

Recherche Géographique

Réalisé par :

HADOU Mohamed

Encadré par :

Mme Nora AHERRAHROU

TABLE DE MATIERE

01

Introduction

03

Choix technique

02

Objectifs

04

Démonstration

05

Conclusion



INTRODUCTION

01



INTRODUCTION

Avec l'essor des technologies de l'information et l'augmentation exponentielle des données géospatiales, la recherche géographique est devenue un domaine crucial pour de nombreuses applications, allant de la navigation et la logistique à la gestion des ressources naturelles et à l'analyse de marché. Pour répondre à ces besoins croissants, les bases de données traditionnelles montrent souvent leurs limites en termes de performance et de flexibilité. C'est dans ce contexte que Neo4j, une base de données orientée graphes, se distingue par ses capacités uniques à gérer et interroger des données géospatiales de manière efficace.



Objectifs

02

OBJECTIF :

L'objectif de notre travail est de localiser la pharmacie la plus proche en déterminant la distance la plus courte entre le point de départ et les différentes pharmacies disponibles, en utilisant les capacités de calcul de distances de Neo4j.



**CHOIX
TECHNIQUE**

03

TECHNOLOGIES



OpenStreetMap

OpenStreetMap (OSM) est un projet collaboratif pour créer une carte mondiale libre et modifiable. Les données sont gratuites, accessibles à tous, et souvent très détaillées.



Neo4j

Neo4j est une base de données graphe, conçue pour stocker et interroger des données hautement connectées. Neo4j utilise une structure de graphe, composée de nœuds (représentant les entités) et de relations (représentant les connexions entre ces entités).





Démonstration

04

Visualisation du Dataset

```
1 LOAD CSV WITH HEADERS FROM 'file:///map.csv' AS row
2 RETURN row
3 LIMIT 1;
```

Table

Text

Code

row

```
"power": null,
"repair": null,
"addr:suburb": null,
"name:kab": null,
"denomination": null,
"consulate": null,
"restriction": null,
"clothes": null,
"alt_name:ar": null,
"leaf_cycle": null,
"name:ary": null,
"name:arz": null,
"electrified": null,
"short_name": null,
"network:wikidata": null,
"WKT": "POINT (-5.0133611 34.0313163)",
```

```
LOAD CSV WITH HEADERS FROM 'file:///map.csv' AS row
WITH row
WHERE row.amenity = 'pharmacy'
RETURN row
LIMIT 10;
```

row

```
"name:pro": null,
"name": "Pharmacie Centrale المركزية الصيدلية",
"name:kk-Arab": null,
"name:ur": null,
"name:ar": null,
"alt_name:fr": null,
"lanes": null,
"population": null,
"brand:wikipedia:es": null,
"name:cs": null,
"local_ref": null,
"substation": null,
"crossing_ref": null,
"layer": null,
"internet_access:fee": null,
"lanes:forward": null,
"operator:wikipedia:ar": null,
```

```

1 LOAD CSV WITH HEADERS FROM 'file:///map.csv' AS row
2 WITH row
3 WHERE row.amenity = 'pharmacy' AND row.WKT IS NOT NULL
4 WITH row, apoc.text.replace(row.WKT, 'POINT \\(|\\)', '') AS cleanedWKT
5 WITH row, apoc.text.split(cleanedWKT, ' ') AS coordinates
6 WHERE size(coordinates) = 2
7 CREATE (p:Pharmacy {
8     id: row.osm_id,
9     name: row.name,
10    address: row.addr:street,
11    latitude: toFloat(trim(coordinates[1])), // Extraction de la latitude
12    longitude: toFloat(trim(coordinates[0])), // Extraction de la longitude
13    phone: row.phone,
14    city: row.addr:city,
15    postcode: row.addr:postcode
16 })
17 RETURN p.latitude AS Latitude, p.longitude AS Longitude
18 LIMIT 10;

```

| | Latitude | Longitude |
|---|------------|------------|
| 1 | 34.036761 | -4.9978456 |
| 2 | 34.0400364 | -5.0057261 |
| 3 | 34.0394593 | -5.0015078 |
| 4 | 34.0296079 | -5.0105717 |
| 5 | 34.033564 | -5.0069404 |

