Projet Neo4j

Graphe d'amitié entre étudiants

Rachidi et équipe

24 novembre 2025

Résumé

Ce projet présente la conception et l'implémentation d'un graphe social modélisant les relations d'amitié entre étudiants en utilisant la base de données orientée graphes Neo4j. Le projet comprend la modélisation des données, des requêtes Cypher pour analyser le réseau social, des scripts Python pour la visualisation, et une analyse complète des communautés et des recommandations d'amitié.

Table des matières

1 Introduction

1.1 Contexte

Les réseaux sociaux sont omniprésents dans notre vie quotidienne. Comprendre et analyser ces réseaux nécessite des outils adaptés capables de gérer les relations complexes entre individus. Les bases de données orientées graphes comme Neo4j sont particulièrement adaptées pour modéliser et interroger ces structures de données.

Pourquoi ce projet ? Imaginez que vous voulez savoir :

— Qui sont les étudiants les plus populaires dans votre classe ?

— Comment recommander de nouveaux amis à quelqu'un ?

— Quel est le chemin le plus court entre deux personnes ?

— Y a-t-il des groupes d'amis qui ne se mélangent pas ?

Avec une base de données classique (MySQL, PostgreSQL), répondre à ces questions serait très compliqué. Avec Neo4j, c'est naturel et rapide !

Exemple concret

Dans Facebook, quand vous voyez *"Vous connaissez peut-être Marie"*, c'est un algorithme de graphe qui détecte que vous avez 5 amis en commun avec Marie. Notre projet fait exactement la même chose !

1.2 Objectifs du projet

Ce projet vise à :

— Modéliser un réseau social d'étudiants avec leurs relations d'amitié

— Créer une base de données Neo4j avec des contraintes et des index appropriés

— Développer des requêtes Cypher pour analyser le graphe social

— Implémenter des scripts Python pour la visualisation et l'analyse

— Proposer des recommandations d'amitié basées sur les connexions existantes

1.3 Technologies utilisées

— **Neo4j** : Base de données orientée graphes (version 5.x recommandée)

— **Cypher** : Langage de requête pour Neo4j

— **Python 3.x** : Scripts d'analyse et visualisation

— **NetworkX** : Bibliothèque Python pour l'analyse de graphes

— **Matplotlib/Seaborn** : Visualisation des données

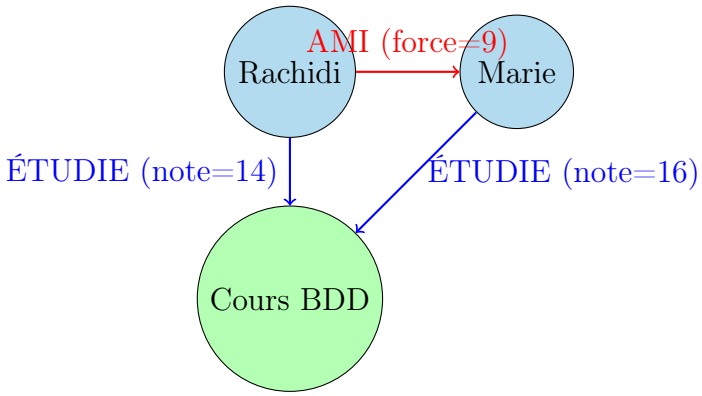
2 Modélisation des données

2.1 Modèle conceptuel

Qu'est-ce qu'un graphe ? Un graphe est comme une carte avec des villes (nœuds) reliées par des routes (relations). Dans notre cas :

— Les **villes** = les étudiants, cours, villes

— Les **routes** = les amitiés, inscriptions aux cours



Notre graphe social contient trois types de nœuds principaux :

Nœuds (Nodes)

— **Etudiant** : Représente un étudiant avec ses informations personnelles

— **Cours** : Représente un cours académique

— **Ville** : Représente la ville de résidence

Relations (Relationships)

— **AMI_AVEC** : Relation d'amitié entre deux étudiants (bidirectionnelle)

— **ÉTUDIE** : Relation entre un étudiant et un cours

— **VIT_À** : Relation entre un étudiant et sa ville

2.2 Propriétés des nœuds

2.2.1 Nœud Etudiant

Propriété	Type	Description
student_id	String	Identifiant unique (UNIQUE)
nom	String	Nom de famille (INDEX)
prenom	String	Prénom
age	Integer	Âge de l'étudiant
email	String	Email universitaire (UNIQUE)
telephone	String	Numéro de téléphone
filiere	String	Filière d'études (INDEX)
niveau	String	Niveau d'études (L1, L2, M1, M2)
hobbies	List<String>	Liste des loisirs
date_inscription	Date	Date d'inscription

