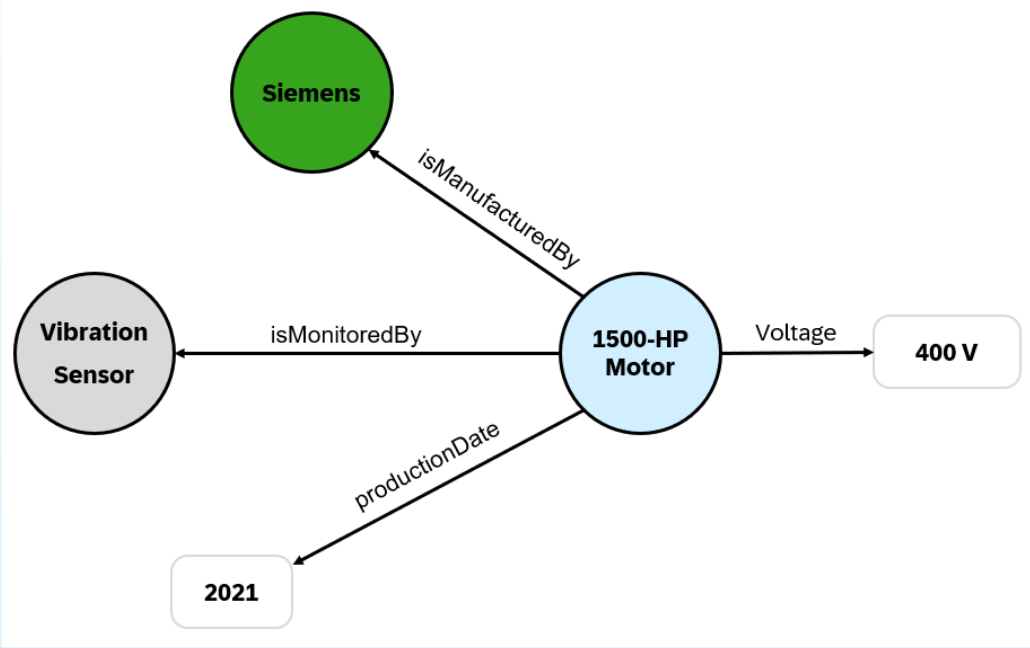
Key Concepts

RAG, KG and RDF

- **Retrieval-Augmented Generation (RAG)** combines a search step with an LLM, finds relevant info first, then uses it to generate better answers

- **Knowledge Graph (KG)** is network of facts made up of entities (things like people or places) and relationships (how they're connected)

- **RDF (Resource Description Framework)** is a standard way to represent data as triples: subject, predicate, and object. The relationships encoded in triples are easy to understand.

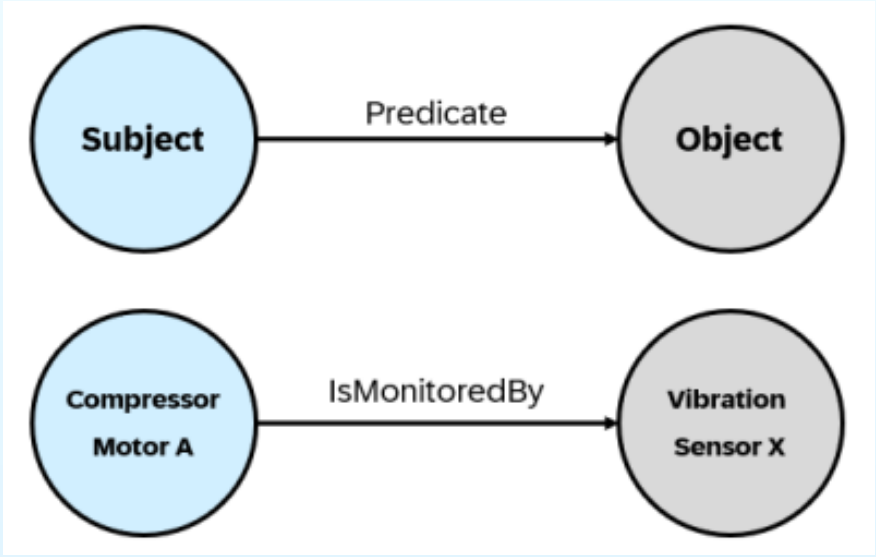


Key Concepts

Triples and Ontologies

- **Triple:** The basic building block of a knowledge graph. In knowledge graph notation, facts are expressed as triples. A triple has three parts: subject – predicate – object (e.g., “Compressor Motor A– IsMonitoredBy – Vibration Sensor X”)

- **Ontology:** An ontology describes how the entities in your KG relate to each other. It's a conceptualization of *what* those entities and relationships mean in that domain, and in the context of the related knowledge graph a representation of the types of entities, their properties, and the relationships between them.



