NEO4J: GUIA PRÁTICO AVANÇADO COM EXEMPLOS DO MUNDO REAL

Índice

EXEMPLOS PRÁTICOS COMPLETOS

Exemplo 1: Rede Social Completa

Passo 1: Criar Estrutura Base

```
-- Criar usuários
CREATE
  (alice:User {id: 1, name: 'Alice', email: 'alice@email.com', joinDate:
date('2020-01-15')}),
  (bob:User {id: 2, name: 'Bob', email: 'bob@email.com', joinDate:
date('2020-06-10')}),
  (carol:User {id: 3, name: 'Carol', email: 'carol@email.com', joinDate:
date('2021-03-20')}),
  (dave:User {id: 4, name: 'Dave', email: 'dave@email.com', joinDate:
date('2021-09-05'}),
  (eve:User {id: 5, name: 'Eve', email: 'eve@email.com', joinDate:
date('2022-02-14'}),

-- Criar posts
  (p1:Post {id: 1, title: 'Neo4j Intro', content: 'Getting started...',
likes: 150, createdAt: datetime('2025-01-10T10:30:00Z')}),
  (p2:Post {id: 2, title: 'Graph Queries', content: 'Cypher basics...',
likes: 120, createdAt: datetime('2025-01-15T14:20:00Z')}),
  (p3:Post {id: 3, title: 'Best Practices', content: 'Performance
tips...', likes: 200, createdAt: datetime('2025-01-18T09:15:00Z')}),

-- Criar relacionamentos de amizade
  (alice)-[:FOLLOWS]->(bob),
  (alice)-[:FOLLOWS]->(carol),
  (bob)-[:FOLLOWS]->(alice),
  (bob)-[:FOLLOWS]->(dave),
  (carol)-[:FOLLOWS]->(dave),
  (carol)-[:FOLLOWS]->(eve),
  (dave)-[:FOLLOWS]->(eve),
  (eve)-[:FOLLOWS]->(alice),
```

```
-- Criar relacionamentos de posts
  (alice)-[:POSTED]->(p1),
  (bob)-[:POSTED]->(p2),
  (carol)-[:POSTED]->(p3),
  (alice)-[:LIKED {timestamp: datetime('2025-01-15T16:00:00Z')}]->(p2),
  (bob)-[:LIKED {timestamp: datetime('2025-01-18T10:30:00Z')}]->(p3),
  (dave)-[:LIKED {timestamp: datetime('2025-01-18T11:00:00Z')}]->(p3),
  (eve)-[:LIKED {timestamp: datetime('2025-01-20T12:00:00Z')}]->(p1)
```

Passo 2: Queries Úteis

Encontrar amigos em comum

```
MATCH (user:User {name: 'Alice'})-[:FOLLOWS]->(friend:User)<-[:FOLLOWS]-
(otherUser:User {name: 'Bob'})
RETURN DISTINCT friend.name AS commonFriend
```

Feed personalizado (últimas postagens de quem você segue)

```
MATCH (user:User {name: 'Alice'})-[:FOLLOWS]->(friend:User)-[:POSTED]->
(post:Post)
RETURN friend.name, post.title, post.createdAt
ORDER BY post.createdAt DESC
LIMIT 10
```

Recomendação de pessoas a seguir (amigos dos seus amigos)

```
MATCH (user:User {name: 'Alice'})-[:FOLLOWS]->(friend:User)-[:FOLLOWS]->
(recommended:User)
WHERE NOT (user)-[:FOLLOWS]->(recommended) AND recommended <> user
WITH recommended, count(friend) AS mutualFriends
RETURN recommended.name, mutualFriends
ORDER BY mutualFriends DESC
```

Influenciadores (usuários mais seguidos)

```
MATCH (user:User)<-[:FOLLOWS]-(follower:User)
WITH user, count(follower) AS followers
WHERE followers > 2
RETURN user.name, followers
ORDER BY followers DESC
```

