Building a Knowledge Graph

From IoT Data to Knowledge

The Badevel Living Lab Story

🏘️ 📊 🔗

Joseph Azar

TD Session 1 - CP58 UTBM Sevenans

Chapter 1

The Challenge

🎯 🤔 💡

Meet Badevel

📍 **Location:** A small municipality in France

🎯 **Goal:** Become a smart, sustainable living lab

💡 **Solution:** Deploy IoT sensors everywhere!

What they monitor:

- 🌡️ Temperature in all public buildings
- 💨 Air quality and humidity
- 💡 Energy consumption
- ♻️ Waste containers status
- 🚗 Vehicle tracking

The Problem They Face

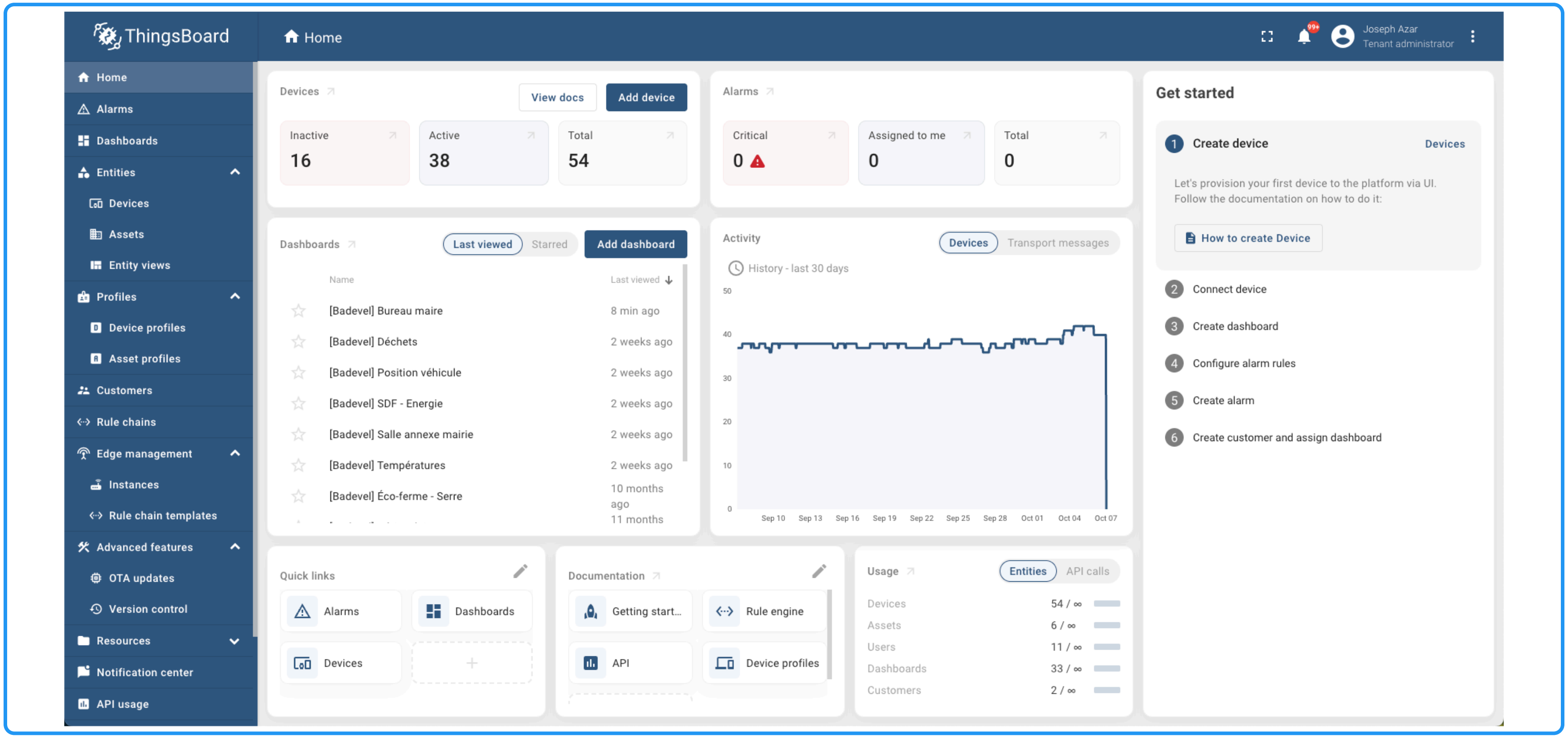
Data Everywhere, Knowledge Nowhere!

Questions they can't easily answer:

- "If the device in the Mayor's office fails, what data do we lose?"
- "Which sensors are monitoring the Town Hall building?"
- "How many temperature sensors do we have across all locations?"
- "What's the complete monitoring setup for the eco-farm?"

Why? Their data is trapped in a dashboard!

