🧪 Hands-on Lab: Build a Mini Ontology and Query It in Neo4j

> You'll build a **mini ontology** (e.g., in healthcare), export data from **Protégé**, load it into **Neo4j**, and then query the knowledge using **Cypher**.

🧠 PART 1: Create a Mini Ontology Using Protégé

✅ Step 1: Download and Install Protégé

1. Go to the official website: https://protege.stanford.edu
2. Download the **latest Protégé Desktop version** (usually a ZIP or DMG file).
3. Extract the ZIP and run `Protégé` (no installation required).

✅ Step 2: Create a New OWL Ontology

1. Open Protégé → `File > New Ontology`
2. Choose the **OWL 2 DL** format
3. Give your ontology a base IRI (e.g., `http://www.example.org/healthcare`)
4. Save your ontology file (e.g., `healthcare.owl`)

✅ Step 3: Define Ontology Structure

We'll model a simple **Healthcare domain**:

🔹 **Classes:**

- `Patient`
- `Doctor`
- `Disease`
- `Treatment`

🔹 **Object Properties:**

- `hasDisease` (Patient → Disease)
- `receivesTreatment` (Patient → Treatment)
- `treatedBy` (Patient → Doctor)

🔹 **Data Properties:**

- `hasAge` (Patient → integer)

- `hasName` (for all classes → string)

✅ **Create Them in Protégé:**

- Go to the **Classes tab** → Click `+` to add each class
- Go to **Object Properties** → Add and set `Domain` and `Range`
- Go to **Data Properties** → Add string/integer properties

✅ Step 4: Add Individuals (Instances)

1. Switch to the **Individuals tab**
2. Create sample entries:
   - `Alice` (a `Patient`, hasAge = 34, hasDisease = Diabetes, treatedBy = DrSmith)
   - `DrSmith` (a `Doctor`)
   - `Diabetes` (a `Disease`)
   - `Insulin` (a `Treatment`)

✅ Step 5: Export Ontology

We need to export in **RDF/XML or Turtle** format.

1. Go to `File` → `Export Ontology`
2. Choose format: **Turtle (.ttl)** or **RDF/XML (.rdf)**
3. Save as `healthcare.ttl` or `healthcare.rdf`

🕸️ PART 2: Load Ontology Data into Neo4j and Query with Cypher

---

✅ Step 1: Install Neo4j Desktop or Neo4j Aura

Option A: Neo4j Desktop (Recommended for Local Use)

1. Download from: https://neo4j.com/download
2. Install and open Neo4j Desktop
3. Create a new **local database project**

Option B: Neo4j Aura (Cloud)

1. Sign up: https://neo4j.com/cloud/aura/
2. Create a free "sandbox" instance
3. Get the Bolt connection URI, username, and password

✅ Step 2: Prepare RDF Import Plugin

Neo4j doesn't natively support RDF, but you can use the **Neosemantics plugin (n10s)**.

1. Enable Neosemantics in Neo4j Desktop:
    - Open database → `Plugins` tab → Install **Neosemantics**
2. Start your database

✅ Step 3: Open Neo4j Browser and Initialize n10s

Open Neo4j Browser (at `http://localhost:7474`) and run:

```
// Create the n10s configuration
CALL n10s.graphconfig.init()
```

✅ Step 4: Load RDF Data from Your File

Let's assume your file is `healthcare.ttl` and accessible locally.

You can either:

- **Host it via a local web server**, or
- **Use file import with APOC** (for advanced users), or
- **Paste the RDF content into the browser**

But the simplest way is to upload the file into **Neo4j's import folder**.

🔹 Example (Turtle format):

```
CALL n10s.rdf.import.fetch("file:///healthcare.ttl", "Turtle")
```

✅ If successful, your ontology triples will now be in Neo4j as nodes and relationships!

✅ Step 5: Query the Graph Using Cypher

Let's try some sample queries:

🔍 Find all patients:

```
MATCH (p)-[:`rdf:type`]->(:`http://www.example.org/healthcare#Patient`)
RETURN p
```

🔍 Find patients and their diseases:

```
MATCH (p)-[:`http://www.example.org/healthcare#hasDisease`]->(d)
RETURN p, d
```

🔍 **Find patients treated by a specific doctor:**

```
MATCH (p)-[:`http://www.example.org/healthcare#treatedBy`]->(d)
WHERE d.name = "DrSmith"
RETURN p.name
```

🔍 **Count patients per disease:**

```
MATCH (p)-[:`http://www.example.org/healthcare#hasDisease`]->(d)
RETURN d, COUNT(p) AS num_patients
```

✅ Summary of What You've Done

- Modeled a mini **ontology** with classes, properties, and individuals
- Exported RDF/Turtle data from Protégé
- Loaded it into **Neo4j** using the **n10s Neosemantics plugin**
- Queried relationships using **Cypher**