Posts mais curtidos no último mês

```
MATCH (post:Post)<-[like:LIKED]-(user:User)
WHERE post.createdAt > datetime() - duration('P30D')
WITH post, count(like) AS likeCount
RETURN post.title, likeCount
ORDER BY likeCount DESC
LIMIT 5
```

Exemplo 2: E-commerce com Recomendações

Criar Dados

```
-- Criar produtos
CREATE
  (p1:Product {id: 1, name: 'Laptop', price: 1200, category:
'Electronics', stock: 5}),
  (p2:Product {id: 2, name: 'Mouse', price: 25, category: 'Electronics',
stock: 50}),
  (p3:Product {id: 3, name: 'Keyboard', price: 75, category:
'Electronics', stock: 30}),
  (p4:Product {id: 4, name: 'Monitor', price: 300, category:
'Electronics', stock: 10}),
  (p5:Product {id: 5, name: 'Book: Neo4j Guide', price: 45, category:
'Books', stock: 100}),
  (p6:Product {id: 6, name: 'Book: Database Design', price: 50, category:
'Books', stock: 80}),

-- Criar clientes
  (c1:Customer {id: 1, name: 'João', email: 'joao@email.com', totalSpent:
3500}),
  (c2:Customer {id: 2, name: 'Maria', email: 'maria@email.com',
totalSpent: 2800}),
  (c3:Customer {id: 3, name: 'Pedro', email: 'pedro@email.com',
totalSpent: 1200}),

-- Compras
  (c1)-[:BOUGHT {date: date('2025-01-10'), quantity: 1, price: 1200}]->
(p1),
  (c1)-[:BOUGHT {date: date('2025-01-12'), quantity: 2, price: 50}]->(p2),
  (c1)-[:BOUGHT {date: date('2025-01-15'), quantity: 1, price: 45}]->(p5),

  (c2)-[:BOUGHT {date: date('2025-01-05'), quantity: 1, price: 75}]->(p3),
  (c2)-[:BOUGHT {date: date('2025-01-08'), quantity: 1, price: 300}]->
(p4),
  (c2)-[:BOUGHT {date: date('2025-01-12'), quantity: 1, price: 50}]->(p6),

  (c3)-[:BOUGHT {date: date('2025-01-18'), quantity: 1, price: 25}]->(p2),
  (c3)-[:BOUGHT {date: date('2025-01-20'), quantity: 1, price: 75}]->(p3),
```

```
  (c3)-[:BOUGHT {date: date('2025-01-22'), quantity: 1, price: 1200}]->
(p1),

-- Favoritos
  (c1)-[:FAVORITE]->(p3),
  (c1)-[:FAVORITE]->(p4),
  (c2)-[:FAVORITE]->(p1),
  (c3)-[:FAVORITE]->(p5)
```

Queries Recomendação

Clientes que compraram juntos (comprou X também comprou Y)

```
MATCH (customer:Customer)-[:BOUGHT]->(product1:Product)
MATCH (other:Customer)-[:BOUGHT]->(product1)
MATCH (other)-[:BOUGHT]->(product2:Product)
WHERE customer <> other AND product1 <> product2
WITH customer, product2, count(other) AS coCustomers
RETURN product2.name, coCustomers
ORDER BY coCustomers DESC
LIMIT 5
```

Produtos frequentemente comprados juntos

```
MATCH (c:Customer)-[:BOUGHT]->(p1:Product)
MATCH (c)-[:BOUGHT]->(p2:Product)
WHERE p1 <> p2 AND p1.id < p2.id
WITH p1, p2, count(c) AS frequency
WHERE frequency > 1
RETURN p1.name, p2.name, frequency
ORDER BY frequency DESC
```

Recomendações para cliente baseado em compras similares

```
MATCH (targetCustomer:Customer {name: 'João'})-[:BOUGHT]->
(product:Product)
MATCH (similarCustomer:Customer)-[:BOUGHT]->(product)
WHERE similarCustomer <> targetCustomer
MATCH (similarCustomer)-[:BOUGHT]->(recommendedProduct:Product)
WHERE NOT (targetCustomer)-[:BOUGHT]->(recommendedProduct)
WITH recommendedProduct, count(similarCustomer) AS similarity
RETURN recommendedProduct.name, similarity
ORDER BY similarity DESC
LIMIT 5
```