Key Concepts

Additional concepts

- **Entity Extraction: Pull out important** names or terms from text (like “Paris” or “SAP BTP”)
- **Relation Extraction:** Find how entities are related (e.g., “works at”, “founded by”, “related at”).
- **Normalization/Deduplication:** The process of merging duplicate or similar entities to keep the graph clean and avoid redundancy. Example: “IBM”, “I.B.M.”, and “International Business Machines” → one node
- SPARQL is a query language for working with RDF data. You use SPARQL to search, filter, and get information from knowledge graphs built with RDF. Check SAP HANA Cloud Knowledge Graph Engine in SAP HANA Database.

Pre-requisites

Commercial

- SAP AI Core with the “Extended” tier on SAP BTP
- SAP HANA Cloud on SAP BTP
- SAP AI Launchpad

Technical

- Setup SAP Business Technology Platform (SAP BTP) subaccount (Setup Guide)
- Create an instance of SAP HANA Cloud (Setup Guide)
- Create an instance of SAP AI Core (Setup Guide)
- Subscribe to SAP AI Launchpad (Setup Guide)
- Enable ‘Triple Store’ on the SAP HANA Cloud Central tool during instance provisioning

SAP Business Technology Platform (SAP BTP)

- SAP Business Technology Platform (BTP) is an integrated suite of cloud services, databases, AI, and development tools that enable businesses to build, extend, and integrate SAP and non-SAP applications efficiently.

SAP AI Core

- SAP AI Core is a managed AI runtime that enables scalable execution of AI models and pipelines, integrating seamlessly with SAP applications and data on SAP BTP that supports full lifecycle management of AI scenarios.

