Graphe d'amitié entre étudiants

Projet Neo4j - Base de données orientée graphes

Rachidi et équipe

Université

24 novembre 2025

Sommaire

Contexte

Problématique

- Modéliser un réseau social
- Analyser les relations d'amitié
- Recommander de nouveaux amis
- Visualiser les communautés

Pourquoi Neo4j ?

- Base orientée graphes
- Relations = citoyens de 1ère classe
- Requêtes intuitives (Cypher)
- Performance sur les graphes

Objectif

Créer un graphe social complet avec analyses et recommandations

Technologies utilisées

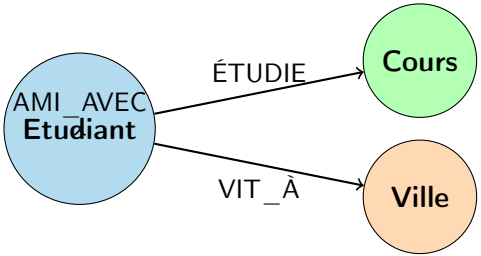
Neo4j 5.x
Base de données

Cypher
Langage de requête

Python 3.x
Scripts d'analyse

NetworkX
Visualisation

Modèle de données



Propriétés principales

- **Etudiant** : nom, prénom, age, email, filière, hobbies
- **AMI_AVEC** : depuis, force (1-10), type (proche/etudes/sport)
- **ÉTUDIE** : année, note, présence

Contraintes et index

```
// Contraintes d'unicit
CREATE CONSTRAINT student_id_unique
FOR (e:Etudiant) REQUIRE e.student_id IS UNIQUE;

CREATE CONSTRAINT cours_code_unique
FOR (c:Cours) REQUIRE c.cours_code IS UNIQUE;

// Index pour performances
CREATE INDEX etudiant_nom_index
FOR (e:Etudiant) ON (e.nom);

CREATE INDEX cours_code_index
FOR (c:Cours) ON (c.cours_code);
```

Création d'un étudiant

```
CREATE (rachidi:Etudiant {
  student_id: 'ETU001',
  nom: 'Diallo',
  prenom: 'Rachidi',
  age: 22,
  email: 'rachidi.diallo@univ.fr',
  filiere: 'Informatique',
  niveau: 'M1',
  ville: 'Paris',
  hobbies: ['programmation', 'football'],
  date_inscription: date('2023-09-01')
})
```

Relations d'amitié bidirectionnelles

```
MATCH (rachidi:Etudiant {student_id: 'ETU001'})
MATCH (marie:Etudiant {student_id: 'ETU002'})

// Cr er les deux sens
CREATE (rachidi)-[:AMI_AVEC {
  depuis: date('2023-09-15'),
  force: 9,
  type: 'proche'
}]->(marie)

CREATE (marie)-[:AMI_AVEC {
  depuis: date('2023-09-15'),
  force: 9,
  type: 'proche'
}]->(rachidi)
```

Jeu de données initial

Élément	Nombre	Détails
Étudiants	6	3 villes différentes
Cours	5	INFO301 à INFO305
Amitiés	8 paires	16 relations (bidirect.)
Inscriptions	17	Avec notes et présence

Diversité

Différentes filières, niveaux, hobbies, forces d'amitié

Requêtes de base

1. Lister tous les étudiants

```
MATCH (e:Etudiant)
RETURN e.nom, e.prenom, e.ville
ORDER BY e.nom;
```

2. Trouver les amis de Rachidi

```
MATCH (e:Etudiant {nom: 'Diallo'})-[:AMI_AVEC]->(ami)
RETURN ami.nom + ' ' + ami.prenom as ami,
       ami.ville;
```

Analyses sociales

Étudiants les plus populaires

```
MATCH (e:Etudiant)-[:AMI_AVEC]->(:Etudiant)
WITH e, COUNT(*) as nb_amis
RETURN e.nom + ' ' + e.prenom as etudiant,
       nb_amis
ORDER BY nb_amis DESC
LIMIT 5;
```