Clientes por valor total gasto

```
MATCH (c:Customer)-[b:BOUGHT]->(p:Product)
WITH c, sum(b.quantity * b.price) AS totalSpent
RETURN c.name, totalSpent
ORDER BY totalSpent DESC
```

Exemplo 3: Estrutura Organizacional

Setup

```
CREATE
  -- Criadores
  (ceo:Person:Executive {name: 'Carlos Silva', role: 'CEO', level: 1,
salary: 15000}),
  (cto:Person:Executive {name: 'Ana Santos', role: 'CTO', level: 1,
salary: 12000}),
  (cfo:Person:Executive {name: 'Ricardo Oliveira', role: 'CFO', level: 1,
salary: 12000}),

  -- Diretores
  (it_dir:Person:Director {name: 'João Costa', role: 'IT Director', level:
2, salary: 8000}),
  (dev_dir:Person:Director {name: 'Maria Gonçalves', role: 'Development
Director', level: 2, salary: 8000}),
  (hr_dir:Person:Director {name: 'Sandra Ferreira', role: 'HR Director',
level: 2, salary: 7000}),

  -- Gerentes
  (dev_manager:Person:Manager {name: 'Pedro Teixeira', role: 'Dev
Manager', level: 3, salary: 5000}),
  (infra_manager:Person:Manager {name: 'Luis Rodrigues', role:
'Infrastructure Manager', level: 3, salary: 5000}),
  (hr_manager:Person:Manager {name: 'Catarina Monteiro', role: 'HR
Manager', level: 3, salary: 4500}),

  -- Desenvolvedores
  (dev1:Person:Developer {name: 'Rafael Alves', role: 'Senior Developer',
level: 4, salary: 3500}),
  (dev2:Person:Developer {name: 'Sofia Martins', role: 'Junior Developer',
level: 4, salary: 2500}),
  (dev3:Person:Developer {name: 'Bruno Mendes', role: 'Developer', level:
4, salary: 3000}),

  -- Relações hierárquicas
  (ceo)-[:MANAGES]->(cto),
  (ceo)-[:MANAGES]->(cfo),
  (cto)-[:MANAGES]->(it_dir),
  (cto)-[:MANAGES]->(dev_dir),
  (cfo)-[:MANAGES]->(hr_dir),
  (dev_dir)-[:MANAGES]->(dev_manager),
```

```
  (it_dir)-[:MANAGES]->(infra_manager),
  (hr_dir)-[:MANAGES]->(hr_manager),
  (dev_manager)-[:MANAGES]->(dev1),
  (dev_manager)-[:MANAGES]->(dev2),
  (dev_manager)-[:MANAGES]->(dev3),

  -- Relações de trabalho
  (dev1)-[:WORKS_ON_PROJECT]->(proj1:Project {name: 'Neo4j Migration',
budget: 50000}),
  (dev1)-[:WORKS_ON_PROJECT]->(proj2:Project {name: 'API Development',
budget: 30000}),
  (dev2)-[:WORKS_ON_PROJECT]->(proj2),
  (dev3)-[:WORKS_ON_PROJECT]->(proj1),
  (dev3)-[:WORKS_ON_PROJECT]->(proj2)
```

Queries Organizacionais

Toda cadeia de comando para uma pessoa

```
MATCH (person:Person {name: 'Sofia Martins'})
MATCH path = (person)<-[:MANAGES*]-(executive:Person)
RETURN [node IN nodes(path) | node.name] AS chain
```

Quantas pessoas relatam indiretamente a um executivo

```
MATCH (executive:Person {name: 'Carlos Silva'})
MATCH (subordinate:Person)
WHERE (executive)-[:MANAGES*]->(subordinate)
RETURN count(DISTINCT subordinate) AS totalSubordinates
```