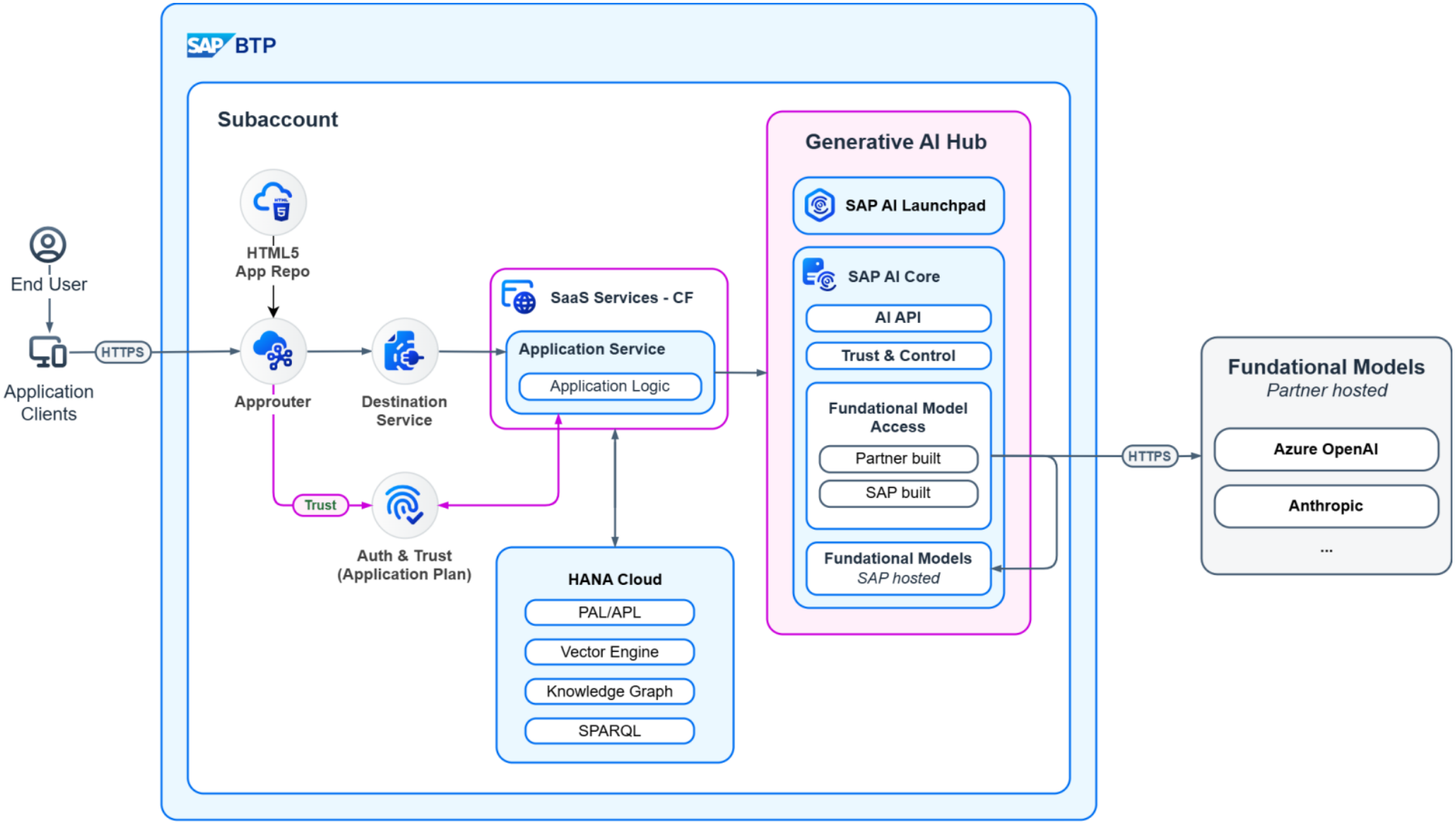
SAP AI Launchpad

- SAP AI Launchpad is a multitenant SaaS application in SAP BTP. Customers can use SAP AI Launchpad to manage AI use cases (scenarios) across multiple instances of AI runtimes.

SAP HANA Cloud with Knowledge Graph Engine

- SAP HANA Cloud is a database as a service that powers mission-critical applications and real-time analytics with one solution at petabyte scale. Use relational, property graph, spatial, vector, and semi-structured data along with embedded machine learning to power intelligent data applications.
- SAP HANA Database Knowledge Graph Engine is a built-in capability that lets you model, store, and query enterprise knowledge as interconnected facts, giving your applications instant access to rich context and relationships without leaving the HANA environment.

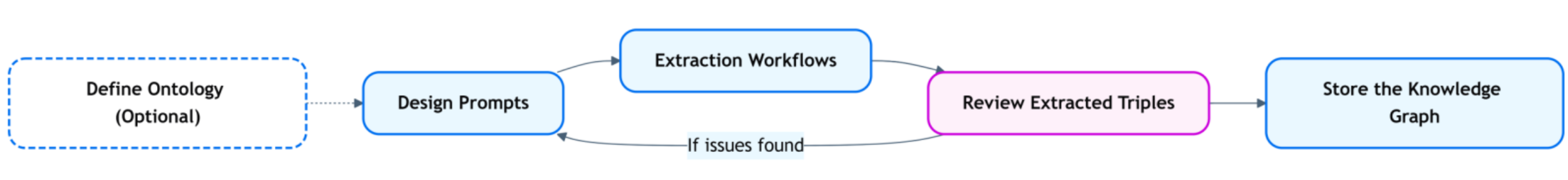
High-level reference architecture



Knowledge Graph Creation Process Flow

Example process flow

1. Define Ontology (optional)
2. Design Prompts
3. Create Extraction Workflows
4. Review Extracted Triples
5. Store the Knowledge Graph



Example use-case: equipment specification documents

Example use-case

Business Context

This example shows how a company can use **large equipment specification documents**, such as PDFs about industrial motors, to build a knowledge graph.

Goal

The goal is to **extract clear facts** about things like power, voltage, and manufacturer for each motor **and create a knowledge graph** based on this information.

Outcome

This can help the company **answer questions** about the equipment, **quickly find** important parts, and **organize its data** well for reporting or maintenance.

Step 1: Define Ontology (optional)

A domain ontology defines the types of entities, properties, and relationships you can use in your Knowledge Graph.

While optional, an ontology helps ensure that extracted facts are consistent, focused, and aligned with your business requirements.

Examples:

- Use existing schemas like schema.org/Product or DMTF CIM Standard if possible.
- List core attributes such as *enginePower*, *torque*, *workingVoltage* for industrial equipment.

