Triggers e Procedures em Neo4j: Tutorial Prático Passo a Passo

Parte 1: Triggers - Automatização de Dados

O que é um Trigger?

Um trigger é uma função que executa **automaticamente** quando detecta uma alteração nos dados:

- Um nó é criado
- Uma propriedade é alterada
- Uma relação é eliminada

Passo 1: Criar o Ambiente de Teste

```
-- Limpar base de dados (cuidado em produção!)
MATCH (n)
DETACH DELETE n

-- Criar alguns nós iniciais
CREATE (cat:Category {name: "Eletrónicos", productCount: 0})
CREATE (stats:Stats {name: "UserCount", count: 0})
RETURN cat, stats
```

Passo 2: Trigger Simples - Timestamp Automático

Objetivo: Adicionar data de criação automaticamente a cada nó novo.

```
-- Executar na BASE DE DADOS SYSTEM
CALL apoc.trigger.install(
  'neo4j',
  'addCreatedAt',
  'UNWIND $createdNodes AS n
   SET n.createdAt = datetime()',
  {phase: 'before'}
)
```

Testar:

```
-- Criar um nó
CREATE (p:Product {name: "Laptop", price: 999})

-- Verificar se o timestamp foi adicionado
MATCH (p:Product {name: "Laptop"})
RETURN p.name, p.createdAt
```

Resultado Esperado:

```
p.name    | p.createdAt
----------|--------------------------
"Laptop"  | 2025-12-12T11:52:00.000Z
```

Passo 3: Trigger com Filtragem - Apenas para Labels Específicos

Objetivo: Criar um log automaticamente quando um novo Movie é criado.

```
-- Criar estrutura para logs
CREATE (log:LogEntry {id: 'init', count: 0})

-- Criar trigger
CALL apoc.trigger.install(
  'neo4j',
  'logMovieCreation',
  'UNWIND $createdNodes AS n
   WHERE "Movie" IN labels(n)
   CREATE (entry:LogEntry {
     action: "MOVIE_CREATED",
     movieTitle: n.title,
     timestamp: datetime()
   })',
  {phase: 'before'}
)
```

Testar:

```
-- Criar um Movie
CREATE (m:Movie {title: "Inception", released: 2010})

-- Verificar logs
MATCH (entry:LogEntry {action: "MOVIE_CREATED"})
RETURN entry.movieTitle, entry.timestamp
```

Passo 4: Trigger com Contador - Atualizar Estatísticas

Objetivo: Incrementar um contador cada vez que um novo User é criado.

```
CALL apoc.trigger.install(
  'neo4j',
```

```
  'incrementUserCount',
  'UNWIND $createdNodes AS n
   WHERE "User" IN labels(n)
   MATCH (stats:Stats {name: "UserCount"})
   SET stats.count = stats.count + 1',
  {phase: 'before'}
)
```

Testar:

```
-- Criar vários Users
CREATE (u1:User {name: "Alice"})
CREATE (u2:User {name: "Bob"})
CREATE (u3:User {name: "Charlie"})

-- Verificar contador
MATCH (stats:Stats {name: "UserCount"})
RETURN stats.count
```

Resultado: O contador deve ser 3.

Passo 5: Trigger com Validação - Rejeitar Dados Inválidos

Objetivo: Impedir que um preço negativo seja atribuído.

```
CALL apoc.trigger.install(
  'neo4j',
  'validatePrice',
  'UNWIND $assignedNodeProperties["price"] AS prop
   WHERE prop.new < 0
   CALL apoc.util.validate(false, "❌ ERRO: Preço não pode ser negativo!")
   RETURN null',
  {phase: 'before'}
)
```

Testar:

```
-- Tentar criar produto com preço negativo (vai falhar)
CREATE (p:Product {name: "Invalid", price: -50})
```

Resultado: Erro: "ERRO: Preço não pode ser negativo!"

Passo 6: Trigger com Relacionamentos - Auto-Conectar Nós

Objetivo: Conectar automaticamente um Produto à sua Categoria.

```
CALL apoc.trigger.install(
  'neo4j',
  'autoCategory',
  'UNWIND $createdNodes AS prod
   WHERE "Product" IN labels(prod) AND prod.category IS NOT NULL
   MATCH (cat:Category {name: prod.category})
   CREATE (prod)-[:IN_CATEGORY]->(cat)',
  {phase: 'before'}
)
```

Testar:

```
-- Criar produto com referência a categoria
CREATE (p:Product {name: "Rato", price: 25, category: "Eletrónicos"})

-- Verificar relação criada
MATCH (p:Product {name: "Rato"})-[r:IN_CATEGORY]->(cat:Category)
RETURN p.name, r, cat.name
```

Passo 7: Trigger com Aviso de Atualização - Timestamp de Modificação

Objetivo: Rastrear quando um nó foi modificado pela última vez.

```
CALL apoc.trigger.install(
  'neo4j',
  'trackUpdates',
  'UNWIND keys($assignedNodeProperties) AS key
   UNWIND $assignedNodeProperties[key] AS prop
   WITH prop.node AS node
   SET node.lastModified = datetime(),
       node.lastModifiedBy = "system"',
  {phase: 'before'}
)
```

Testar:

```
CREATE (p:Product {name: "Monitor", price: 200})
-- Esperar um pouco e depois atualizar
MATCH (p:Product {name: "Monitor"})
SET p.price = 180
RETURN p.name, p.lastModified
```

Parte 2: Procedures Customizadas

O que é uma Procedure?

