Modelagem e Armazenamento de um Sistema de Cidades e Estradas utilizando Neo4j

Trabalho 2 de Banco de Dados para Ciência de Dados

Pedro Augusto Benevides Salviano - 790983

Contextualização

O presente trabalho se propõe a apresentar uma implementação do armazenamento de cidades como nós e as rotas entre tais cidades como arestas em um grafo. Tal implementação se dará no neo4j, um SGDB orientado a grafos.

Dados

Os dados originais foram obtidos do IBGE (Municípios SP), DER-SP (Malha rodoviária SP), overpass - ferramenta de consulta aos dados do OpenStreetMap - (população e coordenadas dos centros urbanos e consulta manual ao google maps para obter os dados das rodovias que vieram de fato a serem utilizados.
Links para datasets originais:

- Municípios
    - https://geoftp.ibge.gov.br/organizacao_do_territorio/malhas_territoriais/malhas_municipais/municipio_2023/UFs/SP/SP_Municipios_2023.zip
- Documentação OpenStreetMap:
    - https://wiki.openstreetmap.org/wiki/Downloading_data

Os arquivos foram então importados na ferramenta QGIS para conversão em GeoJSON, seguindo o passo a passo (mais fácil de trabalhar com dados JSON no Neo4J).

Para importar o JSON, foi utilizado o plugin apoc, e portanto foi necessário seguir o processo de instalação e configuração para habilitar a importação de arquivos.

A importação do conjunto de dados para cidades foi direto, no entanto optei por não importar as coordenadas que definem os limites da área da cidade, e utilizei um segundo dataset, obtido através da ferramenta overpass turbo para obter o conjunto de cidades do estado de São Paulo, com as coordenadas para os centros urbanos e respectivas populações.

Para a última etapa, que seria a importação dos dados das rodovias, após muito esforço, optei por desistir de importar os dados da malha rodoviária obtidos do DER-SP, pois a transformação dos dados em geometria multilinestring para arestas entre cidades se

apresentou uma tarefa muito complexa, e após gastar muito tempo insistindo optei por consultar manualmente as rotas entre as 20 cidades mais populosas do estado e limitar o escopo à esse subconjunto, gerando o tipo de rodovia aleatoriamente, uma vez que a rota entre 2 cidades pode atravessar diferentes rodovias, e pelo mesmo motivo não registrei o nome da rodovia.

Os dados obtidos na consulta manual foram salvos em um csv e importados no neo4J.

Os dados sobre regiões também foram imputados manualmente.