Additional Guidance:

- For quick prototyping, simply write out the main types and properties as a list or table — this already forms a basic ontology.
- For production, formalize your ontology as a knowledge graph using OWL and manage it directly in SAP HANA Cloud.
- You can also create an ontology by analyzing your real data: review repeated terms and property names in your documents, group similar concepts, and validate your list with domain experts.

Step 2: Design Prompts

We make explicit what labels, relationships, and structures the LLM must follow

```
You are a top-tier algorithm designed for extracting information in structured formats to build a knowledge graph.
- **Nodes** represent entities and concepts. They're akin to Wikipedia nodes.
- The aim is to achieve simplicity and clarity in the knowledge graph, making it accessible for a vast audience.
```

We establish rules for labeling, naming and identification. We also define what relationships are permitted, as well the expected data format.

```
- **Consistency**: Ensure you use basic or elementary types for node labels.
- For example, when you identify an entity representing an Equipment, always label it as **"Equipment"**. Avoid using more specific terms like "1500-HP Equipment".
- **Node IDs**: Never utilize integers as node IDs. Node IDs should be names or human-readable identifiers found in the text.
- **Allowed Node Labels**: Equipment, Voltage, Power, Frequency, Temperature, Current, Efficiency, Manufacturer, Enclosure, Bearing, Test, Location
- **Allowed Relationship Types**: is a, has, includes, is rated for, operates at, measures, is equipped with, is tested for, provides, is designed for

  Handling Numerical Data and Dates
- Numerical data, like voltage, current, efficiency, etc., should be incorporated as attributes or properties of the respective nodes.
- **No Separate Nodes for Dates/Numbers**: Do not create separate nodes for dates or numerical values. Always attach them as attributes or properties of nodes.
- **Property Format**: Properties must be in a key-value format.
- **Naming Convention**: Use camelCase for property keys, e.g., `ratedVoltage`.
```

We specify the property format and naming and include a real-world example:

```
**Strict Compliance**
Adhere to the rules strictly.
"""
    "EXAMPLE\n"
    "Equipment Model A, a 1125-HP induction Equipment, is rated for 4160V at 60Hz with a full load speed of 3222 RPM. The Equipment is designed with Class F insulation and a Class B temperature rise. It is equipped with RTDs for temperature detection and vibration sensors. "
    "Efficiency: 95% at full load. Power factor at full load: 0.90. Noise level: 85 dBA at 1 meter.\n\n"
    f"Output: (Equipment Model A, is a, Equipment){KG_TRIPLE_DELIMITER}"
    f"(Equipment Model A, has, 1125-HP power){KG_TRIPLE_DELIMITER}"
    f"(Equipment Model A, operates at, 4160V){KG_TRIPLE_DELIMITER}"
    f"(Equipment Model A, operates at, 60Hz){KG_TRIPLE_DELIMITER}"
....
"""
```

Step 3: Create Extraction Workflows

3.1 Import the required libraries

3.2 Load environment variables from a local .env file (API keys, DB creds, etc.)

3.3 Create triples based on text from pdf

```python
# Create a proxy client for the AI Hub
proxy_client = get_proxy_client('gen-ai-hub')
chat_llm = ChatOpenAI(proxy_model_name='gpt-4o', proxy_client=proxy_client, temperature=0.0)

# Build the Knowledge Graph using a Language Model (LLM)
loader = PyPDFLoader("2312_-_Four_1500-HP_Main_Air_Compressor_Motors_FINAL.pdf")
docs = loader.load()
doc_content = 'This is a compressor specification: '
for doc in docs:
    doc_content = doc_content + ' ' + doc.page_content

index_creator = GraphIndexCreator(llm=chat_llm)

graph = index_creator.from_text(text=doc_content, prompt=KNOWLEDGE_TRIPLE_EXTRACTION_PROMPT)

# Print triples
triples  = graph.get_triples()
print(triples)
```