Uma procedure é uma **função reutilizável** que você chama quando necessário:

```
CALL custom.myProcedure(parametro1, parametro2)
YIELD resultado1, resultado2
RETURN resultado1
```

Passo 1: Procedure Simples - Buscar por Nome

Objetivo: Criar uma function que procura um Product pelo nome.

```
CALL apoc.custom.installProcedure(
  'getProduct',
  'MATCH (p:Product {name: $name})
   RETURN p',
  'read',
  [['name', 'STRING']],
  [['product', 'NODE']]
)
```

Usar:

```
CALL custom.getProduct("Laptop") YIELD product
RETURN product.name, product.price
```

Passo 2: Procedure com Múltiplos Parâmetros - Filtrar por Intervalo de Preço

Objetivo: Encontrar produtos dentro de um intervalo de preço.

```
CALL apoc.custom.installProcedure(
  'getProductsByPrice',
  'MATCH (p:Product)
   WHERE p.price >= $minPrice AND p.price <= $maxPrice
   RETURN p
   ORDER BY p.price ASC',
  'read',
  [['minPrice', 'FLOAT'], ['maxPrice', 'FLOAT']],
  [['product', 'NODE']]
)
```

Usar:

```
CALL custom.getProductsByPrice(20, 200) YIELD product
RETURN product.name, product.price
```

Resultado:

```
product.name | product.price
-------------|--------------
"Rato"       | 25
"Monitor"    | 180
```

Passo 3: Procedure com Retorno Formatado - Resumo de Produto

Objetivo: Retornar informações formatadas de um produto.

```
CALL apoc.custom.installProcedure(
  'productSummary',
  'MATCH (p:Product {name: $name})
   WITH p, [(p)-[:IN_CATEGORY]->(cat) | cat.name][0] AS category
   RETURN {
     name: p.name,
     price: "€" + toString(p.price),
     category: category,
     createdAt: p.createdAt
   } AS summary',
  'read',
  [['name', 'STRING']],
  [['summary', 'MAP']]
)
```

Usar:

```
CALL custom.productSummary("Laptop") YIELD summary
RETURN summary
```

Resultado:

```
{
  name: "Laptop",
  price: "€999",
  category: "Eletrónicos",
```

```
  createdAt: 2025-12-12T11:52:00.000Z
}
```

Passo 4: Procedure com Agregação - Estatísticas por Categoria

Objetivo: Obter estatísticas de produtos por categoria.

```
CALL apoc.custom.installProcedure(
  'categoryStats',
  'MATCH (cat:Category {name: $categoryName})<-[:IN_CATEGORY]-(p:Product)
   WITH cat,
        count(p) AS productCount,
        avg(p.price) AS avgPrice,
        min(p.price) AS minPrice,
        max(p.price) AS maxPrice
   RETURN {
     category: cat.name,
     productCount: productCount,
     averagePrice: apoc.math.round(avgPrice, 2),
     minPrice: minPrice,
     maxPrice: maxPrice
   } AS stats',
  'read',
  [['categoryName', 'STRING']],
  [['stats', 'MAP']]
)
```

Usar:

```
CALL custom.categoryStats("Eletrónicos") YIELD stats
RETURN stats
```

Passo 5: Procedure de Escrita - Criar Produto com Validação

Objetivo: Criar um novo Product com validações automáticas.

```
CALL apoc.custom.installProcedure(
  'createProduct',
  'WITH $name AS name, $price AS price, $category AS category
   WHERE name IS NOT NULL AND price > 0
   CREATE (p:Product {
     name: name,
     price: price,
     category: category,
     createdAt: datetime()
```

```
    })-[:IN_CATEGORY]->(cat:Category {name: category})
    RETURN p',
   'write',
   [['name', 'STRING'], ['price', 'FLOAT'], ['category', 'STRING']],
   [['product', 'NODE']]
)
```

Usar:

```
CALL custom.createProduct("Teclado", 75.50, "Eletrónicos") YIELD product
RETURN product.name, product.createdAt
```