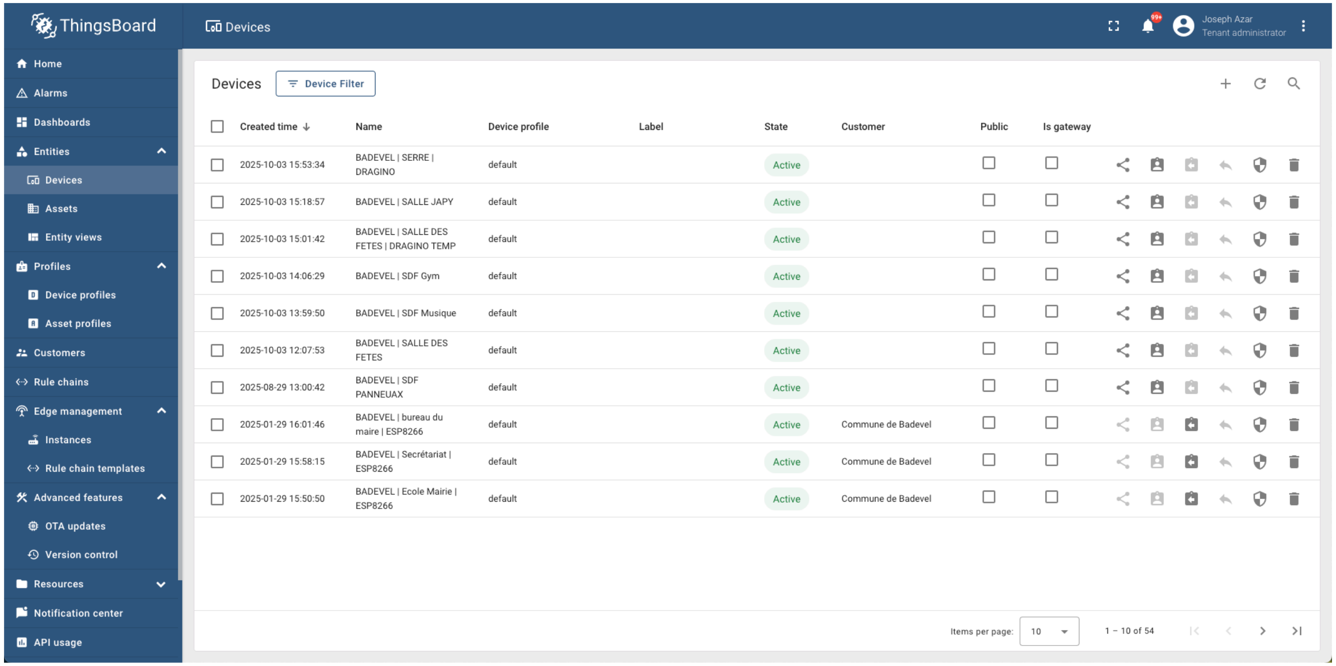
Their Current System: ThingsBoard



What we see:

- 54 devices (38 active, 16 inactive)
- 6 assets, 33 dashboards
- Real-time activity monitoring

Looking at Their Devices



🤔 ***Think about it:***

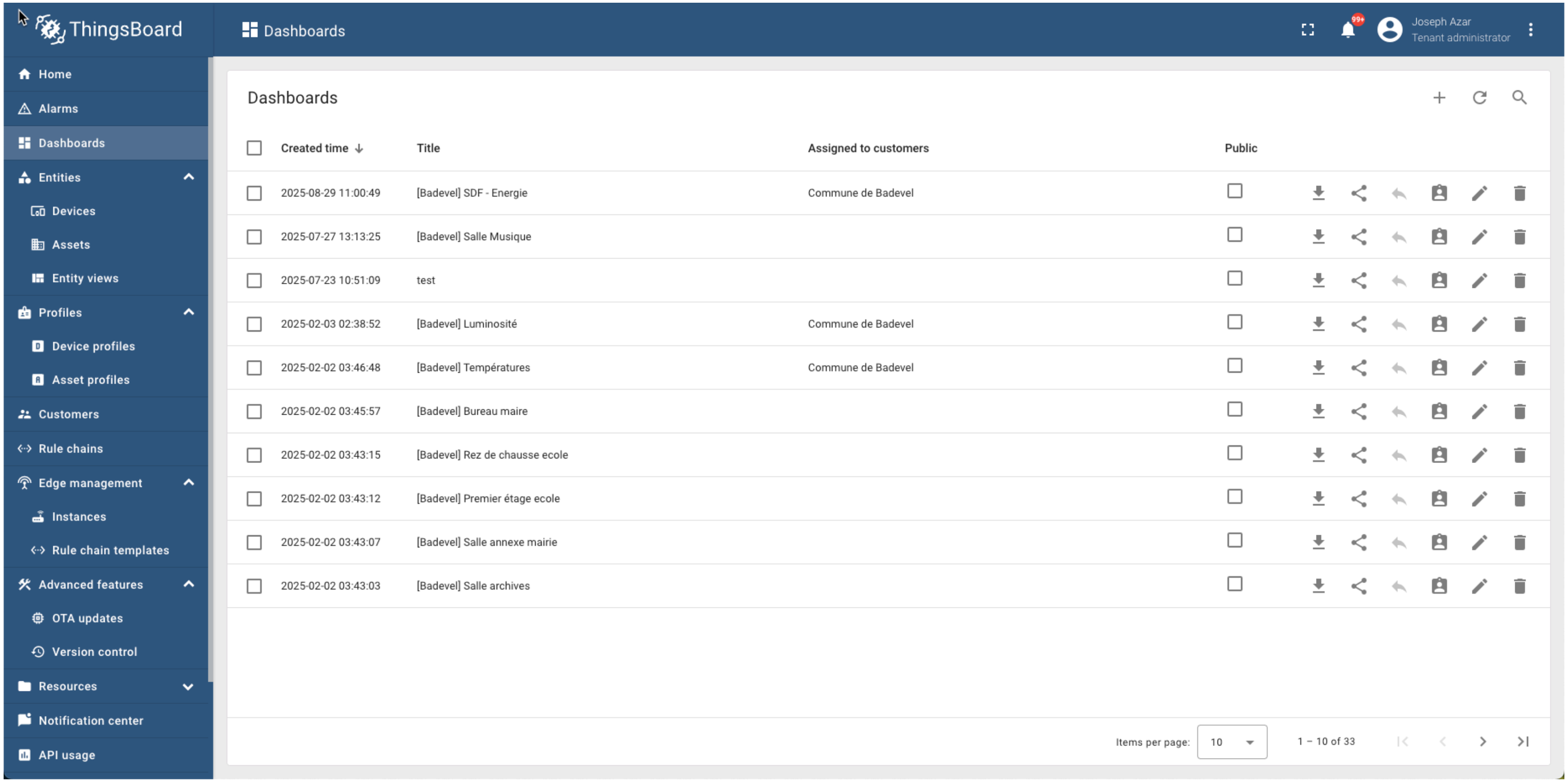
Each device has a name like "BADEVEL | bureau du maire | ESP8266"

This tells us:

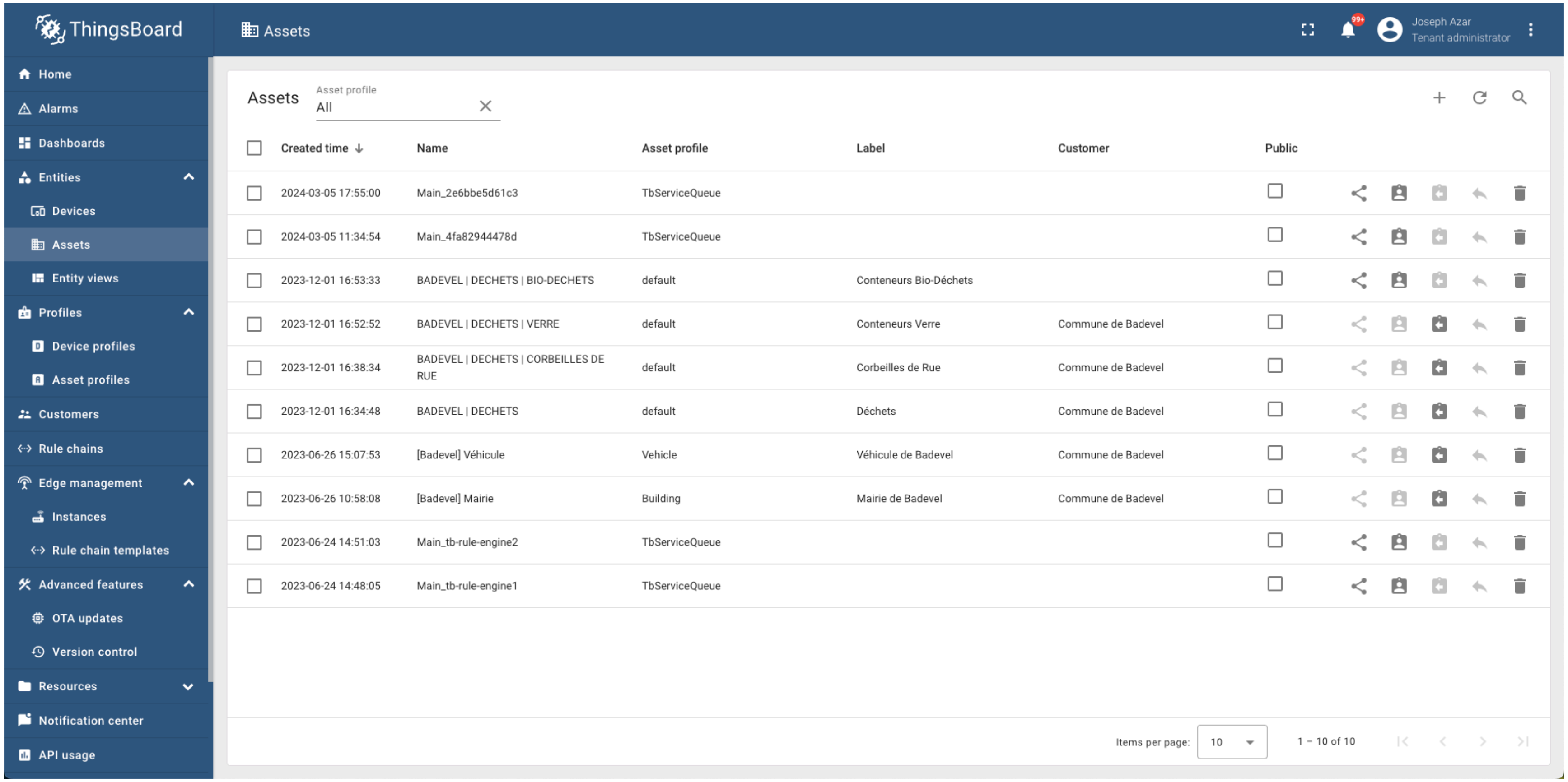
- *Location: "bureau du maire" (Mayor's office)*
- *Device type: ESP8266*
- *It belongs to BADEVEL*

But how do we query these connections?

Looking at Their Dashboards



Looking at Their Assets



The Solution: A Knowledge Graph!

Transform data into interconnected knowledge

❌ Dashboard (Current)

- Good for viewing
- Hard to query
- No relationships
- Fixed reports

✅ Knowledge Graph (Goal)

- Flexible querying
- Explore relationships
- Ad-hoc questions
- Deep insights

Today: We'll build this together! 🚀

Chapter 2

Understanding the Data

🔍 📋 🗺️

From Raw Data to Structure

What Do We Have in Badevel?

1 Locations

Physical places where things happen

Examples from the dashboard:

- 🏛 **Mairie** (Town Hall) - a building
- 📝 **Bureau du maire** (Mayor's office) - a room inside Mairie
- 🎉 **Salle des Fêtes** (Festival Hall) - a building
- 🌱 **Éco-ferme - Serre** (Eco-farm greenhouse) - outdoor
- 🏋 **SDF Gym** (Sports facility gym) - a room

💡 ***First insight:*** *Some locations are inside other locations!*

Mayor's office → PART_OF → Town Hall

What Do We Have in Badevel?

2 Devices

IoT hardware that collects data

Examples from the dashboard:

- 📡 **BADEVEL | bureau du maire | ESP8266**
  - Type: ESP8266 microcontroller
  - Location: Mayor's office
  - State: Active
  - Created: 2025-01-29
- 📡 **BADEVEL | SERRE | DRAGINO**
  - Type: DRAGINO LoRaWAN sensor
  - Location: Greenhouse
  - State: Active

💡 ***Second insight:** Devices are located in specific places!*

What Do We Have in Badevel?

3 Sensors

What each device can measure

The ESP8266 in Mayor's office has:

- 🌡️ **Temperature** sensor (°C)
- 💧 **Humidity** sensor (%)
- 🌀 **Pressure** sensor (hPa)
- 💡 **Luminosity** sensor (lux)

The DRAGINO in greenhouse has:

- 🌡️ **Temperature** sensor (°C)

💡 ***Third insight:*** *Devices have sensors!*

What Do We Have in Badevel?