Step 4: Review Extracted Triples

```
triples  = graph.get_triples()
print(triples)

# Example of output:
# [('1500-HP Main Air Compressor Motor', 'Equipment', 'is a'), ('1500-HP Main Air Compressor Motor', '1500-HP power', 'has'), ('1500-HP Main Air Compressor Motor', '4160V', 'operates at'), ('1500-HP Main Air Compressor Motor', '60Hz', 'operates at'), ('1500-HP Main Air Compressor Motor', 'efficiency of 95% or higher at 100% load', 'has'), ('1500-HP Main Air Compressor Motor', 'power factor of 0.936 or higher', 'has'), ('1500-HP Main Air Compressor Motor', 'sound level of 85 dB(A) at 3 feet', 'has'), ('1500-HP Main Air Compressor Motor', 'Class F insulation', 'is designed with'), ('1500-HP Main Air Compressor Motor', 'Class B temperature rise', 'is designed with'), ('1500-HP Main Air Compressor Motor', 'vibration sensors', 'is equipped with'), ('1500-HP Main Air Compressor Motor', 'winding and bearing temperature sensors', 'is equipped with'), ('1500-HP Main Air Compressor Motor', 'space heaters', 'is designed for'), ('1500-HP Main Air Compressor Motor', 'main and auxiliary terminal boxes', 'is equipped with'), ('1500-HP Main Air Compressor Motor', 'continuous duty at 90°C', 'is designed for'), ('1500-HP Main Air Compressor Motor', 'sound pressure level', 'is tested for'), ('1500-HP Main Air Compressor Motor', 'vibration measurements', 'is tested for'), ('1500-HP Main Air Compressor Motor', 'heat run', 'is tested for'), ('1500-HP Main Air Compressor Motor', 'surge comparison test', 'is tested for'), ('1500-HP Main Air Compressor Motor', 'DC high-potential test', 'is tested for'), ('1500-HP Main Air Compressor Motor', 'bearing dimensional and alignment checks', 'is tested for'), ('1500-HP Main Air Compressor Motor', 'rotor residual unbalance verification test', 'is tested for'),...

```

Step 5: Store the Knowledge Graph

```python
# Functions for converting values to IRIs and literals
def to_iri(value: str, base: str = "http://example.com/") -> str:
    import re
    encoded_part = re.sub(
        r"[^a-zA-Z0-9\\-_\\.]", lambda m: f"%{ord(m.group(0)):02X}", value
    )
    return f"<{base}{encoded_part}>"

def to_literal(value: str) -> str:
    if value is None:
        return '""'
    escaped = (
        value.replace("\\", "\\\\")
        .replace('"', '\\"')
        .replace("\n", "\\n")
        .replace("\r", "\\r")
    )
    return f'"{escaped}"'
```

Step 5: Store the Knowledge Graph

```python
# Connection parameters
HANA_ADDRESS = os.getenv("hana_address")
HANA_PORT = os.getenv("hana_port")
HANA_USER = os.getenv("hana_user")
HANA_PASSWORD = os.getenv("hana_password")
GRAPH_NAME = "product_custom_HANA_KG"

# Connecting to SAP HANA
conn = dbapi.connect(
    address=HANA_ADDRESS, port=HANA_PORT, user=HANA_USER, password=HANA_PASSWORD
)
cursor = conn.cursor()

# Defining IRI for graph
graph_iri = to_iri(GRAPH_NAME, base="http://graph/")

# Bulk insertion of graph data
print("[INFO] Inserting data into the graph...")

# Prepare all triples in one SPARQL INSERT statement
triples_statements = []
for subj, pred, obj in triplets:
    subj_iri = to_iri(subj, base="http://example.com/resource/")
    pred_iri = to_iri(pred, base="http://example.com/property/")
    obj_literal = to_literal(obj)
    triples_statements.append(f"    {subj_iri} {pred_iri} {obj_literal} .")

# Single bulk INSERT query
sparql_bulk_insert = f"""
INSERT DATA {{
  GRAPH {graph_iri} {{
{chr(10).join(triples_statements)}
  }}
}}
""".strip()

try:
    cursor.callproc("SPARQL_EXECUTE", [sparql_bulk_insert, "", None, None])
    print(f"[INFO] Bulk insert of {len(triplets)} triples successful.")
except Exception as e:
    print(f"[ERROR] Bulk insert failed: {e}")
    print(f"[ERROR] Query:\n{sparql_bulk_insert}")

conn.commit()
cursor.close()


# Expected output
# [INFO] Inserting data into the graph...
# [INFO] Bulk insert of 111 triples successful.
```