Script

O script .cypher e os documentos importados podem ser encontrados em https://github.com/pedro-salviano/cityGraphOnNeo4j

```
Unset
// Importar dataset cidades de São Paulo, dados obtidos no site do IBGE
https://geoftp.ibge.gov.br/organizacao_do_territorio/malhas_territoriais/mal
has_municipais/municipio_2023/UFs/SP/SP_Municipios_2023.zip
// e convertidos em geojson para importação utilizando a ferramenta QGIS


CALL apoc.load.json("file:///SP_Municipios_2023.geojson") YIELD value
WITH value.features AS features
UNWIND features AS feature
CREATE (c:Cidade {
       cod_municipio: feature.properties.CD_MUN,
       municipio: feature.properties.NM_MUN,
       pop: 0,
       coord: point(
       {
              longitude: 0, latitude: 0
       }
       )
})
RETURN c;


// Obter dados de população e coordenadas do OpenStreetMap através do
overpassTurbo.
// Script:
// [out:csv( name, ::lat, ::lon, population)];
// area["name"="São Paulo"]["admin_level"="4"]->.estado;
// node["place"~"^(city|town|village)$"](area.estado);
// out body;
```

```
// Importar dados do overpass no neo4j para preencher os dados de população
e coordenadas do centro urbano
CALL apoc.load.csv('file:///populacao_coords.csv', {header:false, skip:1,
sep: ' '}) YIELD list
MATCH (c:Cidade)
WHERE c.municipio = list[0]
SET  c.pop = toInteger(list[3]), c.coord = point({latitude:
toFloat(list[1]), longitude: toFloat(list[2])})
RETURN c;


// Ao tentar importar os dados das rodovias foi notada que seria uma tarefa
de grande complexidade, e portanto decidi selecionar as 20 cidades mais
populosas do dataset
MATCH (c) WHERE c.pop IS NULL DELETE c;
MATCH (c:Cidade) ORDER BY c.pop DESC SKIP 20 DELETE c;

// e pesquisei manualmente os trajetos, levando em consideração em calcular
apenas os trajetos entre as cidades imediatamente conectadas entre si do
conjunto
// Os resultados foram salvos em um csv
CALL apoc.load.csv("file:///trajetos.csv") YIELD list
MATCH (c1:Cidade{municipio: list[0]}), (c2:Cidade{municipio: list[1]})
MERGE (c1)-[r1:Trajeto{distanciaKM: toFloat(list[2])}]->(c2);


// Definir aleatoriamente o tipo de rodovia, sendo que direções diferentes
de uma mesma rodovia podem ter tipos diferentes.
MATCH ()-[r]->()
SET r.tipo = apoc.coll.randomItem(['Federal', 'Estadual', 'Vicinal'])
RETURN r;

//Criar e associar regiões com Cidades
CREATE (RMSP:Regiao{nome: "Região Metropolitana de São Paulo", Decricao: "Na
Região Metropolitana de São Paulo se concentram os melhores serviços urbanos
e sociais, comércio e serviços sofisticados, instituições de pesquisa e
ensino superior de referência, uma complexa rede de atendimento à saúde e a
maior oferta de grandes eventos e instituições culturais."})
CREATE (RMC:Regiao{nome: "Região Metropolitana de Campinas", Decricao: "A
Região Metropolitana de Campinas comporta um parque industrial moderno,
diversificado e composto por segmentos de natureza complementar. Possui uma
estrutura agrícola e agroindustrial bastante significativa e desempenha
atividades terciárias de expressiva especialização."})
CREATE (VP:Regiao{nome: "Vale do Paraíba", Decricao: "O Vale do Paraíba,
localizado entre São Paulo e Rio de Janeiro, é uma região marcada pela
indústria automobilística, aeroespacial e tecnológica, com destaque para São
José dos Campos. Além da economia, é conhecido por seu patrimônio histórico,
religioso e cultural, como Aparecida, centro de peregrinação. A região
```

```
combina desenvolvimento econômico com áreas de preservação ambiental, como a Serra da Mantiqueira."})
CREATE (BS:Regiao{nome: "Baixada Santista", Decricao: "A Baixada Santista, localizada no estado de São Paulo, é uma região costeira composta por nove municípios, com destaque para Santos, o maior porto da América Latina. Conhecida por sua importância econômica e turística, possui praias, áreas de preservação ambiental e infraestrutura portuária estratégica. A região enfrenta desafios como expansão urbana desordenada e questões ambientais."})
CREATE (ISP:Regiao{nome: "Interior de São Paulo", Decricao: "O interior de São Paulo é uma região diversificada, com cidades desenvolvidas como Campinas, Ribeirão Preto e São José do Rio Preto, que se destacam pela agroindústria, tecnologia e educação. A área combina grandes polos urbanos com áreas rurais produtivas, sendo essencial para a economia do estado. Além disso, oferece qualidade de vida e riqueza cultural, com eventos tradicionais e culinária marcante."})

MATCH (c1:Cidade{municipio: "Ribeirão Preto"}), (ISP:Regiao{nome: "Interior de São Paulo"})
CREATE (c1)-[r:FazParte]->(ISP);
MATCH (c1:Cidade{municipio: "Campinas"}), (ISP:Regiao{nome: "Interior de São Paulo"})