4 Assets

Things being monitored

- 🏛 [Badevel] Mairie (Building)
- 🚗 [Badevel] Véhicule (Vehicle)
- ♻️ BADEVEL | DECHETS | VERRE (Waste containers)

5 Customer

Who owns everything

- 🏛 **Commune de Badevel**
  - Owns all devices
  - Manages all assets
  - Controls all dashboards

💡 ***Fourth insight:*** *Ownership relationships!*

Device → BELONGS_TO → Customer

Asset → RELATED_TO → Device (monitoring)

Putting It All Together

Our Data Model (Entities)

Entity	Properties	Example
Location	name, type, description	Bureau du maire (room)
Device	name, deviceType, state, createdTime	ESP8266 (Active)
Sensor	type, unit	Temperature (°C)
Asset	name, assetType, label	Mairie (Building)
Customer	name	Commune de Badevel

The Connections (Relationships)

How Things Connect

Relationship	Pattern	Example
LOCATED_IN	Device → Location	ESP8266 → Bureau du maire
HAS_SENSOR	Device → Sensor	ESP8266 → Temperature
PART_OF	Location → Location	Bureau → Mairie
BELONGS_TO	Device → Customer	ESP8266 → Commune de Badevel
RELATED_TO	Asset → Device	Mairie Asset → ESP8266

This is our Knowledge Graph Model! 🎉

Visualizing Our Model

```
    (Customer: Commune de Badevel)
           ↑
           | BELONGS_TO
           |
    (Device: ESP8266)
      ↓          ↓
   LOCATED_IN   HAS_SENSOR
      ↓          ↓
(Location)    (Sensor: Temperature)
      ↑
      | PART_OF
      |
(Location: Mairie)
```

Now we can answer questions like:

- "Show me all devices in the Mairie building (including rooms inside it)"
- "What sensors does the Commune de Badevel have deployed?"
- "Find all temperature sensors and where they are located"

Chapter 3

Building the Graph

🏗️ ⚒️ 🔨

Hands-On with Neo4j

Before We Start: Setup Check

✋ **Make sure you have:**

✅ **Neo4j Desktop** or **Neo4j AuraDB** running

✅ **Neo4j Browser** open and connected

✅ **Empty database** ready to use

Quick test - run this query:

```
RETURN "Hello Neo4j!" AS message
```

If you see "Hello Neo4j!" you're ready! 🎉

Step 1: Create Our Locations

Let's create the physical places in Badevel

```
// Create Location nodes
CREATE (:Location {id: 'loc_mairie', name: 'Mairie',
        type: 'building', description: 'Town Hall of Badevel'})

CREATE (:Location {id: 'loc_bureau_maire', name: 'Bureau du maire',
        type: 'room', description: 'Mayor\'s office'})

CREATE (:Location {id: 'loc_salle_fetes', name: 'Salle des Fêtes',
        type: 'building', description: 'Festival Hall'})

CREATE (:Location {id: 'loc_serre', name: 'Éco-ferme - Serre',
        type: 'outdoor', description: 'Greenhouse at eco-farm'})

CREATE (:Location {id: 'loc_sdf_gym', name: 'SDF Gym',
        type: 'room', description: 'Gym at sports facility'})
```

✅ **Verify:** Run this to see your locations

```
MATCH (l:Location)
RETURN l.name, l.type
ORDER BY l.name
```

Step 2: Create Our Devices

Now let's add the IoT devices

```
// Create Device nodes
CREATE (:Device {
  id: 'dev_esp8266_maire',
  name: 'BADEVEL | bureau du maire | ESP8266',
  deviceType: 'ESP8266',
  state: 'Active',
  createdTime: datetime('2025-01-29T15:01:46')
})

CREATE (:Device {
  id: 'dev_dragino_serre',
  name: 'BADEVEL | SERRE | DRAGINO',
  deviceType: 'DRAGINO',
  state: 'Active',
  createdTime: datetime('2025-10-03T13:53:34')
})

CREATE (:Device {
  id: 'dev_dragino_fetes',
  name: 'BADEVEL | SALLE DES FETES | DRAGINO TEMP',
  deviceType: 'DRAGINO',
  state: 'Active',
  createdTime: datetime('2025-10-03T13:01:42')
})
```

💡 **Notice:** We use `datetime()` for timestamps!

Step 3: Create Sensor Types

Create reusable sensor types

```
// Create Sensor nodes (reusable types)
CREATE (:Sensor {id: 'sensor_temp', type: 'temperature', unit: '°C'})
CREATE (:Sensor {id: 'sensor_hum', type: 'humidity', unit: '%'})
CREATE (:Sensor {id: 'sensor_press', type: 'pressure', unit: 'hPa'})
CREATE (:Sensor {id: 'sensor_lum', type: 'luminosity', unit: 'lux'})
```

🤔 *Why create sensor "types" instead of individual sensors?*

Because multiple devices can have the SAME TYPE of sensor!

This lets us ask: "Find ALL temperature sensors across ALL devices"

✅ Verify:

```
MATCH (s:Sensor)
RETURN s.type, s.unit
```

Step 4: Create Customer & Assets

```
// Create Customer node
CREATE (:Customer {
  id: 'cust_commune_badevel',
  name: 'Commune de Badevel'
})

// Create Asset nodes
CREATE (:Asset {
  id: 'asset_mairie',
  name: '[Badevel] Mairie',
  assetType: 'Building',
  label: 'Mairie de Badevel'
})

CREATE (:Asset {
  id: 'asset_vehicule',
  name: '[Badevel] Véhicule',
  assetType: 'Vehicle',
  label: 'Véhicule de Badevel'
})
```

So far we have:

✅ 5 Locations | ✅ 3 Devices | ✅ 4 Sensors | ✅ 1 Customer | ✅ 2 Assets

Now: The Magic Happens! 🪄

We have nodes... but they're lonely!

Time to connect them with **RELATIONSHIPS**

Remember:

- Nodes = Things (nouns)
- Relationships = Connections (verbs)

Relationships are what make graphs powerful!

Step 5: Location Hierarchy

Connect rooms to their buildings

```
// Bureau du maire is PART_OF Mairie
MATCH (room:Location {name: 'Bureau du maire'}),
      (building:Location {name: 'Mairie'})
CREATE (room)-[:PART_OF]->(building)

RETURN room.name AS Room, building.name AS Building
```

💡 ***The pattern:***

1. *`MATCH` finds the nodes we want to connect*
2. *`CREATE` creates the relationship between them*
3. *`RETURN` shows us what we just did*

✅ **Now you can query:** "What's in the Mairie?"