Folha de pagamento por departamento

```
MATCH (director:Director)<-[:MANAGES*]-(employee:Person)
WITH director, sum(employee.salary) AS deptSalary
RETURN director.name, director.role, deptSalary
ORDER BY deptSalary DESC
```

Pessoas que trabalham no mesmo projeto

```
MATCH (p1:Person {name: 'Rafael Alves'})-[:WORKS_ON_PROJECT]->
(proj:Project)<-[:WORKS_ON_PROJECT]-(p2:Person)
WHERE p1 <> p2
RETURN DISTINCT p2.name, proj.name
```

Orçamento total de projetos por pessoa

```
MATCH (person:Person)-[:WORKS_ON_PROJECT]->(project:Project)
WITH person, sum(project.budget) AS totalBudget
RETURN person.name, totalBudget
ORDER BY totalBudget DESC
```

CASOS DE USO REAIS

Caso 1: Sistema de Detecção de Fraude

```
-- Criar rede de transações suspeitas
CREATE
  (a1:Account {id: 'ACC001', owner: 'John Doe', balance: 50000}),
  (a2:Account {id: 'ACC002', owner: 'Jane Smith', balance: 75000}),
  (a3:Account {id: 'ACC003', owner: 'Bob Wilson', balance: 30000}),
  (a4:Account {id: 'ACC004', owner: 'Unknown Seller', balance: 5000}),

  -- Transações
  (t1:Transaction {id: 'TXN001', amount: 5000, date: date('2025-01-01')}),
  (t2:Transaction {id: 'TXN002', amount: 3000, date: date('2025-01-02')}),
  (t3:Transaction {id: 'TXN003', amount: 7500, date: date('2025-01-03')}),
  (t4:Transaction {id: 'TXN004', amount: 2000, date: date('2025-01-04')}),

  -- Fluxo de transações
  (a1)-[:SENT {time: '10:30'}]->(t1)-[:RECEIVED_BY]->(a4),
  (a2)-[:SENT {time: '11:00'}]->(t2)-[:RECEIVED_BY]->(a4),
  (a1)-[:SENT {time: '14:30'}]->(t3)-[:RECEIVED_BY]->(a3),
  (a3)-[:SENT {time: '15:00'}]->(t4)-[:RECEIVED_BY]->(a4)

-- Detectar padrões suspeitos: múltiplas pessoas enviando para mesma conta
MATCH (sender:Account)-[:SENT]->(:Transaction)-[:RECEIVED_BY]->
(receiver:Account {owner: 'Unknown Seller'})
WITH receiver, count(DISTINCT sender) AS senderCount, collect(DISTINCT
sender.owner) AS senders
WHERE senderCount > 2
RETURN receiver.owner AS suspiciousAccount, senderCount, senders

-- Detectar ciclos (transferências em círculo)
MATCH path = (account:Account)-[:SENT]->(:Transaction)-[:RECEIVED_BY]->+
(account)
WHERE length(path) <= 5
RETURN [node IN nodes(path) WHERE node:Account | node.owner] AS cycle
```