CREATE (c1)-[r:FazParte]->(ISP);
MATCH (c1:Cidade{municipio: "São José dos Campos"}), (ISP:Regiao{nome: "Interior de São Paulo"})
CREATE (c1)-[r:FazParte]->(ISP);
MATCH (c1:Cidade{municipio: "Franca"}), (ISP:Regiao{nome: "Interior de São Paulo"})
CREATE (c1)-[r:FazParte]->(ISP);
MATCH (c1:Cidade{municipio: "Bauru"}), (ISP:Regiao{nome: "Interior de São Paulo"})
CREATE (c1)-[r:FazParte]->(ISP);
MATCH (c1:Cidade{municipio: "São José do Rio Preto"}), (ISP:Regiao{nome: "Interior de São Paulo"})
CREATE (c1)-[r:FazParte]->(ISP);
MATCH (c1:Cidade{municipio: "Sorocaba"}), (ISP:Regiao{nome: "Interior de São Paulo"})
CREATE (c1)-[r:FazParte]->(ISP);
MATCH (c1:Cidade{municipio: "Campinas"}), (ISP:Regiao{nome: "Interior de São Paulo"})
CREATE (c1)-[r:FazParte]->(ISP);
MATCH (c1:Cidade{municipio: "Piracicaba"}), (ISP:Regiao{nome: "Interior de São Paulo"})
CREATE (c1)-[r:FazParte]->(ISP);
MATCH (c1:Cidade{municipio: "Jundiaí"}), (ISP:Regiao{nome: "Interior de São Paulo"})
CREATE (c1)-[r:FazParte]->(ISP);
```

```
MATCH (c1:Cidade{municipio: "Santos"}), (BS:Regiao{nome: "Baixada
Santista"})
CREATE (c1)-[r:FazParte]->(BS);

MATCH (c1:Cidade{municipio: "São José dos Campos"}), (VP:Regiao{nome: "Vale
do Paraíba"})
CREATE (c1)-[r:FazParte]->(VP);

MATCH (c1:Cidade{municipio: "Campinas"}), (RMC:Regiao{nome: "Região
Metropolitana de Campinas"})
CREATE (c1)-[r:FazParte]->(RMC);

MATCH (c1:Cidade{municipio: "Carapicuíba"}), (RMSP:Regiao{nome: "Região
Metropolitana de São Paulo"})
CREATE (c1)-[r:FazParte]->(RMSP);
MATCH (c1:Cidade{municipio: "Osasco"}), (RMSP:Regiao{nome: "Região
Metropolitana de São Paulo"})
CREATE (c1)-[r:FazParte]->(RMSP);
MATCH (c1:Cidade{municipio: "Mogi das Cruzes"}), (RMSP:Regiao{nome: "Região
Metropolitana de São Paulo"})
CREATE (c1)-[r:FazParte]->(RMSP);
MATCH (c1:Cidade{municipio: "Itaquaquecetuba"}), (RMSP:Regiao{nome: "Região
Metropolitana de São Paulo"})
CREATE (c1)-[r:FazParte]->(RMSP);
MATCH (c1:Cidade{municipio: "Guarulhos"}), (RMSP:Regiao{nome: "Região
Metropolitana de São Paulo"})
CREATE (c1)-[r:FazParte]->(RMSP);
MATCH (c1:Cidade{municipio: "Mauá"}), (RMSP:Regiao{nome: "Região
Metropolitana de São Paulo"})
CREATE (c1)-[r:FazParte]->(RMSP);
MATCH (c1:Cidade{municipio: "Santo André"}), (RMSP:Regiao{nome: "Região
Metropolitana de São Paulo"})
CREATE (c1)-[r:FazParte]->(RMSP);
MATCH (c1:Cidade{municipio: "São Bernardo do Campo"}), (RMSP:Regiao{nome:
"Região Metropolitana de São Paulo"})
CREATE (c1)-[r:FazParte]->(RMSP);
MATCH (c1:Cidade{municipio: "Diadema"}), (RMSP:Regiao{nome: "Região
Metropolitana de São Paulo"})
CREATE (c1)-[r:FazParte]->(RMSP);
MATCH (c1:Cidade{municipio: "São Paulo"}), (RMSP:Regiao{nome: "Região
Metropolitana de São Paulo"})
CREATE (c1)-[r:FazParte]->(RMSP);


// Menor distância entre 2 cidades
MATCH (source:Cidade)-[r:Trajeto]-(target:Cidade)
RETURN gds.graph.project(
  'distanciaEntreCidade',
```

```
  source,
  target,
  { relationshipProperties: r { .distanciaKM } }
)

MATCH (c1:Cidade {municipio: 'Ribeirão Preto'}), (c2:Cidade {municipio: 'São
Bernardo do Campo'})
CALL gds.shortestPath.dijkstra.stream('distanciaEntreCidade', {
       sourceNode: c1,
       targetNodes: c2,
       relationshipWeightProperty: 'distanciaKM'
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
RETURN
       index,
       gds.util.asNode(sourceNode).municipio AS MunicipioOrigem,
       gds.util.asNode(targetNode).municipio AS MunicipioDestino,
       totalCost AS DistanciaTotal,
       [nodeId IN nodeIds | gds.util.asNode(nodeId).municipio] AS Caminho
ORDER BY index;

// Listar cidades conectadas por rodovia de determinado tipo
MATCH (a)-[r:Trajeto]->(b)
WHERE r.tipo = 'Federal'
RETURN a.municipio, r.tipo, b.municipio;

//Encontrar as cidades que estejam à uma distância máxima definida de uma
cidade escolhida
WITH 150 as maxDist
MATCH (c1:Cidade {municipio: 'Sorocaba'}), (c2:Cidade)
CALL gds.shortestPath.dijkstra.stream('distanciaEntreCidade', {
       sourceNode: c1,
       targetNodes: c2,
       relationshipWeightProperty: 'distanciaKM'
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
WHERE totalCost <= maxDist
RETURN
       index,
       gds.util.asNode(sourceNode).municipio AS MunicipioOrigem,
       gds.util.asNode(targetNode).municipio AS MunicipioDestino,
       totalCost AS DistanciaTotal,
       [nodeId IN nodeIds | gds.util.asNode(nodeId).municipio] AS Caminho
ORDER BY index;

//Identificar hubs
MATCH (a:Cidade)-[r:Trajeto]-(b:Cidade)
WITH a, COUNT(DISTINCT r) AS qtdConexoes
```

```
WHERE qtdConexoes > 3
RETURN a.municipio, qtdConexoes;

// Criar indíce de texto
CREATE TEXT INDEX idx_municipio FOR (c:Cidade) ON c.municipio ;
```