Step 6: Place Devices in Locations

Tell Neo4j WHERE each device is

```
// ESP8266 is LOCATED_IN Bureau du maire
MATCH (dev:Device {name: 'BADEVEL | bureau du maire | ESP8266'}),
      (loc:Location {name: 'Bureau du maire'})
CREATE (dev)-[:LOCATED_IN]->(loc)

// DRAGINO is LOCATED_IN Greenhouse
MATCH (dev:Device {name: 'BADEVEL | SERRE | DRAGINO'}),
      (loc:Location {name: 'Éco-ferme - Serre'})
CREATE (dev)-[:LOCATED_IN]->(loc)

// DRAGINO is LOCATED_IN Festival Hall
MATCH (dev:Device {name: 'BADEVEL | SALLE DES FETES | DRAGINO TEMP'}),
      (loc:Location {name: 'Salle des Fêtes'})
CREATE (dev)-[:LOCATED_IN]->(loc)
```

✅ Verify where devices are:

```
MATCH (d:Device)-[:LOCATED_IN]->(l:Location)
RETURN d.name AS Device, l.name AS Location
```

Step 7: Connect Devices to Sensors

Tell Neo4j WHAT each device can measure

```
// ESP8266 in Mayor's office HAS 4 sensors
MATCH (dev:Device {name: 'BADEVEL | bureau du maire | ESP8266'}),
      (temp:Sensor {type: 'temperature'}),
      (hum:Sensor {type: 'humidity'}),
      (press:Sensor {type: 'pressure'}),
      (lum:Sensor {type: 'luminosity'})
CREATE (dev)-[:HAS_SENSOR]->(temp)
CREATE (dev)-[:HAS_SENSOR]->(hum)
CREATE (dev)-[:HAS_SENSOR]->(press)
CREATE (dev)-[:HAS_SENSOR]->(lum)

// DRAGINO in greenhouse HAS temperature sensor
MATCH (dev:Device {name: 'BADEVEL | SERRE | DRAGINO'}),
      (temp:Sensor {type: 'temperature'})
CREATE (dev)-[:HAS_SENSOR]->(temp)

// DRAGINO in festival hall HAS temperature sensor
MATCH (dev:Device {name: 'BADEVEL | SALLE DES FETES | DRAGINO TEMP'}),
      (temp:Sensor {type: 'temperature'})
CREATE (dev)-[:HAS_SENSOR]->(temp)
```

Check Our Sensor Setup

> ## Let's see what we built!

```
// Show all devices and their sensors
MATCH (d:Device)-[:HAS_SENSOR]->(s:Sensor)
RETURN d.name AS Device,
       collect(s.type) AS Sensors
ORDER BY d.name
```

Expected Result:

- ESP8266: [temperature, humidity, pressure, luminosity]
- DRAGINO Greenhouse: [temperature]
- DRAGINO Festival Hall: [temperature]

> 💡 ***Notice:*** `collect()` *groups multiple values into a list!*

Step 8: Assign Ownership

All devices belong to Commune de Badevel

```
// Connect all BADEVEL devices to customer
MATCH (dev:Device), (cust:Customer {name: 'Commune de Badevel'})
WHERE dev.name STARTS WITH 'BADEVEL'
CREATE (dev)-[:BELONGS_TO]->(cust)

// Also assign assets
MATCH (asset:Asset), (cust:Customer {name: 'Commune de Badevel'})
WHERE asset.name STARTS WITH '[Badevel]'
CREATE (asset)-[:BELONGS_TO]->(cust)
```

💡 ***Smart pattern:*** *We use `STARTS WITH` to match multiple devices at once!*

This creates relationships for ALL devices whose names start with 'BADEVEL'

Step 9: Asset Monitoring Link

Connect assets to the devices that monitor them

```
// Mairie asset is monitored by the device in Mayor's office
MATCH (asset:Asset {name: '[Badevel] Mairie'}),
      (dev:Device {name: 'BADEVEL | bureau du maire | ESP8266'})
CREATE (asset)-[:RELATED_TO {relationship: 'monitors'}]->(dev)

RETURN asset.name AS Asset, dev.name AS MonitoringDevice
```

✅ **Notice:** Relationships can have properties too!

We added `{relationship: 'monitors'}` to explain the connection type

Step 10: See the Full Graph! 🎉

Moment of truth!

Let's visualize everything we built

```
// Show the entire knowledge graph
MATCH (n)
RETURN n
LIMIT 25
```

In Neo4j Browser, you should see:

- 🔵 Blue nodes for Locations
- 🟢 Green nodes for Devices
- 🟡 Yellow nodes for Sensors
- 🔴 Red nodes for Customer & Assets
- ↗️ Arrows showing all relationships

Click and drag to explore! 🎨

Chapter 4

Asking Questions

❓ 🔍 💬

The Power of Queries

Why We Built This Graph

Remember Badevel's problem?

"We have data but can't answer questions!"

Now we can answer ANYTHING! 🚀

Questions we'll answer today:

1. Simple: "Show me all locations"
2. Medium: "What sensors are in the Mayor's office?"
3. Advanced: "Find all temperature sensors and where they are"
4. Complex: "Show complete monitoring for Mairie building"

Query 1: All Locations

Question: "Show me all locations in Badevel"

Difficulty: ⭐ Simple

```
MATCH (l:Location)
RETURN l.name AS Location, l.type AS Type
ORDER BY l.name
```

Result:

Location	Type
Bureau du maire	room
Éco-ferme - Serre	outdoor
Mairie	building

Query 2: Find Active Devices

Question: "Show me only active devices"

Difficulty: ⭐ Simple

```
MATCH (d:Device)
WHERE d.state = 'Active'
RETURN d.name AS DeviceName, d.deviceType AS Type
ORDER BY d.name
```

💡 ***The WHERE clause:***

Acts like a filter - only returns devices where `state = 'Active'`

You can use: `=`, `<`, `>`, `AND`, `OR`

Query 3: Count Sensor Types

Question: "How many of each sensor type do we have?"