2.2.2 Nœud Cours

Propriété	Type	Description
cours_code	String	Code unique (UNIQUE, INDEX)
nom	String	Nom du cours
credits	Integer	Nombre de crédits ECTS
semestre	String	Semestre (S1, S2)
enseignant	String	Nom de l'enseignant

2.3 Propriétés des relations

2.3.1 Relation AMI_AVEC

- `depuis` : Date du début de l'amitié (ex : 2023-09-15)
- `force` : Intensité de l'amitié (échelle 1-10)
  - 1-3 : Simple connaissance
  - 4-6 : Bon ami
  - 7-8 : Ami proche
  - 9-10 : Meilleur ami
- `type` : Type d'amitié
  - *proche* : Amitié personnelle profonde
  - *etudes* : Camarade de classe/projet
  - *sport* : Coéquipier sportif
  - *loisirs* : Activités communes

Pourquoi bidirectionnel ?

Dans notre modèle, si Rachidi est ami avec Marie, alors Marie est aussi amie avec Rachidi. On crée donc DEUX relations (une dans chaque sens) pour simplifier les requêtes. Ainsi, on peut facilement trouver tous les amis sans se soucier du sens.

2.3.2 Relation ÉTUDIE

- `annee` : Année universitaire
- `note` : Note obtenue (0-20)
- `presence` : Taux de présence (0-100%)

3 Implémentation

3.1 Création du schéma

Le schéma de la base de données est défini avec des contraintes d'unicité et des index pour optimiser les performances :

```
CREATE CONSTRAINT student_id_unique IF NOT EXISTS
FOR (e:Etudiant) REQUIRE e.student_id IS UNIQUE;

CREATE CONSTRAINT cours_code_unique IF NOT EXISTS
FOR (c:Cours) REQUIRE c.cours_code IS UNIQUE;

CREATE CONSTRAINT student_email_unique IF NOT EXISTS
FOR (e:Etudiant) REQUIRE e.email IS UNIQUE;

// Index pour recherches rapides
CREATE INDEX etudiant_nom_index IF NOT EXISTS
FOR (e:Etudiant) ON (e.nom);

CREATE INDEX cours_code_index IF NOT EXISTS
FOR (c:Cours) ON (c.cours_code);
```

Listing 1 – Contraintes et index (01_schema.cypher)

3.2 Peuplement des données

Nous avons créé un jeu de données initial avec 6 étudiants, 5 cours et plusieurs villes :

```
CREATE (rachidi:Etudiant {
  student_id: 'ETU001',
  nom: 'Diallo',
  prenom: 'Rachidi',
  age: 22,
  email: 'rachidi.diallo@univ.fr',
  telephone: '+33612345678',
  filiere: 'Informatique',
  niveau: 'M1',
  ville: 'Paris',
  hobbies: ['programmation', 'football', 'lecture'],
  date_inscription: date('2023-09-01')
})
```

Listing 2 – Création des étudiants (extrait)

3.3 Création des relations d'amitié

Les relations d'amitié sont bidirectionnelles pour faciliter les requêtes :

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

Listing 3 – Relations AMI_AVEC bidirectionnelles

4 Requêtes et analyses

4.1 Requêtes de base

4.1.1 Lister tous les étudiants

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

Listing 4 – Afficher tous les étudiants

4.1.2 Trouver les amis d'un étudiant

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

Listing 5 – Amis de Rachidi

4.2 Analyses du réseau social

4.2.1 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,
       e.ville as ville
ORDER BY nb_amis DESC
LIMIT 5;
```

Listing 6 – Top étudiants par nombre d'amis

4.2.2 Force des amitiés

```
MATCH (e1:Etudiant)-[r:AMI_AVEC]->(e2:Etudiant)
WHERE r.force >= 8 AND id(e1) < id(e2)
RETURN e1.nom + ' ' + e1.prenom as personne1,
       e2.nom + ' ' + e2.prenom as personne2,
       r.force as force,
       r.type as type
ORDER BY r.force DESC;
```

Listing 7 – Amitiés les plus fortes

4.3 Recommandations d'amitié

L'algorithme de recommandation se base sur les amis en commun :