Added 57 labels, created 57 nodes, set 276 properties, started streaming 10 records after 581 ms and completed after 1146 ms.

```
Entrée [18]: import csv

# Fonction pour créer des relations entre les pharmacies consécutives
def create_relations_between_pharmacies(csv_file):
    pharmacies = []
    with open(csv_file, mode='r', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        previous_pharmacy = None
        for row in reader:
            # Vérifier si la ligne contient une pharmacie
            if row.get('amenity') == 'pharmacy':
                current_pharmacy = row['osm_id']
                # Si c'est la première pharmacie, passer à la suivante
                if previous_pharmacy is None:
                    previous_pharmacy = current_pharmacy
                    continue
                # Ajouter la relation entre la pharmacie précédente et la pharmacie actuelle
                relationship = {'source_osm_id': previous_pharmacy, 'target_osm_id': current_pharmacy}
                pharmacies.append(relationship)
                # Mettre à jour la pharmacie précédente pour la prochaine itération
                previous_pharmacy = current_pharmacy
    return pharmacies

# Fonction pour écrire les relations dans un fichier CSV
def write_relationships_to_csv(output_csv_file, relationships):
    with open(output_csv_file, mode='w', newline='', encoding='utf-8') as file:
        writer = csv.DictWriter(file, fieldnames=['source_osm_id', 'target_osm_id'])
        writer.writeheader()
        for relationship in relationships:
            writer.writerow(relationship)

# Nom du fichier CSV
csv_file = 'map.csv'

# Appeler la fonction pour créer les relations entre les pharmacies consécutives
relationships = create_relations_between_pharmacies(csv_file)

# Nom du fichier CSV de sortie pour les relations
output_csv_file = 'pharmacy_relationships1.csv'

# Écrire les relations dans un fichier CSV
write_relationships_to_csv(output_csv_file, relationships)
```

```
1 LOAD CSV WITH HEADERS FROM 'file:///pharmacy_relationships1.csv' AS row
2 RETURN row
3 LIMIT 1;
4
```

Table

row

1

```
{
  "target_osm_id": "2328748922",
  "source_osm_id": "1866502248"
}
```

Text

Code

Started streaming 1 records after 13 ms and completed after 53 ms.

```
1 LOAD CSV WITH HEADERS FROM 'file:///pharmacy_relationships1.csv' AS row
2 MATCH (source:Pharmacy {id: row.source_osm_id})
3 MATCH (target:Pharmacy {id: row.target_osm_id})
4 CREATE (source)-[:CONNECTED_TO]->(target)
```

Table

Created 56 relationships, completed after 299 ms.

Code

Created 56 relationships, completed after 299 ms.

```
neo4j$ MATCH (p:Pharmacy)-[r:CONNECTED_TO]→(p2:Pharmacy) RETURN p, r, p2
```

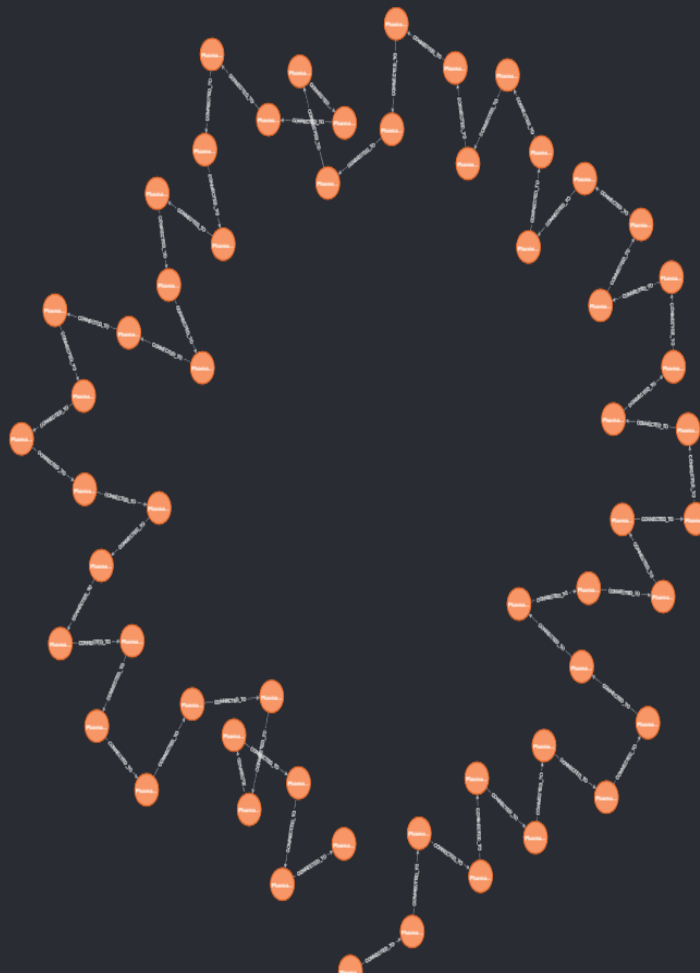


Graph

Table

Text

Code



Overview

Node labels

* (57)

Pharmacy (57)

Relationship types

* (56)

CONNECTED_TO (56)

Displaying 57 nodes, 56 relationships.



```
neo4j$ CREATE (:Position {latitude: 34.0363086, longitude: -4.9967841})
```



Table



Code

Added 1 label, created 1 node, set 2 properties, completed after 24 ms.