Modelagem

Os requisitos funcionais exigiam a implementação das cidades como nós e rodovias como arestas entre tais cidades, tal abordagem é a mais intuitiva, no entanto apresenta um desafio caso o desenvolvedor tente utilizar dados em formatos georreferenciados, uma vez que tais formatos registram coordenadas como pontos em um vetor, e por exemplo, uma rodovia é registrada com uma geometria multilinestring, que é uma série de coordenadas que formam um trecho da rodovia, e uma determinada rodovia pode ser quebrada em diversos pequenos trechos representados por conjuntos de coordenadas.

Dessa forma, o conjunto de cidades foi importado, ignorando o vetor de coordenadas que define os limites da área do município, e foi cruzado com um conjunto obtido do OpenStreetMap, que continha as coordenadas do centro urbano e a população das cidades, vilas e distritos do estado de São Paulo. As propriedades inseridas nas cidades foram: Munícipio (nome do município), coord (Coordenada do centro urbano), cod_municipio (código IBGE de 7 dígitos do município), pop (População do município) Modelagem cidade:

Unset

```
"identity": 68,
  "labels": [
       "Cidade"
  ],
  "properties": {
       "pop": 379146,
       "cod_municipio": "3506003",
       "coord": point({srid:4326, x:-49.0705863, y:-22.3218102}),
       "municipio": "Bauru"
  }
```

Como relatado a importação dos dados a partir do conjunto de dados baixados foi abandonada, devido a complexidade da tarefa, pelo exercício: uma modelagem que facilitaria a tarefa e ainda assim representaria bem a rodovia, seria importar cada coordenada de um trecho como um nó para representar um ponto de uma rodovia, e conectar tais pontos com arestas que representam os trechos da rodovia, e conectar com outras rodovias ou outros trechos da mesma rodovia a partir da interseção de pontos de diferentes rodovias. Nesta modelagem as rodovias não se conectam diretamente às

cidades, no entanto poderíamos obter o caminho entre as cidades ao buscar o nó de rodovias que estejam a um determinado raio do nodo da cidade, e partir desse nó fazer a consulta do caminho até uma próxima cidade.