Passo 6: Procedure Complexa - Análise de Ator de Filmes

Objetivo: Análise completa de um ator (filmes, anos, média de idade).

```
CALL apoc.custom.installProcedure(
  'actorAnalysis',
  'MATCH (a:Person {name: $actorName})-[:ACTED_IN]->(m:Movie)
   WITH a, collect({title: m.title, year: m.released}) AS filmography
   RETURN {
     actor: a.name,
     totalMovies: size(filmography),
     filmography: filmography,
     yearsActive: {
       first: apoc.coll.min([f.year | f IN filmography]),
       last: apoc.coll.max([f.year | f IN filmography])
     },
     era: CASE
       WHEN apoc.coll.max([f.year | f IN filmography]) < 2000 THEN
"Classic"
       WHEN apoc.coll.max([f.year | f IN filmography]) < 2010 THEN
"Modern"
       ELSE "Contemporary"
     END
   } AS analysis',
  'read',
  [['actorName', 'STRING']],
  [['analysis', 'MAP']]
)
```

Usar:

```
CALL custom.actorAnalysis("Tom Hanks") YIELD analysis
RETURN analysis
```

Parte 3: Combinando Triggers e Procedures

Cenário Completo: Sistema de Comentários em Filmes

1. Preparação

```
-- Criar estrutura
CREATE (m:Movie {title: "The Matrix", released: 1999})
CREATE (stats:Stats {name: "CommentStats", count: 0})
RETURN m, stats
```

2. Trigger: Auto-Incrementar Contador de Comentários

```
CALL apoc.trigger.install(
  'neo4j',
  'countComments',
  'UNWIND $createdNodes AS n
   WHERE "Comment" IN labels(n)
   MATCH (stats:Stats {name: "CommentStats"})
   SET stats.count = stats.count + 1',
  {phase: 'before'}
)
```

3. Procedure: Obter Comentários de um Filme

```
CALL apoc.custom.installProcedure(
  'getMovieComments',
  'MATCH (m:Movie {title: $title})<-[:COMMENTS_ON]-(c:Comment)
   RETURN c
   ORDER BY c.createdAt DESC
   LIMIT $limit',
  'read',
  [['title', 'STRING'], ['limit', 'INTEGER']],
  [['comment', 'NODE']]
)
```

4. Procedure: Criar Comentário com Auto-Conexão

```
CALL apoc.custom.installProcedure(
  'addComment',
  'MATCH (m:Movie {title: $movieTitle})
   CREATE (c:Comment {
     text: $text,
```

```
      createdAt: datetime(),
      rating: $rating
    })-[:COMMENTS_ON]->(m)
    RETURN c',
  'write',
  [['movieTitle', 'STRING'], ['text', 'STRING'], ['rating', 'INTEGER']],
  [['comment', 'NODE']]
)
```

5. Testar o Fluxo Completo

```
-- Adicionar comentários
CALL custom.addComment("The Matrix", "Filme épico!", 5) YIELD comment
RETURN comment

CALL custom.addComment("The Matrix", "Muito bom, recomendo", 5) YIELD
comment
RETURN comment

-- Obter comentários
CALL custom.getMovieComments("The Matrix", 10) YIELD comment
RETURN comment.text, comment.rating

-- Verificar stats
MATCH (stats:Stats {name: "CommentStats"})
RETURN stats.count AS totalComentarios
```

Gerenciamento de Triggers e Procedures

Listar Todos os Triggers

```
CALL apoc.trigger.show('neo4j')
```

Listar Todas as Procedures

```
CALL apoc.custom.list()
```

Parar um Trigger Temporariamente

```
CALL apoc.trigger.stop('neo4j', 'addCreatedAt')
```

Retomar um Trigger

```
CALL apoc.trigger.start('neo4j', 'addCreatedAt')
```

Eliminar um Trigger

```
CALL apoc.trigger.drop('neo4j', 'addCreatedAt')
```

Eliminar Todos os Triggers

```
CALL apoc.trigger.dropAll('neo4j')
```

Dicas Importantes

Evitar Loops Infinitos

```
-- ❌ PERIGOSO: Trigger que cria nó, que dispara trigger novamente
UNWIND $createdNodes AS n
WHERE "Product" IN labels(n)
CREATE (log:Log)  -- Isto cria nós, disparando novamente!

-- ✅ CORRETO: Criar nós com label que não dispara trigger
UNWIND $createdNodes AS n
WHERE "Product" IN labels(n) AND NOT "Log" IN labels(n)
CREATE (log:InternalLog)  -- Log interno não tem trigger
```

Usar Timestamps Corretamente

```
-- Em fase 'before' e 'after' (usa datetime())
SET n.createdAt = datetime()

-- Em triggers com $commitTime (timestamp em ms)
SET n.txTime = $commitTime
```

Testar Sempre em Pequena Escala

```
-- Primeiro: teste com 1 nó
CREATE (test:TestNode {name: "teste"})
```

```
-- Depois: monitore logs
MATCH (log:LogEntry) RETURN log

-- Finalmente: aplique em produção
```

Versão: 2.0 | **Tutorial Prático** | **Neo4j 5.x com APOC**