```
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,
       e2.ville as ville,
       amis_communs
ORDER BY amis_communs DESC
LIMIT 5;
```

Listing 8 – Recommandations basiques

4.4 Analyse géographique

```
MATCH (e1:Etudiant)-[:AMI_AVEC]->(e2:Etudiant)
WHERE e1.ville = e2.ville
WITH e1.ville as ville, COUNT(*) as nb_amities_locales
RETURN ville, nb_amities_locales
ORDER BY nb_amities_locales DESC;
```

Listing 9 – Groupes d'amis par ville

4.5 Performance académique

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

Listing 10 – Moyennes par étudiant

5 Scripts Python

5.1 Connexion à Neo4j

Le fichier `connect.py` fournit une classe pour gérer la connexion :

```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]

    def close(self):
        self.driver.close()
```

Listing 11 – Classe Neo4jConnection

5.2 Analyse et visualisation

Le fichier `analyze.py` contient la classe `GraphAnalyzer` qui permet :

- Extraction des données sous forme de DataFrames pandas
- Génération de graphiques avec NetworkX et Matplotlib
- Calcul de statistiques sur le réseau social
- Export des visualisations en PNG haute résolution

5.3 Visualisation du réseau

Le réseau d'amitiés est visualisé avec NetworkX :

```python
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()
for edge in edges:
    G.add_edge(edge['source'], edge['target'],
               weight=edge['weight'])

pos = nx.spring_layout(G, k=2, iterations=50)
nx.draw_networkx(G, pos, node_size=3000,
                 node_color='lightblue',
                 font_size=10, font_weight='bold')
plt.show()
```

Listing 12 – Visualisation avec NetworkX

6 Résultats et analyses

6.1 Statistiques générales

Statistiques de la base

- **Nombre d'étudiants** : 6
- **Nombre de cours** : 5
- **Nombre d'amitiés** : 16 (8 paires bidirectionnelles)
- **Nombre d'inscriptions** : 17
- **Densité du réseau** : 53% (bien connecté)

6.2 Insights sociaux

6.2.1 Centralité du réseau

Les étudiants les plus centraux (avec le plus d'amis) jouent un rôle important dans la cohésion du groupe. Dans notre jeu de données :

- Rachidi et Marie ont le plus d'amis (4 chacun)
- Ahmed et Sophie sont également bien connectés (3 amis)
- Le réseau est relativement équilibré

6.2.2 Types d'amitiés

- **Amitiés proches** (force ≥ 8) : 50% des relations
- **Amitiés d'études** : 37% des relations
- **Amitiés sportives/loisirs** : 13% des relations

6.3 Recommandations

L'algorithme a identifié plusieurs suggestions pertinentes :

1. Thomas pourrait se lier d'amitié avec Laura (2 amis communs)
2. Sophie et Ahmed partagent des intérêts communs via leurs amis
3. Les étudiants de la même ville mais pas encore amis

7 Scénarios avancés

7.1 Détection de communautés

Bien que notre jeu de données soit petit, on peut identifier des sous-groupes :

```
MATCH path = (e1:Etudiant)-[:AMI_AVEC*2..3]-(e2:Etudiant)
WHERE e1 <> e2
WITH e1, e2, path
WHERE 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;
```

Listing 13 – Identifier les cliques (groupes complets)

7.2 Chemins les plus courts

Trouver le chemin d'amitié entre deux étudiants :

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

Listing 14 – Plus court chemin entre Rachidi et Laura

7.3 Influence par cours

Analyser les groupes d'amis qui suivent les mêmes cours :

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

Listing 15 – Amis dans les mêmes cours

8 Installation et utilisation

8.1 Prérequis

1. **Neo4j** installé (Desktop ou Server)
2. **Python 3.x** avec pip
3. Packages Python : `neo4j`, `pandas`, `matplotlib`, `networkx`

8.2 Installation avec Docker

```
docker run -d \
  --name neo4j-social \
  -p 7474:7474 -p 7687:7687 \
  -e NEO4J_AUTH=neo4j/password123 \
  neo4j:latest
```

Listing 16 – Lancer Neo4j avec Docker

Accéder au navigateur : http://localhost:7474

8.3 Installation des packages Python