Difficulty: ⭐⭐ Medium

```
MATCH (d:Device)-[:HAS_SENSOR]->(s:Sensor)
RETURN s.type AS SensorType, count(*) AS Count
ORDER BY Count DESC
```

Result:

SensorType	Count
temperature	3
humidity	1

💡 ***Notice:*** *We traverse a relationship and aggregate!*

`count(*)` *counts how many times each sensor type appears*

Query 4: Device Location

Question: "Where is the ESP8266 device located?"

Difficulty: ⭐⭐ Medium

```
MATCH (d:Device {name: 'BADEVEL | bureau du maire | ESP8266'})
      -[:LOCATED_IN]->(l:Location)
RETURN d.name AS Device, l.name AS Location, l.type AS Type
```

Result:

Device: "BADEVEL | bureau du maire | ESP8266"

Location: "Bureau du maire"

Type: "room"

💡 ***The pattern:*** `(Device)-[:LOCATED_IN]->(Location)`

Reads like: "Device is LOCATED IN Location"

The arrow shows direction!

Query 5: Devices in a Location

Question: "What devices are in the Mayor's office?"

Difficulty: ⭐⭐ Medium

```
MATCH (d:Device)-[:LOCATED_IN]->(l:Location {name: 'Bureau du maire'})
RETURN d.name AS Device, d.state AS State
```

💡 ***Same relationship, different direction!***

Query 4: Started from Device → found Location

Query 5: Started from Location → found Devices

Graphs let you traverse relationships in BOTH directions! 🔄

Query 6: Sensors in a Location

Question: "What sensors are in the Mayor's office?"

Difficulty: ⭐⭐⭐ Advanced

```
MATCH (l:Location {name: 'Bureau du maire'})
      <-[:LOCATED_IN]-(d:Device)
      -[:HAS_SENSOR]->(s:Sensor)
RETURN l.name AS Location,
       d.name AS Device,
       collect(s.type) AS Sensors
```

Result:

Location: "Bureau du maire"

Device: "BADEVEL | bureau du maire | ESP8266"

Sensors: [temperature, humidity, pressure, luminosity]

🎯 **This is the power of graphs!**

We chained TWO relationships in ONE query:

Location ← LOCATED_IN ← Device → HAS_SENSOR → Sensors

Query 7: All Temperature Sensors

Question: "Find ALL temperature sensors and their locations"

Difficulty: ⭐⭐⭐ Advanced

```
MATCH (s:Sensor {type: 'temperature'})
      <-[:HAS_SENSOR]-(d:Device)
      -[:LOCATED_IN]->(l:Location)
RETURN s.type AS SensorType,
       d.name AS Device,
       l.name AS Location
ORDER BY l.name
```

Result (3 rows):

SensorType	Device	Location
temperature	...ESP8266	Bureau du maire
temperature	...DRAGINO	Éco-ferme

Query 8: Building Hierarchy

Question: "Show all devices in Mairie (including rooms inside)"

Difficulty: ⭐⭐⭐⭐ Complex

```
MATCH (building:Location {name: 'Mairie'})
MATCH (d:Device)-[:LOCATED_IN]->(l:Location)
WHERE l = building OR (l)-[:PART_OF]->(building)
MATCH (d)-[:HAS_SENSOR]->(s:Sensor)
RETURN building.name AS Building,
       l.name AS Location,
       d.name AS Device,
       collect(s.type) AS Sensors
```

💡 ***Breaking it down:***

1. *Find the Mairie building*
2. *Find devices that are EITHER:*
   - *Directly in Mairie, OR*
   - *In a location that is PART_OF Mairie*
3. *Get their sensors*

This handles the hierarchy! 🏗️

Query 9: Customer's Devices

Question: "What does Commune de Badevel own?"

Difficulty: ⭐⭐⭐ Advanced

```
MATCH (c:Customer {name: 'Commune de Badevel'})
      <-[:BELONGS_TO]-(d:Device)
      -[:LOCATED_IN]->(l:Location)
RETURN c.name AS Customer,
       count(d) AS TotalDevices,
       collect(DISTINCT l.type) AS LocationTypes
```

Result:

Customer: "Commune de Badevel"

TotalDevices: 3

LocationTypes: [room, outdoor, building]

💡 **We used:** `collect(DISTINCT ...)` to get unique types!

Query 10: Shortest Path

Question: "How is a device connected to a customer?"

Difficulty: ⭐⭐⭐⭐ Complex

```
MATCH path = shortestPath(
  (d:Device {name: 'BADEVEL | bureau du maire | ESP8266'})
  -[*]-(c:Customer {name: 'Commune de Badevel'})
)
RETURN path
```

✨ **Magic!**

Neo4j finds the shortest path automatically!

Visualize it in Neo4j Browser to see the connection chain 🔗

💡 ***The `[*]` means:** "any number of relationships of any type"*

Neo4j will find the shortest way to connect these two nodes!

Chapter 5

Updating Our Graph

✏️ 🔄 🗑️

Graphs Change, Data Evolves

Real World = Change

Badevel's reality:

- 📍 Devices get moved to new locations
- 🔧 Devices need maintenance (go offline)
- ➕ New sensors are added
- ❌ Old equipment is removed

> **Our graph must reflect these changes!**

Update 1: Change Device State

Scenario: The greenhouse device needs maintenance

```
// Set device to Inactive
MATCH (d:Device {name: 'BADEVEL | SERRE | DRAGINO'})
SET d.state = 'Inactive'
RETURN d.name AS Device, d.state AS NewState
```

Result: State changed from "Active" → "Inactive"

💡 ***The pattern:***

1. `MATCH` finds the node | 2. `SET` updates the property | 3. `RETURN` shows the change

Update 1: Verify the Change

✅ **Always verify your updates!**

```
MATCH (d:Device {state: 'Inactive'})
RETURN d.name AS DeviceName, d.state AS State
```

This shows all inactive devices:

✅ BADEVEL | SERRE | DRAGINO - State: Inactive

💡 **Tip:** You can also check the specific device:

```
MATCH (d:Device {name: 'BADEVEL | SERRE | DRAGINO'})
RETURN d.name, d.state, d.deviceType
```