Added 1 label, created 1 node, set 2 properties, completed after 24 ms.

```
1 MATCH (position:Position), (pharmacy:Pharmacy)
2 CREATE (position)-[:CONNECTED_TO]-(pharmacy)
```



Table



Warn



Code

Created 57 relationships, completed after 20 ms.

Created 57 relationships, completed after 20 ms.

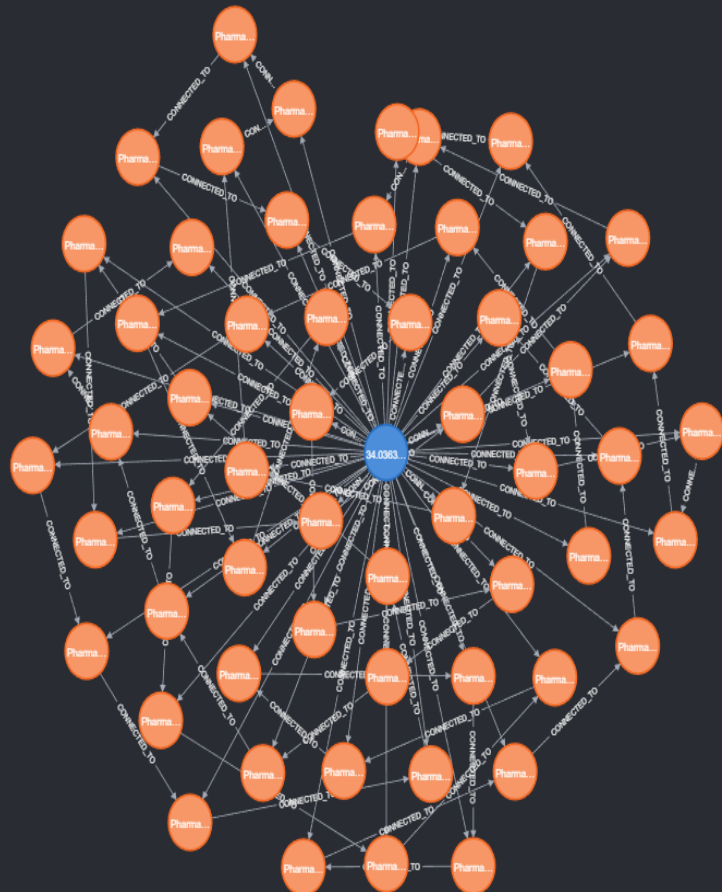
```
neo4j$ MATCH (p1:Position)-[r:CONNECTED_TO]→(p:Pharmacy) RETURN p1, r, p
```

Graph

Table

Text

Code



Overview

Node labels

* (58) **Position (1)**

Pharmacy (57)

Relationship types

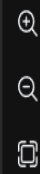
* (113)

CONNECTED_TO (113)

Displaying 58 nodes, 57 relationships.

Copy Ctrl+C

Paste Ctrl+V




```
// Ensure the Position node already exists with the specified coordinates
MATCH (start:Position {latitude: 34.0363086, longitude: -4.9967841})
```

```
WITH start
MATCH (pharmacy:Pharmacy)
CALL apoc.algo.dijkstra(start, pharmacy, 'CONNECTED_TO', 'distance') YIELD path, weight
RETURN path, last(nodes(path)).distance AS distance
ORDER BY distance ASC
LIMIT 1
```




| start | pharmacy | distance |
|--|---|--------------------|
| 1 | | |
| { "identity": 57, "labels": ["Position"], "properties": { "latitude": 34.0363086, "longitude": -4.9967841 }, "elementId": "57" } | { "identity": 0, "labels": ["Pharmacy"], "properties": { "phone": "0535622504", "latitude": 34.036761, "name": "Pharmacie Centrale الصيدلية المركزية", "id": "1866502248", "longitude": -4.9978456 }, "elementId": "0" } | 110.11301152066389 |



CONNECTED_TO



 Use Ctrl or Shift + scroll to zoom
[Don't show again](#)

Overview

Node labels

* (2) Position (1)

Pharmacy (1)

Relationship types

* (1)

CONNECTED_TO (1)


Displaying 2 nodes, 0 relationships.





Conclusion

05



Conclusion :

En conclusion, Neo4j est une base de données graphe puissante et flexible, idéale pour gérer des données hautement connectées. Son modèle de graphe, son langage Cypher, et ses performances optimisées en font un outil essentiel pour des applications comme les réseaux sociaux, la détection de fraudes et les moteurs de recommandation. Adopter Neo4j permet de mieux exploiter les connexions entre les données pour résoudre des problèmes complexes.



**Merci pour votre
attention**