Caso 2: Recomendação de Aprendizado

```
-- Criar dados de cursos e alunos
CREATE
  -- Cursos
  (c1:Course {id: 1, title: 'Python Basics', level: 'Beginner', duration:
40, rating: 4.5}),
  (c2:Course {id: 2, title: 'Advanced Python', level: 'Intermediate',
duration: 60, rating: 4.8}),
  (c3:Course {id: 3, title: 'Django Web Dev', level: 'Intermediate',
duration: 80, rating: 4.7}),
  (c4:Course {id: 4, title: 'Database Design', level: 'Advanced',
duration: 70, rating: 4.6}),
  (c5:Course {id: 5, title: 'Neo4j & Graphs', level: 'Advanced', duration:
50, rating: 4.9}),

  -- Alunos
  (s1:Student {id: 1, name: 'Alice', level: 'Intermediate'}),
  (s2:Student {id: 2, name: 'Bob', level: 'Beginner'}),

  -- Pré-requisitos
  (c2)-[:REQUIRES]->(c1),
  (c3)-[:REQUIRES]->(c2),
  (c4)-[:REQUIRES]->(c3),
  (c5)-[:REQUIRES]->(c4),

  -- Progresso dos alunos
  (s1)-[:COMPLETED {score: 92, date: date('2025-01-10')}]->(c1),
  (s1)-[:COMPLETED {score: 88, date: date('2025-01-25')}]->(c2),
  (s1)-[:IN_PROGRESS {progress: 45}]->(c3),

  (s2)-[:COMPLETED {score: 85, date: date('2025-02-05')}]->(c1),
  (s2)-[:INTERESTED_IN]->(c2),

  -- Tópicos dos cursos
  (c1)-[:COVERS]->(t1:Topic {name: 'Variables'}),
  (c1)-[:COVERS]->(t2:Topic {name: 'Loops'}),
  (c2)-[:COVERS]->(t2),
  (c2)-[:COVERS]->(t3:Topic {name: 'OOP'}),
  (c3)-[:COVERS]->(t3),
  (c3)-[:COVERS]->(t4:Topic {name: 'Web Frameworks'}),
  (c4)-[:COVERS]->(t5:Topic {name: 'Schema Design'}),
  (c5)-[:COVERS]->(t5),
  (c5)-[:COVERS]->(t6:Topic {name: 'Graph Theory'})

-- Recomendar próximo curso baseado em progresso
MATCH (student:Student {name: 'Alice'})-[:COMPLETED|IN_PROGRESS]->
(completedCourse:Course)
MATCH (completedCourse)-[:REQUIRES*..1]-(nextCourse:Course)
WHERE NOT (student)-[:COMPLETED|IN_PROGRESS]->(nextCourse)
AND nextCourse <> completedCourse
RETURN DISTINCT nextCourse.title, nextCourse.level
ORDER BY nextCourse.rating DESC

-- Encontrar colegas com interesses similares
```

```
MATCH (student1:Student {name: 'Alice'})-[:COMPLETED|IN_PROGRESS]->
(course:Course)-[:COVERS]-(topic:Topic)
MATCH (student2:Student)-[:COMPLETED|IN_PROGRESS]->(otherCourse:Course)-
[:COVERS]-(topic)
WHERE student1 <> student2
RETURN DISTINCT student2.name
```

OTIMIZAÇÃO E PERFORMANCE

Índices e Constraints

```
-- Criar índices para performance
CREATE INDEX ON :Person(email)
CREATE INDEX ON :Product(sku)
CREATE INDEX ON :Transaction(date)
CREATE INDEX ON :User(username)

-- Constraints de unicidade
CREATE CONSTRAINT ON (u:User) ASSERT u.email IS UNIQUE
CREATE CONSTRAINT ON (p:Product) ASSERT p.sku IS UNIQUE
CREATE CONSTRAINT ON (c:Customer) ASSERT c.accountNumber IS UNIQUE

-- Constraints de existência (Neo4j 4.1+)
CREATE CONSTRAINT ON (u:User) ASSERT u.email IS NOT NULL
CREATE CONSTRAINT ON (p:Product) ASSERT p.name IS NOT NULL

-- Verificar índices
SHOW INDEXES

-- Verificar constraints
SHOW CONSTRAINTS
```

EXPLAIN e PROFILE

```
-- Analisar plano de execução
EXPLAIN MATCH (p:Person)-[:KNOWS]->(friend:Person)
WHERE p.name = 'Alice'
RETURN friend

-- Executar e obter estatísticas reais
PROFILE MATCH (p:Person)-[:KNOWS]->(friend:Person)
WHERE p.name = 'Alice'
RETURN friend
```