Update 2: Add New Property

Scenario: Add detailed description to a location

```
MATCH (l:Location {name: 'Salle des Fêtes'})
SET l.description = 'Main festival hall of Badevel, used for community events',
    l.capacity = 200,
    l.lastUpdated = datetime()
RETURN l.name, l.description, l.capacity
```

✅ **You can:**

- Update existing properties
- Add new properties
- Set multiple properties at once (comma-separated)

Delete 1: Remove a Relationship

Scenario: Remove humidity sensor from ESP8266

```
// Find and delete the relationship
MATCH (d:Device {name: 'BADEVEL | bureau du maire | ESP8266'})
      -[r:HAS_SENSOR]->(s:Sensor {type: 'humidity'})
DELETE r
RETURN d.name AS Device, 'Humidity sensor removed' AS Status
```

💡 ***Important:***

*We delete the **relationship** (`r`), not the sensor node!*

The sensor node still exists for other devices to use

✅ **Verify remaining sensors:**

```
MATCH (d:Device {name: 'BADEVEL | bureau du maire | ESP8266'})
      -[:HAS_SENSOR]->(s:Sensor)
RETURN collect(s.type) AS RemainingSensors
```

Delete 2: Remove a Device

⚠️ **Scenario:** Decommission the Festival Hall device

```
// DETACH DELETE removes node and all its relationships
MATCH (d:Device {name: 'BADEVEL | SALLE DES FETES | DRAGINO TEMP'})
DETACH DELETE d
```

⚠️ **DETACH DELETE:**

- Deletes the node
- Automatically deletes ALL relationships connected to it
- Cannot be undone!

✅ **Best practice:** Always inspect before deleting!

```
// See what will be deleted
MATCH (d:Device {name: 'BADEVEL | SALLE DES FETES | DRAGINO TEMP'})
      -[r]-(n)
RETURN d, r, n
```

Add New Monitoring Setup

Scenario: Add a new device to the school

```
// Step 1: Create the location
CREATE (l:Location {
  id: 'loc_ecole_maternelle',
  name: 'École Maternelle',
  type: 'building'
})

// Step 2: Create the device
WITH l
CREATE (d:Device {
  id: 'dev_esp8266_ecole',
  name: 'BADEVEL | École Maternelle | ESP8266',
  deviceType: 'ESP8266',
  state: 'Active',
  createdTime: datetime()
})

// Step 3: Connect device to location
CREATE (d)-[:LOCATED_IN]->(l)

// Step 4: Connect to existing sensors
WITH d
MATCH (temp:Sensor {type: 'temperature'}),
      (hum:Sensor {type: 'humidity'})
CREATE (d)-[:HAS_SENSOR]->(temp)
CREATE (d)-[:HAS_SENSOR]->(hum)

// Step 5: Assign to customer
WITH d
MATCH (c:Customer {name: 'Commune de Badevel'})
CREATE (d)-[:BELONGS_TO]->(c)
```

Chapter 6

Your Turn!

👨‍💻 🧑‍💻 👩‍💻

Hands-On Exercises

Exercise Set 1: Basic Creation

Exercise 1.1: Create a new location

Task: Create a location for "Secrétariat" (Secretary office)

- id: 'loc_secretariat'
- name: 'Secrétariat'
- type: 'room'
- description: 'Secretary office'

💡 ***Hint:*** *Use the CREATE statement with Location label*

Check your work with: `MATCH (l:Location {name: 'Secrétariat'}) RETURN l`

Exercise Set 1: Basic Creation

Exercise 1.2: Create and connect a device

Task: Create a device in the Secrétariat

- Name: "BADEVEL | Secrétariat | ESP8266"
- Device Type: ESP8266
- State: Active
- Then connect it to the Secrétariat location with LOCATED_IN

💡 ***Hints:***

1. *First CREATE the device*
2. *Then MATCH both device and location*
3. *Finally CREATE the relationship*

Exercise Set 2: Querying

Exercise 2.1: Find all buildings

Task: Write a query to find all locations of type 'building'

Return: location name and description

Exercise 2.2: Count devices by type

Task: Count how many devices of each deviceType exist

Return: device type and count, ordered by count (descending)

Exercise Set 3: Relationships

Exercise 3.1: Find sensors in Mayor's office

Task: Find all sensors in devices located in "Bureau du maire"

Return: location name, device name, list of sensor types

Hint: Chain LOCATED_IN and HAS_SENSOR relationships

Exercise 3.2: Find devices without sensors

Task: Find all devices that don't have any sensors

Hint: Use `WHERE NOT (d)-[:HAS_SENSOR]->()`

Exercise Set 4: Advanced

Challenge 1: Complete monitoring report

Task: Create a report showing:

- Total number of locations
- Total number of devices (active and inactive separately)
- Total number of unique sensor types deployed
- Most common sensor type

Hint: You'll need multiple queries or use UNION

Challenge 2: Sensor coverage analysis

Task: Find all locations that have temperature sensors and count how many

What We Accomplished

🎉 ✨ 🚀

Our Journey Today

From Dashboard to Knowledge Graph!

🎯 What We Started With

- Raw IoT dashboard
- 54 devices
- Limited querying
- Hard to answer questions

✅ What We Built

- Complete knowledge graph
- 5 node types
- 5 relationship types
- Unlimited querying power!

Now Badevel can answer ANY question about their IoT system! 🎯

Skills You Learned Today

✅ Knowledge Modeling

- Identifying entities (nodes) from raw data
- Defining relationships between entities
- Designing a complete graph model

✅ Neo4j & Cypher

- Creating nodes and relationships
- Querying with MATCH and WHERE
- Traversing relationships
- Updating and deleting data
- Aggregating with count(), collect()

✅ Graph Thinking

Where Can You Use This?

Knowledge graphs are everywhere!