Modelagem efetivamente feita neste trabalho: seguindo os requisitos funcionais, as rodovias foram modeladas como arestas entre cidades, no entanto como o caminho entre 2 cidades poderiam passar por diferentes rodovias, optei por não inserir dados do nome da rodovia, e inseri aleatoriamente o tipo das rodovias em cada aresta. Também tomei a liberdade de inserir 2 arestas (uma para cada direção) dos trechos encontrados, assim o grafo pode representar unicamente os sentidos das rodovias.

As rodovias foram mapeadas como relacionamentos do tipo “Trajeto”, e propriedades sendo um tipo [“Federal”, “Estadual”, “Vicinal”], e a distância do trajeto entre os nodos origem e destino.

```
Unset
{
  "identity": 6942299925103247794,
  "start": 68,
  "end": 434,
  "type": "Trajeto",
  "properties": {
       "tipo": "Estadual",
       "distanciaKM": "193"
  },
  "elementId": "5:f0108985-680b-44cf-9681-e2157f6d55ce:6942299925103247794",
  "startNodeElementId": "4:f0108985-680b-44cf-9681-e2157f6d55ce:68",
  "endNodeElementId": "4:f0108985-680b-44cf-9681-e2157f6d55ce:434"
}
```

Por fim as regiões foram inseridas manualmente, sendo elas [“Região Metropolitana de São Paulo”, “Região Metropolitana de Campinas”, “Vale do Paraíba”, “Baixada Santista”, “Interior de São Paulo”], tendo apenas o rótulo Regiao e as propriedades nome e descricao.

```
Unset
{
  "identity": 397,
  "labels": [
       "Regiao"
  ],
  "properties": {
       "Decricao": "Na Região Metropolitana de São Paulo se concentram os
melhores serviços urbanos e sociais, comércio e serviços sofisticados,
instituições de pesquisa e ensino superior de referência, uma complexa rede
de atendimento à saúde e a maior oferta de grandes eventos e instituições
culturais.",
       "nome": "Região Metropolitana de São Paulo"
```

```
  },
  "elementId": "4:f0108985-680b-44cf-9681-e2157f6d55ce:397"
}
```

Consultas

a) A consulta da rota mais curta entre duas cidades e a distância total da rota foi implementada utilizando do plugin de ciência de dados “Neo4j Graph Data Science” criando uma projeção, e utilizando a projeção com a procedure que implementa o algoritmo de dijkstra para encontrar o menor caminho:

```
Unset
MATCH (source:Cidade)-[r:Trajeto]-(target:Cidade)
RETURN gds.graph.project(
  'distanciaEntreCidade',
  source,
  target,
  { relationshipProperties: r { .distanciaKM } }
)

MATCH (c1:Cidade {municipio: 'Ribeirão Preto'}), (c2:Cidade {municipio: 'São
Bernardo do Campo'})
CALL gds.shortestPath.dijkstra.stream('distanciaEntreCidade', {
       sourceNode: c1,
       targetNodes: c2,
       relationshipWeightProperty: 'distanciaKM'
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
RETURN
       index,
       gds.util.asNode(sourceNode).municipio AS MunicipioOrigem,
       gds.util.asNode(targetNode).municipio AS MunicipioDestino,
       totalCost AS DistanciaTotal,
       [nodeId IN nodeIds | gds.util.asNode(nodeId).municipio] AS Caminho
ORDER BY index;
```