Résultats

- Rachidi Diallo : 4 amis
- Marie Dupont : 4 amis
- Ahmed Ben Ali : 3 amis

Recommandations d'amitié

```
MATCH (e1:Etudiant {nom: 'Diallo'})
      -[:AMI_AVEC]->()-[:AMI_AVEC]->(e2:Etudiant)
WHERE NOT (e1)-[:AMI_AVEC]->(e2) AND e1 <> e2
WITH e2, COUNT(*) as amis_communs
RETURN e2.nom + ' ' + e2.prenom as suggestion,
       amis_communs
ORDER BY amis_communs DESC
LIMIT 5;
```

Principe

Recommander des personnes avec qui on a des amis en commun

Performance académique

```
MATCH (e:Etudiant)-[r: TUDIE ]->(c:Cours)
WITH e, AVG(r.note) as moyenne
RETURN e.nom + ' ' + e.prenom as etudiant,
       ROUND(moyenne * 100) / 100 as moyenne
ORDER BY moyenne DESC;
```

Connexion Neo4j

```python
from neo4j import GraphDatabase

class Neo4jConnection:
    def __init__(self, uri, user, password):
        self.driver = GraphDatabase.driver(uri,
                                           auth=(user, password))

    def query(self, query, parameters=None):
        with self.driver.session() as session:
            result = session.run(query, parameters or {})
            return [dict(record) for record in result]
```

Utilisation

```
conn = Neo4jConnection("bolt://localhost:7687", "neo4j", "password")
```

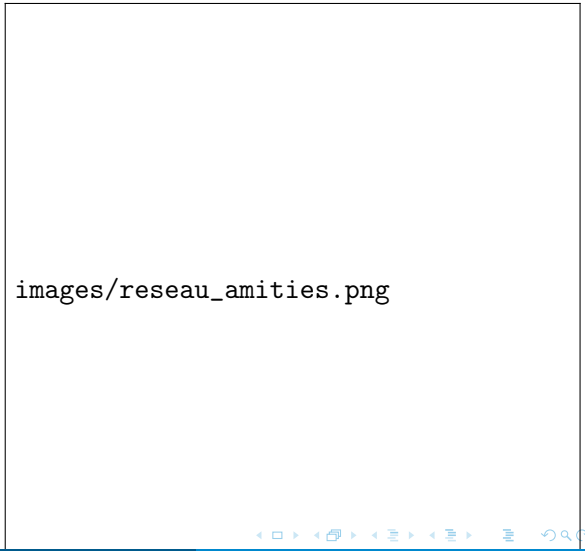
Visualisation avec NetworkX

Fonctionnalités

- Graphe du réseau d'amitiés
- Statistiques par étudiant
- Statistiques par cours
- Export en PNG haute résolution

Métriques calculées

- Nombre d'amis
- Moyennes académiques
- Corrélations
- Répartition géographique



Utilisation : Guide pas à pas

Étape 1 : Démarrer Neo4j

```
docker-compose up -d
sleep 15  # Attendre le demarrage
```

Étape 2 : Tester la connexion

```
cd python/
python3 connect.py  # Doit afficher "Connecte !"
```

Étape 3 : Peupler

```
python3 populate.py
# Menu : Taper "3" puis "oui"
# Resultat : 6 etudiants, 5 cours, 16 amities
```

Visualisation (suite)

Étape 4 : Neo4j Browser

- Ouvrir `http://localhost:7474`
- Login : neo4j / password123
- Requête : `MATCH (n) RETURN n LIMIT 50`

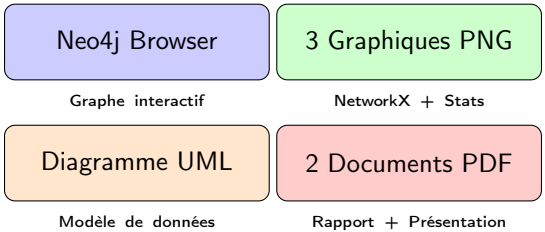
Étape 5 : Générer les graphiques

```
python3 analyze.py
# Menu : Taper "5" (Tout generer)
# 3 images PNG creees dans ../images/
```