🏭 Industrial IoT

Factory sensors, equipment monitoring, predictive maintenance

🏥 Healthcare

Patient records, treatment paths, drug interactions

📱 Social Networks

Friends, followers, recommendations

🛒 E-commerce

Products, customers, recommendations, supply chain

🚗 Smart Cities

Traffic, transportation, infrastructure

💰 Finance

Fraud detection, risk analysis, transactions

Next Steps

📚 Continue Learning

- Neo4j Graph Academy: graphacademy.neo4j.com
- Neo4j Docs: neo4j.com/docs
- Cypher Reference Guide

🔨 Practice Projects

- Expand Badevel graph with more data
- Add measurement nodes for time-series
- Create alert rules and notifications
- Model your own domain

⚡ Advanced Topics

- Graph algorithms (PageRank, community detection)
- App integration (Python, JavaScript drivers)
- Performance optimization & indexing

Key Takeaways 🎯

Remember these core concepts:

1. Graphs Model Relationships

When data is interconnected, graphs are natural

2. Cypher is Visual

`(Node)-[:RELATIONSHIP]->(Node)` reads like you speak

3. Traversal is Power

Chain relationships to answer complex questions

4. Data → Information → Knowledge

Structure + Context + Relationships = Knowledge

Exercise Solutions

📝 ✅ 💡

Complete Answers

Solution 1.1: Create Location

Task: Create "Secrétariat" location

```
CREATE (:Location {
  id: 'loc_secretariat',
  name: 'Secrétariat',
  type: 'room',
  description: 'Secretary office'
})
```

✅ **Verify:**

```
MATCH (l:Location {name: 'Secrétariat'})
RETURN l
```

Solution 1.2: Create and Connect Device

Task: Create device in Secrétariat and connect it

```
// Step 1: Create the device
CREATE (:Device {
  id: 'dev_esp8266_secretariat',
  name: 'BADEVEL | Secrétariat | ESP8266',
  deviceType: 'ESP8266',
  state: 'Active',
  createdTime: datetime()
})

// Step 2: Connect to location
MATCH (d:Device {id: 'dev_esp8266_secretariat'}),
      (l:Location {name: 'Secrétariat'})
CREATE (d)-[:LOCATED_IN]->(l)
RETURN d.name AS Device, l.name AS Location
```

Solution 2.1: Find All Buildings

Task: Find all locations of type 'building'

```
MATCH (l:Location {type: 'building'})
RETURN l.name AS BuildingName, l.description AS Description
ORDER BY l.name
```

Expected Result:

- École Maternelle
- Mairie
- Périscolaire
- Salle des Fêtes

Solution 2.2: Count Devices by Type

Task: Count devices by deviceType

```
MATCH (d:Device)
RETURN d.deviceType AS DeviceType,
       count(d) AS Count
ORDER BY Count DESC
```

Expected Result:

DeviceType	Count
ESP8266	2-3
DRAGINO	2

Solution 3.1: Sensors in Location

Task: Find all sensors in "Bureau du maire"

```
MATCH (l:Location {name: 'Bureau du maire'})
      <-[:LOCATED_IN]-(d:Device)
      -[:HAS_SENSOR]->(s:Sensor)
RETURN l.name AS Location,
       d.name AS Device,
       collect(s.type) AS Sensors
```

Result: [temperature, humidity, pressure, luminosity]

💡 ***Key:*** *Chain LOCATED_IN and HAS_SENSOR | Use collect() to group sensors*

Solution 3.2: Devices Without Sensors

Task: Find devices with no sensors

```
MATCH (d:Device)
WHERE NOT (d)-[:HAS_SENSOR]->()
RETURN d.name AS DeviceName,
       d.deviceType AS DeviceType
ORDER BY d.name
```

Alternative solution using OPTIONAL MATCH:

```
MATCH (d:Device)
OPTIONAL MATCH (d)-[:HAS_SENSOR]->(s:Sensor)
WITH d, count(s) AS sensorCount
WHERE sensorCount = 0
RETURN d.name AS DeviceName
```

Solution Challenge 1: Monitoring Report

Task: Complete monitoring statistics

```
// Total locations
MATCH (l:Location)
WITH count(l) AS totalLocations

// Device statistics
MATCH (d:Device)
WITH totalLocations,
     count(d) AS totalDevices,
     sum(CASE WHEN d.state = 'Active' THEN 1 ELSE 0 END) AS activeDevices,
     sum(CASE WHEN d.state = 'Inactive' THEN 1 ELSE 0 END) AS inactiveDevices

// Unique sensor types
MATCH (s:Sensor)<-[:HAS_SENSOR]-()
WITH totalLocations, totalDevices, activeDevices, inactiveDevices,
     count(DISTINCT s.type) AS uniqueSensorTypes

// Most common sensor
MATCH (s:Sensor)<-[:HAS_SENSOR]-()
WITH totalLocations, totalDevices, activeDevices, inactiveDevices,
     uniqueSensorTypes, s.type AS sensorType, count(*) AS sensorCount
ORDER BY sensorCount DESC
LIMIT 1

RETURN totalLocations, totalDevices, activeDevices, inactiveDevices,
       uniqueSensorTypes, sensorType AS mostCommonSensor, sensorCount
```

Solution Challenge 2: Sensor Coverage

Task: Count temperature sensors by location

```
MATCH (l:Location)<-[:LOCATED_IN]-(d:Device)
      -[:HAS_SENSOR]->(s:Sensor {type: 'temperature'})
RETURN l.name AS Location,
       l.type AS LocationType,
       count(s) AS TemperatureSensors,
       count(DISTINCT d) AS DevicesWithTemp
ORDER BY TemperatureSensors DESC, l.name
```

Shows: Which locations have temperature monitoring and how many sensors

Tips for Solving Exercises

🎯 Approach Strategy

- Start with simple MATCH to explore data
- Add WHERE filters one at a time
- Build complex queries by chaining relationships
- Always use RETURN to verify each step

✅ Debugging Tips

- Use LIMIT 5 when testing queries
- Check node existence with simple MATCH first
- Use EXPLAIN to understand execution
- Visualize results in Neo4j Browser

💡 Common Patterns

- `collect()` - Group values | `count()` - Count occurrences
- `NOT (pattern)` - Find without relationship
- `OPTIONAL MATCH` - Match if exists, null otherwise

Merci!

🙏 🎉 🚀

Questions?

joseph.azar@univ-fcomte.fr

TD Session 1 - Knowledge Graphs with Neo4j

CP58 UTBM Sevenans