b) Para listar as cidades conectadas por um determinado tipo de estrada podemos apenas fazer um match, com condicional do tipo de rodovia e retornei o nome das cidades e o tipo de estrada, no meu exemplo a estrada é federal:

```
Unset
MATCH (a)-[r:Trajeto]->(b)
WHERE r.tipo = 'Federal'
RETURN a.municipio, r.tipo, b.municipio;
```

c) Encontrar as cidades que estejam à uma distância máxima definida de uma cidade escolhida, no meu exemplo eu reaproveito a projeção criada na questão A, faço a mesma consulta, no entanto adicionando o condicional de distância máxima maxDist, como 150 KM e determino minha cidade origem como Sorocaba:

```
Unset
WITH 150 as maxDist
MATCH (c1:Cidade {municipio: 'Sorocaba'}), (c2:Cidade)
CALL gds.shortestPath.dijkstra.stream('distanciaEntreCidade', {
       sourceNode: c1,
       targetNodes: c2,
       relationshipWeightProperty: 'distanciaKM'
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
WHERE totalCost <= maxDist
RETURN
       index,
       gds.util.asNode(sourceNode).municipio AS MunicipioOrigem,
       gds.util.asNode(targetNode).municipio AS MunicipioDestino,
       totalCost AS DistanciaTotal,
       [nodeId IN nodeIds | gds.util.asNode(nodeId).municipio] AS Caminho
ORDER BY index;
```

d) Identificar cidades que formam hubs (mais de 3 conexões):

```
Unset
MATCH (a:Cidade)-[r:Trajeto]-(b:Cidade)
WITH a, COUNT(DISTINCT r) AS qtdConexoes
WHERE qtdConexoes > 3
RETURN a.municipio, qtdConexoes;
```

Otimização/Tuning

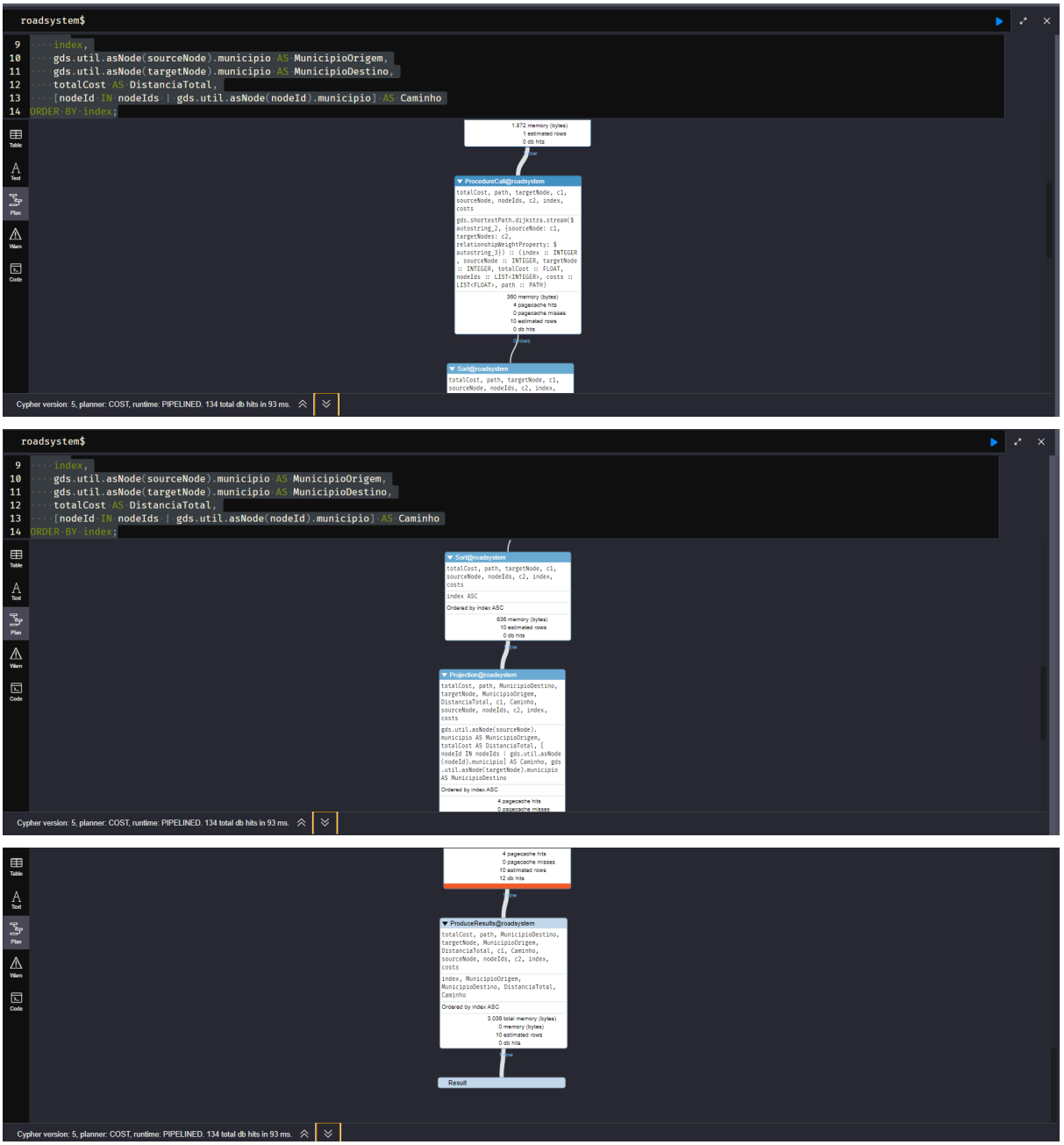
O processo de otimização de um banco deve sempre levar em consideração as características dos dados e a utilização do mesmo. Dessa forma os índices devem ser bem pensados, para evitar gerar gastos desnecessários com atualização de índices nas atualizações de dados no banco.