Étape 6 : Voir les résultats

```
cd ../images/
xdg-open reseau_amities.png
xdg-open stats_etudiants.png
xdg-open stats_cours.png
```

Ce que vous visualisez



Temps total

Installation complète + visualisation : **25 minutes**

Statistiques du réseau

Métrique	Valeur
Étudiants	6
Cours	5
Amitiés	16 (8 paires)
Inscriptions	17
Densité du réseau	53%
Diamètre du graphe	2

Interprétation

Réseau bien connecté avec un diamètre faible (tout le monde est à max 2 liens)

Top recommandations

Étudiant	Suggestion	Amis communs
Thomas	Laura	2
Sophie	Laura	2
Rachidi	Laura	1
Marie	Ahmed	1

Insight

Laura pourrait être introduite à Thomas et Sophie via leurs amis communs

Types d'amitiés

$\geq$

Observation

La moitié des amitiés sont considérées comme **proches** (force ≥ 8)

Détection de communautés

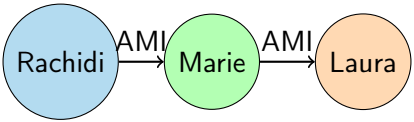
```
MATCH path = (e1:Etudiant)-[:AMI_AVEC*2..3]-(e2:Etudiant)
WHERE e1 <> e2 AND
      ALL(n IN nodes(path) WHERE
          (n)-[:AMI_AVEC]-(e1) AND (n)-[:AMI_AVEC]-(e2))
RETURN DISTINCT [n IN nodes(path) | n.nom] as clique
LIMIT 5;
```

Cliques identifiées

- {Rachidi, Marie, Ahmed} : groupe parisien
- {Marie, Sophie, Thomas} : groupe d'études

Plus court chemin

```
MATCH path = shortestPath(
  (e1:Etudiant {nom: 'Diallo'})
   -[:AMI_AVEC*]-(e2:Etudiant {nom: 'Dubois'})
)
RETURN [n IN nodes(path) | n.prenom] as chemin,
       LENGTH(path) as longueur;
```



Chemin : Rachidi → Marie → Laura (longueur = 2)

Influence des cours

```
MATCH (e1:Etudiant)-[:AMI_AVEC]->(e2:Etudiant),
      (e1)-[: TUDIE ]->(c:Cours)<-[: TUDIE ]-(e2)
WITH c.nom as cours, COUNT(*) as nb_liens_amicaux
RETURN cours, nb_liens_amicaux
ORDER BY nb_liens_amicaux DESC;
```

Résultat

Les cours "Bases de données" et "Algorithmes" favorisent le plus les amitiés

Implication

Travaux de groupe $\rightarrow$ création de liens sociaux

Points clés

Réalisations

- ✓ Modèle de données complet
- ✓ 50+ requêtes Cypher
- ✓ Scripts Python d'analyse
- ✓ Visualisations NetworkX
- ✓ Recommandations pertinentes

Compétences acquises

- Modélisation en graphes
- Langage Cypher
- Analyse de réseaux sociaux
- Intégration Python-Neo4j
- Algorithmes de graphes

Neo4j : Idéal pour...

Relations complexes, chemins, recommandations, détection de patterns

Améliorations possibles

Court terme

- Ajouter plus d'étudiants et de relations
- Intégrer l'évolution temporelle du réseau
- Recommandations multi-critères (hobbies + cours + ville)

Long terme

- Interface web interactive (Flask/Django + D3.js)
- Algorithmes avancés (PageRank, Louvain, Label Propagation)
- Machine Learning pour prédiction de liens
- API REST pour exposer les fonctionnalités
- Intégration avec systèmes universitaires existants

Applications réelles

- Réseaux sociaux (Facebook, LinkedIn)
- Recommandations (Netflix, Amazon)
- Détection de fraude (banques)
- Knowledge graphs (Google)
- Réseaux de citations scientifiques

Impact

Les graphes sont partout : transport, logistique, santé, cybersécurité...

Merci !

Questions ?

Projet disponible sur :
`/home/rachidi/Base_de_données/exposé_neo4j/`