3. Knowledge Graphs

Knowledge Graph Theory — Stored Semantic Data

Knowledge Graph = Meaningful graph database

A Knowledge Graph is built from three fundamental components:

- **Nodes (Entities)** → Things that exist
- **Edges (Relationships)** → Connections between things
- **Labels (Meaning)** → Semantic interpretation of those connections

Together, they allow machines to store **data + meaning + logic**.

Nodes (Entities)

Nodes represent real-world objects or concepts.

Examples:

- Alice → Person (entity)
- Pizza → Food (entity)

Nodes answer the question:

> What exists in this domain?

Edges (Relationships)

Relationships connect entities.

Example:

(Alice) – eats –> (Pizza)

This means:

- Alice is connected to Pizza
- The connection type is `eats`

In simple terms:

- Relationship = the **link between nodes**

Labels (Meaning)

Meaning defines what the relationship actually represents.

Ontology adds semantics:

- Alice is a Person
- Pizza is Food
- eats = consuming edible items
- Only Persons can eat Food

In simple terms:

- Meaning = the **understanding of the link**

This enables:

- Reasoning
- Validation
- Inference

Intuitive Analogy

- **Relationship** = Wire
- **Meaning** = Electrical rules

Wire connects.

Rules define how electricity flows safely.

RDF/OWL Triples

- In RDF/OWL, triples store data as:
  - Subject (Node) → Predicate (Edge/Relationship) → Object (Node)
  - In other words, there is no difference between relationships and meaning at the storage level.

Two Layers Inside One Triple

Example:

(Alice) – eats –> (Pizza)

This contains two layers:

Graph Structure Layer (Relationship View)

- Node → Edge → Node
- Just connectivity

Semantic Layer (Meaning View)

- Who Alice is
- What Pizza is
- What "eats" logically means

Full Example Graph

```
(Alice) — eats —> (Pizza)
(Alice) — type —> (Person)
(Pizza) — type —> (Food)
```

Interpretation:

- Alice is a Person
- Pizza is Food
- Alice eats Pizza
- This is logically valid

Knowledge Graph Components Summary

Nodes → Entities (things)

Edges → Relationships (connections)

Labels → Meaning (semantics)

Each layer adds more intelligence:

- Nodes give existence
- Edges give structure
- Meaning gives understanding
- Meaning gives understanding

Why This Matters

Without meaning:

- Graph = connected data

With meaning:

- Graph = knowledge system
- Automatic classification
- Logical consistency
- AI-friendly reasoning

Normal Graph vs Knowledge Graph

Normal Graph

```
Node — Edge — Node
```

Knowledge Graph

```
Node — Edge + Meaning — Node
```

Meaning is what turns data into knowledge.

Traditional DB vs Knowledge Graph

Relational tables	Graph / semantic model
Tables	Graph structure
Schema-based	Meaning-based (semantic model)
No reasoning	Automatic inference
Hard to integrate	Easy data linking

Structure Comparison

Traditional Database

- Data stored in rows and columns
- Relationships handled using foreign keys
- Optimized for transactions and fixed schemas

Knowledge Graph

- Data stored as nodes and edges
- Relationships are first-class citizens
- Optimized for connected data and exploration---

GraphDB Databases

- Ontotext GraphDB is a highly efficient, scalable and robust graph database with RDF and SPARQL support.
- We can use the free version for learning and prototyping.