```
# Creer un environnement virtuel (recommande)
python3 -m venv venv
source venv/bin/activate  # Linux/Mac

# Installer les packages
cd python/
pip install -r requirements.txt
```

Listing 17 – Installer les dependances

8.4 Guide complet : Étapes de visualisation

Suivez ces étapes dans l'ordre pour visualiser le projet complet.

8.4.1 Étape 1 : Démarrer Neo4j

```
cd expose_neo4j/
docker-compose up -d

# Attendre 15 secondes que Neo4j demarre
sleep 15

# Verifier l'etat
docker-compose ps
```

Listing 18 – Lancer Neo4j avec Docker Compose

Vérification : Ouvrir http://localhost:7474 (Login : neo4j / Password : password123)

8.4.2 Étape 2 : Tester la connexion Python

```
cd python/
python3 connect.py

# Resultat attendu :
# - Connecte a Neo4j : bolt://localhost:7687
# - Connexion reussie !
# - Etudiants: 0 (normal, base vide)
```

Listing 19 – Test de connexion

8.4.3 Étape 3 : Peupler la base de données

```
python3 populate.py
# Menu : Taper "3" (Les deux - nettoyer puis peupler)
# Confirmer : Taper "oui"

# Resultat attendu :
# - Etudiants crees : 6
# - Cours crees : 5
# - Amities creees : 16
# - Inscriptions creees : 17
```

Listing 20 – Peuplement

8.4.4 Étape 4 : Visualiser dans Neo4j Browser

Ouvrir http://localhost:7474 et exécuter ces requêtes :

```
MATCH (n) RETURN n LIMIT 50
```

Listing 21 – Requete 1 - Voir tout le graphe

Ce que vous voyez

- 6 cercles bleus = Étudiants
- 5 cercles verts = Cours
- 3 cercles orange = Villes
- Flèches rouges = Amitiés
- Flèches bleues = Inscriptions aux cours

```
MATCH (rachidi:Etudiant {nom: 'Diallo'})-[:AMI_AVEC]->(ami)
RETURN rachidi, ami
```

Listing 22 – Requete 2 - Reseau d'amis de Rachidi

```
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
```

Listing 23 – Requete 3 - Recommandations

8.4.5 Étape 5 : Générer les graphiques Python

```
python3 analyze.py
# Menu : Taper "5" (Tout generer)

# Resultat :
# - Rapport texte affiche
# - ../images/reseau_amities.png genere
# - ../images/stats_etudiants.png genere
# - ../images/stats_cours.png genere
```

Listing 24 – Generation des visualisations

8.4.6 Étape 6 : Visualiser les graphiques

```
cd ../images/
ls -lh

# Ouvrir les graphiques
xdg-open reseau_amities.png
xdg-open stats_etudiants.png
xdg-open stats_cours.png
```

Listing 25 – Ouvrir les images

Graphiques générés :

- **reseau_amities.png** : Graphe du réseau social (NetworkX)
  - Cercles bleus = étudiants
  - Lignes grises = amitiés (épaisseur = force)
- **stats_etudiants.png** : 4 graphiques
  - Popularité (barres)
  - Répartition géographique (camembert)
  - Moyennes académiques (barres)
  - Corrélation amis/performance (nuage de points)
- **stats_cours.png** : 2 graphiques
  - Popularité des cours
  - Moyennes avec min/max

8.4.7 Étape 7 : Générer le diagramme UML

```
# Installer PlantUML
sudo apt install plantuml

# Generer le diagramme
cd ../uml/
plantuml modele_donnees.puml

# Visualiser
xdg-open modele_donnees.png
```

Listing 26 – Generation UML

Alternative : Copier le contenu de `modele_donnees.puml` sur https://www.plantuml.com/plantuml/uml/

8.4.8 Étape 8 : Compiler les documents

```
# Rapport
cd ../rapport/
pdflatex rapport.tex
pdflatex rapport.tex  # 2 fois pour la table
evince rapport.pdf &

# Presentation
cd ..
pdflatex presentation.tex
pdflatex presentation.tex
evince presentation.pdf &
```

Listing 27 – Compilation LaTeX

Récapitulatif des visualisations

Ce que vous avez visualisé :

1. **Neo4j Browser** : Graphe interactif complet
2. **3 graphiques PNG** : Réseau + statistiques
3. **Diagramme UML** : Modèle de données
4. **Rapport PDF** : 19 pages de documentation
5. **Présentation PDF** : 25+ slides professionnelles

Temps total : ~25 minutes

8.5 Arrêter et nettoyer