Step 5: Store the Knowledge Graph

```python
# Form and execute a SPARQL query
sparql_query = f"""
SELECT DISTINCT ?s ?p ?o
FROM {graph_iri}
WHERE {{
  ?s ?p ?o
}}
"""

print("[INFO] Executing SPARQL SELECT query...")
try:
    cursor.execute(f"SELECT * FROM SPARQL_TABLE('{sparql_query}')")
    results = cursor.fetchall()
    if results:
        print("[RESULT] Query results:")
        for row in results:
            # usualy row = (subject, predicate, object)
            print(row)
    else:
        print("[RESULT] No data found.")
except Exception as e:
    print(f"[ERROR] SPARQL SELECT query failed: {e}")

# Closing the connection
cursor.close()
conn.close()
print("[INFO] Connection closed.")

# Expected output
# ('http://example.com/resource/Compressor', 'http://example.com/property/terminal%20box%20certifications', 'is designed for')
# ('http://example.com/resource/Compressor', 'http://example.com/property/terminal%20box%20shipment%20protection', 'is designed for')
# ('http://example.com/resource/Compressor', 'http://example.com/property/terminal%20box%20quick%20disconnects', 'is designed for')
# ('http://example.com/resource/Compressor', 'http://example.com/property/IEEE%20522', 'has industry standard')
# ('http://example.com/resource/Compressor', 'http://example.com/property/ASTM%20A345%2D19', 'has industry standard')
# ('http://example.com/resource/Compressor', 'http://example.com/property/NEMA%20MG%201%2D20.18', 'has industry standard')
# ....
```

Expand your knowledge: Tutorials and Learning Journeys

Learn more!

Recommended

- Building Intelligent Data Applications with SAP HANA Cloud Knowledge Graphs (Discovery Center Mission)
- Python Analysis with Multi-Model Data in SAP HANA Cloud: Learn about the enhancements made to the hana-ml library to support the multi-model capabilities of SAP HANA Cloud, SAP HANA database.

Discovery Mission

- Building Intelligent Data Applications with SAP HANA Cloud Knowledge Graphs.
- Reference code for Discovery mission SAP HANA Cloud Knowledge Graph Engine (KGE)

Reference Code

- **SAP BTP AI Best Practices - Sample Code**

Other

- Visualize Spatial Data in SAP HANA Cloud Using Jupyter Notebook: Connect your Python script from Jupyter Notebook to SAP HANA Cloud, SAP HANA database, creating a visualization of spatial data in SAP HANA Cloud.
- Analyze Graph Workspace in SAP HANA Cloud, SAP HANA Database: Learn to use algorithms to analyze and process a Graph Workspace in SAP HANA Cloud, SAP HANA database.
- Smart Multi-Model Data Processing with SAP HANA Cloud Get to know spatial and graph processing in SAP HANA Cloud, SAP HANA database with 10 hands-on exercises. Use your own (trial) database and our sample data to experience the advantages of smart multi-model data processing.

Expand your knowledge: Additional Materials

Learn more!

SPARQL syntax in SAP HANA cloud

- SAP HANA SQL Reference Guide for SAP HANA Platform The SAP HANA System Views Reference describes all SYS schema views that allow you to query for various information about the system state using SQL operations.
- SPARQL SELECT Queries Using SPARQL_TABLE SPARQL SELECT queries examples with SAP HANA database function SPARQL_TABLE.
- SPARQL Query Forms Using SPARQL_EXECUTE SPARQL queries with the built-in SAP HANA database procedure SPARQL_EXECUTE.

KG creation

- Choosing Between Knowledge Graphs and Property Graphs in SAP HANA Cloud and Why Both Matter Guides users on evaluating use-cases to effectively select between Knowledge Graphs, suited for semantic querying via RDF, and Property Graphs, optimized for relationship-intensive analytical workloads, within SAP HANA Cloud.
- Get Started with SAP HANA Graph Get an overview of SAP HANA capabilities related to graph processing.
- Take Your First Steps with the SAP HANA Graph Engine Learn how to prepare your data for the SAP HANA Cloud, SAP HANA database Graph Engine how to create a Graph Workspace.

KG use cases

- Try Out Multi-Model Functionality with the SAP HANA Database Explorer and Database Objects App Explore knowledge graph, property graph, JSON document store, and spatial capabilities in the SAP HANA database explorer.
- Connecting the Facts: SAP HANA Cloud's Knowledge Graph Engine for Business Context Learn how to extend and personalize SAP applications. Follow the SAP technology blog for insights into SAP BTP, ABAP, SAP Analytics Cloud, SAP HANA, and more.
- KG: Business Benefits and Use Cases Use cases include intelligent data applications, generative AI, data fabric establishment, and improved decision making.

Contributors

Rzhaksynskyi, Andrii

Marques, Luis

CIGAINA, MARCO

Thank you