Download and Install GraphDB

Download link: https://www.ontotext.com/products/graphdb/

Request a free GraphDB license

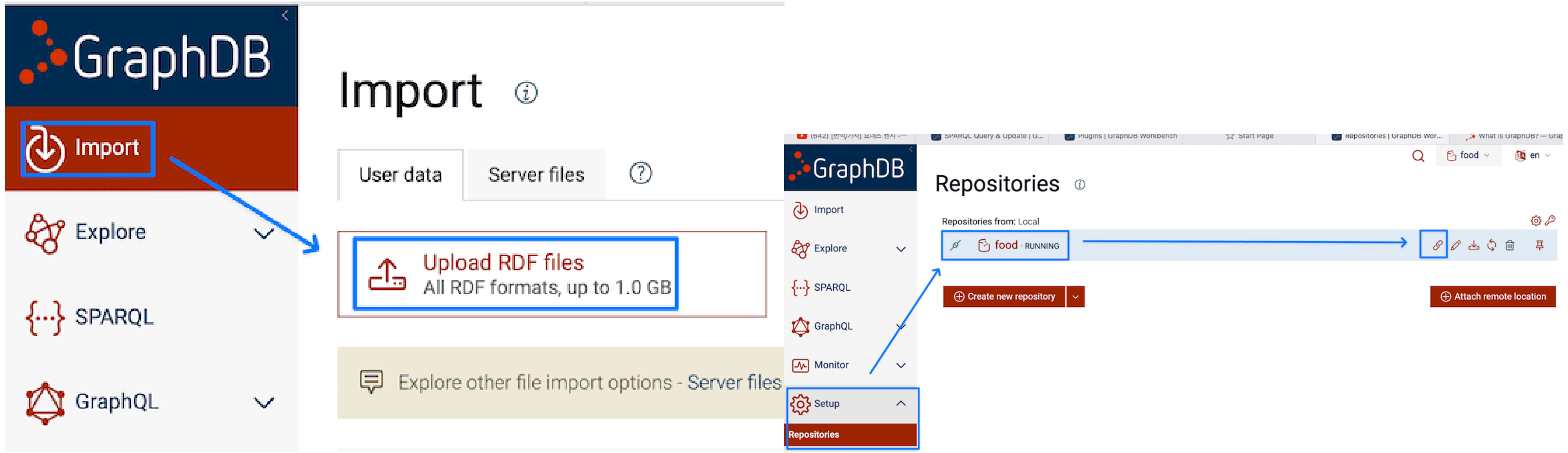
Using GraphDB

Start GraphDB server

1. Use web browser to access the GraphDB Workbench at (for example): http://localhost:7200
2. Use any programming language to connect to the SPARQL endpoint at (for example): http://localhost:7200/repositories/your-repo

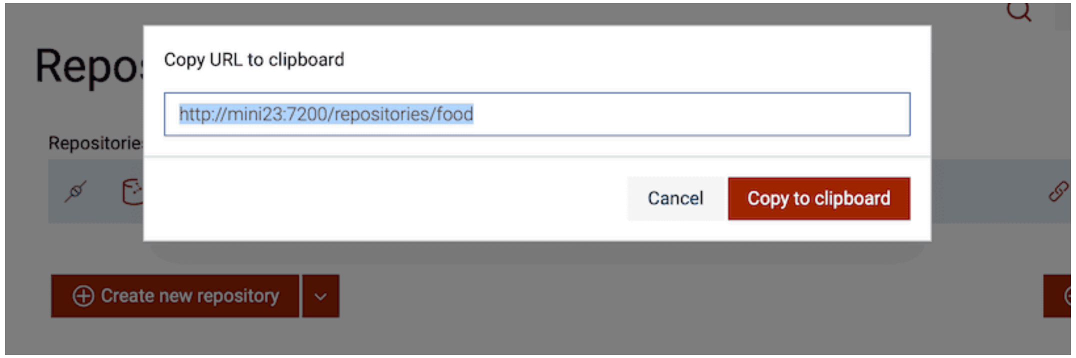
Protege and GraphDB Integration

- In Protege: Save your ontology as an RDF/OWL file.
- In GraphDB: Import the RDF/OWL file to create a knowledge graph.



Access the SPARQL Endpoint

- Get the SPARQL endpoint URL to query the knowledge graph.



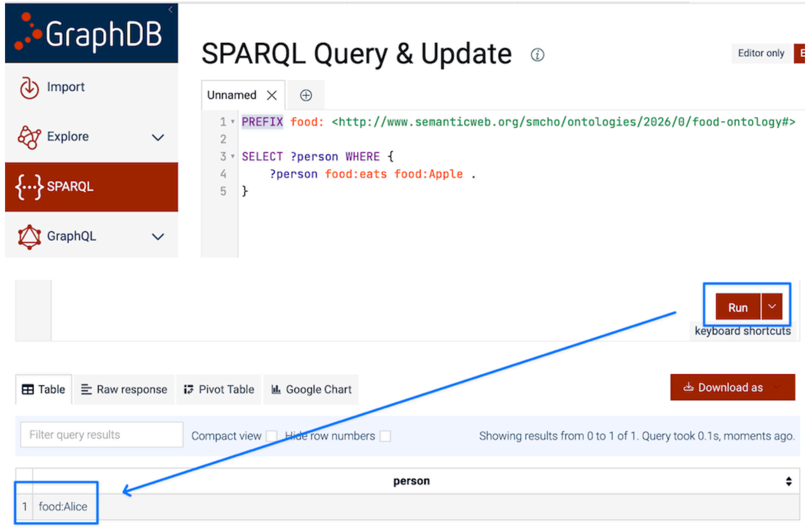
Querying the Knowledge Graph with SPARQL

This is the SPARQL query to find all persons who eat Apple:

```
PREFIX food: <http://www.semanticweb.org/smcho/ontologies/2026/0/food-ontology#>

SELECT ?person WHERE {
    ?person food:eats food:Apple .
}
```

Executing the SPARQL Query on GraphDB Workbench



Executing the SPARQL Query via Python

We need to retrieve the JSON results from the SPARQL endpoint.

- From the `SELECT ?person`, the answer variable is `?person`.
- We can retrieve the value from the JSON response.

```python
import requests

endpoint = "http://mini23:7200/repositories/food"

query = """..."""

response = requests.get(
    endpoint,
    headers={"Accept": "application/sparql-results+json"},
    params={"query": query},
    timeout=30,
)

response.raise_for_status()

data = response.json()

for row in data["results"]["bindings"]:
    print(row["person"]["value"])
```