Query Optimization Tips

```
-- ❌ Lento: Sem índices, full scan
MATCH (p:Person)
WHERE p.name = 'Alice'

-- ✅ Rápido: Com índice em name
CREATE INDEX ON :Person(name)
MATCH (p:Person {name: 'Alice'})

-- ❌ Lento: Múltiplos MATCH
MATCH (a:Person)
MATCH (b:Person)
MATCH (c:Person)

-- ✅ Rápido: Padrão único
MATCH (a:Person)-[:KNOWS]->(b:Person)-[:KNOWS]->(c:Person)

-- ❌ Lento: Sem limites
MATCH (p:Person)-[:KNOWS*]-(other)
RETURN other

-- ✅ Rápido: Com limite
MATCH (p:Person)-[:KNOWS*1..5]-(other)
RETURN other
LIMIT 100
```

INTEGRAÇÃO COM APLICAÇÕES

Exemplo: Python com Neo4j Driver

```
from neo4j import GraphDatabase

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

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

    def create_person(self, name, age, email):
        with self.driver.session() as session:
            session.write_transaction(self._create_and_return_person,
name, age, email)

    @staticmethod
    def _create_and_return_person(tx, name, age, email):
        result = tx.run("""
            CREATE (p:Person {name: $name, age: $age, email: $email})
            RETURN p.name AS name, p.age AS age, p.email AS email
        """, name=name, age=age, email=email)
```

```
            return result.single()

        def find_friends(self, name):
            with self.driver.session() as session:
                return session.read_transaction(self._find_friends, name)

        @staticmethod
        def _find_friends(tx, name):
            result = tx.run("""
                MATCH (p:Person {name: $name})-[:KNOWS]-(friend:Person)
                RETURN friend.name AS name
            """, name=name)
            return [row['name'] for row in result]

    # Uso
    repo = PersonRepo("bolt://localhost:7687", "neo4j", "password")
    repo.create_person("João", 30, "joao@email.com")
    friends = repo.find_friends("João")
    repo.close()
```

Exemplo: JavaScript com Neo4j Driver

```
    const neo4j = require('neo4j-driver');

    const driver = neo4j.driver(
        'bolt://localhost:7687',
        neo4j.auth.basic('neo4j', 'password')
    );

    const session = driver.session();

    // Criar pessoa
    async function createPerson(name, age, email) {
        const result = await session.run(
            'CREATE (p:Person {name: $name, age: $age, email: $email}) RETURN
    p',
            {name, age, email}
        );
        return result.records[0].get('p').properties;
    }

    // Encontrar amigos
    async function findFriends(name) {
        const result = await session.run(
            'MATCH (p:Person {name: $name})-[:KNOWS]-(friend) RETURN
    friend.name',
            {name}
        );
        return result.records.map(r => r.get('friend.name'));
    }
```

```
// Usar
createPerson('João', 30, 'joao@email.com')
    .then(person => console.log(person))
    .catch(err => console.error(err));

await session.close();
await driver.close();
```

TROUBLESHOOTING

Erro: "No database selected"

Solução:

```
-- Use a base de dados padrão
:USE system
:USE neo4j
```

Erro: "Property does not exist"

Problema:

```
-- Isso falha se 'age' não existir
MATCH (p:Person)
RETURN p.age + 10
```

Solução:

```
-- Usar coalesce
MATCH (p:Person)
RETURN coalesce(p.age, 0) + 10
```

Query muito lenta

Diagnosticar:

```
PROFILE MATCH (p:Person)-[:KNOWS*3]-(other)
RETURN other
```

Otimizar:

```
-- Adicionar índice
CREATE INDEX ON :Person(name)

-- Limitar profundidade
MATCH (p:Person)-[:KNOWS*1..3]-(other)
RETURN other LIMIT 100

-- Usar shortestPath
MATCH (p:Person {name: 'Alice'}), (o:Person {name: 'Bob'})
MATCH path = shortestPath((p)-[:KNOWS*]-(o))
RETURN path
```

Erro: "Heap Size"

Solução:

```
-- Usar LIMIT
MATCH (p:Person)
RETURN p
LIMIT 1000

-- Usar paginação com SKIP
MATCH (p:Person)
RETURN p
SKIP 10000
LIMIT 1000
```

Manual de Exemplos Versão: 1.0
Data: Dezembro 2025
Neo4j Versão: 5.x