Dito isso, nosso banco possui algumas propriedades que são candidatos à indexação em diferentes cenários. Um desses cenários é caso estivéssemos utilizando os dados de coordenadas das cidades para consultas, uma vez que nesse caso poderíamos utilizar *point indexes*, que é uma categoria de índice voltada para tal tipo de dado, otimizando consultas sobre essas informações.

No contexto das consultas que fizemos, os principais índices com os quais nos devemos atentar são os de token lookup sobre os rótulos, mas que já são criados por padrão. Além disso, nossas consultas se baseiam muito nos nomes das cidades (propriedade município) dessa forma podemos ganhar performance com indexação de texto nesta propriedade).

Demonstração índice texto na propriedade município:
Para a consulta abaixo, executada com profile, teve antes da criação de um índice, um tempo de execução de 93 ms

```
Unset
PROFILE MATCH (c1:Cidade {municipio: 'Sorocaba'}), (c2:Cidade {municipio:
'Jundiaí'})
CALL gds.shortestPath.dijkstra.stream('distanciaEntreCidade', {
       sourceNode: c1,
       targetNodes: c2,
       relationshipWeightProperty: 'distanciaKM'
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
RETURN
       index,
       gds.util.asNode(sourceNode).municipio AS MunicipioOrigem,
       gds.util.asNode(targetNode).municipio AS MunicipioDestino,
       totalCost AS DistanciaTotal,
       [nodeId IN nodeIds | gds.util.asNode(nodeId).municipio] AS Caminho
ORDER BY index;
```



Então o índice foi criado com o seguinte comando

```
Unset
CREATE TEXT INDEX idx_municipio FOR (c:Cidade) ON c.municipio ;
```

Foi então feita uma segunda execução, dessa vez com um tempo de execução maior de 218 ms (o aumento do tempo de execução nessa primeira execução se deve ao DBMS precisar fazer um novo plano de execução para salvar no cache), no entanto o plano de execução foi diferente (a captura da tela exibe apenas o trecho do plano que foi diferente).

Na terceira execução (segunda após indexação e primeira depois de atualizar o plano de execução) o tempo de execução já cai para 8ms.