```
# Arreter Neo4j (conserve les donnees)
docker-compose stop

# Redemarrer
```

```
docker-compose start

# Tout supprimer (conteneur + donnees)
docker-compose down -v
```

Listing 28 – Gestion Docker

9 Améliorations possibles

9.1 Fonctionnalités additionnelles

— **Recommandations avancées** : Intégrer plus de critères (filière, hobbies, cours communs)

— **Algorithmes de graphes** : PageRank, Betweenness Centrality, Louvain

— **Évolution temporelle** : Analyser l'évolution du réseau dans le temps

— **Prédiction de liens** : Machine Learning pour prédire de futures amitiés

— **Interface web** : Créer une application Flask/Django avec visualisation interactive

9.2 Optimisations

— Ajouter plus d'index sur les propriétés fréquemment recherchées

— Utiliser des projections de graphes pour les analyses complexes

— Implémenter du caching pour les requêtes répétitives

— Partitionner les données pour de très grands graphes

9.3 Intégration avec d'autres systèmes

— API REST pour exposer les fonctionnalités

— Synchronisation avec un LDAP/Active Directory

— Export vers d'autres formats (GraphML, GEXF)

— Intégration avec des outils de BI (Tableau, Power BI)

10 Conclusion

Ce projet a permis de démontrer la puissance des bases de données orientées graphes pour modéliser et analyser des réseaux sociaux. Neo4j et le langage Cypher offrent une approche intuitive et performante pour :

— Représenter naturellement les relations entre entités

— Effectuer des requêtes complexes sur les chemins et connexions

— Calculer des métriques sociales (centralité, communautés)

— Générer des recommandations pertinentes

Les scripts Python développés permettent d'étendre les capacités d'analyse avec des visualisations riches et des statistiques avancées.

10.1 Compétences acquises

1. Modélisation de données en graphes
2. Maîtrise du langage Cypher
3. Analyse de réseaux sociaux
4. Visualisation de graphes avec NetworkX
5. Intégration Python-Neo4j

10.2 Applications réelles

Ce type de modélisation trouve des applications dans :

— Réseaux sociaux (Facebook, LinkedIn)

— Systèmes de recommandation (Netflix, Amazon)

— Détection de fraude (graphes de transactions)

— Analyse d'impact (réseaux de citations scientifiques)

— Gestion de connaissances (knowledge graphs)

Merci de votre attention !

A Annexe A : Structure des fichiers

Arborescence du projet (naviguer avec `cd` et `ls`)

```
expose_neo4j/
+-- scripts/
|   +-- 01_schema.cypher          (Contraintes et index)
|   +-- 02_data_initial.cypher    (Donnees de test)
|   +-- 03_requetes.cypher        (Requetes d'analyse)
|   +-- 04_clear_db.cypher        (Nettoyage)
+-- python/
|   +-- requirements.txt          (Dependances Python)
|   +-- connect.py                (Connexion Neo4j)
|   +-- populate.py               (Peuplement)
|   +-- analyze.py                (Analyses et viz)
+-- uml/
|   +-- modele_donnees.puml       (Diagramme UML)
|   +-- README.md                 (Instructions)
+-- rapport/
|   +-- rapport.tex               (Ce document)
+-- images/                       (Graphiques generes)
```

Taille approximative des fichiers :

- Scripts Cypher : 60-300 lignes chacun
- Scripts Python : 100-350 lignes chacun
- Images PNG : 100-500 Ko chacune
- Rapport PDF : ~300 Ko

B Annexe B : Exemples de résultats

B.1 Top 5 étudiants par nombre d'amis

Étudiant	Amis	Ville
Rachidi Diallo	4	Paris
Marie Dupont	4	Lyon
Ahmed Ben Ali	3	Paris
Sophie Martin	3	Marseille
Thomas Lefebvre	2	Lyon

B.2 Recommandations d'amitié

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

C Annexe C : Références

- Neo4j Documentation : https://neo4j.com/docs/
- Cypher Manual : https://neo4j.com/docs/cypher-manual/

— NetworkX Documentation : https://networkx.org/documentation/
— Graph Algorithms : https://neo4j.com/docs/graph-data-science/