UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GUSTAVO LUIS V. CARVALHO

Ambiente de Dados do Sistema de Informação de Agravos de Notificação com Neo4J

RIO DE JANEIRO
2021

GUSTAVO LUIS V. CARVALHO

Ambiente de Dados do Sistema de Informação de Agravos de Notificação com Neo4J

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Profa. Valeria Menezes Bastos
Co-orientador: Profa. Rejane Sobrino Pinheiro

RIO DE JANEIRO
2021

CIP - Catalogação na Publicação

C331a

Carvalho, Gustavo Luis Veloso de
Ambiente de Dados do Sistema de Informação de Agravos de Notificação com Neo4J / Gustavo Luis Veloso de Carvalho. -- Rio de Janeiro, 2021.
66 f.

Orientadora: Valeria Menezes Bastos.
Coorientadora: Rejane Sobriño Pinheiro.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2021.

1. NoSQL. 2. Grafos. 3. ETL. 4. Saúde pública. 5. Neo4J. I. Bastos, Valeria Menezes, orient. II. Pinheiro, Rejane Sobriño, coorient. III. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

GUSTAVO LUIS V. CARVALHO

Ambiente de Dados do Sistema de Informação de Agravos de Notificação com Neo4J

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 13 de Julho de 2021

BANCA EXAMINADORA:

Valeria Menezes Bastos
Valeria Menezes Bastos
D.Sc (Depto de Ciência da Computação - UFRJ)

Participação por videoconferência
Rejane Sobrino Pinheiro
D.Sc (Instituto de Estudos de Saúde Coletiva - UFRJ)

Participação por videoconferência
Adriana Santarosa Vivacqua
D.Sc (Depto de Ciência da Computação - UFRJ)

Participação por videoconferência
Valeria Saraceni
D.Sc (Secretaria Municipal de Saúde do Rio de Janeiro - SMS/RJ)

Dedicatória: A todos os profissionais dedicados da ciência brasileira que buscam por meio da tecnologia e inovação identificar modelos e técnicas capazes não somente prolongar a vida humana assim como também prove-la com maior qualidade.

AGRADECIMENTOS

Agradeço a minha orientadora Valeria Menezes Bastos e a co-orientadora Rejane Sobriño Pinheiro, por toda orientação acadêmica.

A Fundação Bill & Melinda Gates e ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), pelo apoio e incentivo que possibilitou este estudo.

Toda gratidão aos professores e técnicos da UFRJ (Universidade Federal do Rio de Janeiro), em especial do Departamento de Ciência da Computação do Instituto de Matemática , por todo conhecimento passado ao longo da minha formação.

Aos meus colegas de laboratório do LinkDataPop da UFRJ, por toda generosidade em compartilhar informação e conhecimento. Tão importante quanto todos meus amigos e colegas na Universidade que foram fundamentais para esse trabalho. Em especial a Nicole, minha amiga e companheira desde o ínicio das minhas atividades acadêmicas.

"Information is something that can be used to remove uncertainty."

Claude Shannon

RESUMO

O Ministério da Saúde Brasileiro mantém o Sistema de Informação de Agravos de Notificação (SINAN) que tem como objetivo coletar, transmitir e disseminar dados gerados rotineiramente pela Vigilância Epidemiológica. Tais dados são oferecidos em padrões e tipos de arquivos que não permitem uma análise em cima de eventos. O objetivo deste trabalho é realizar uma extração, transformação e carga(ETL) dos dados da base de notificação de tuberculose(SINAN-TB) para um banco de dados em grafos (Neo4J), produzindo um ambiente analítico adequado para a análise de eventos a fim de extrair conhecimento para entendimento e propostas de melhoria em saúde coletiva. Foi escolhido o banco de dados Neo4J, pelas diversas ferramentas já desenvolvidas que facilita a construção e visualização da análise feita. O processo envolve uma série de etapas de conversão dos arquivos originais e conversões e adaptações de um banco relacional para o formato de um banco de dados em grafos ligados. Após o processo de ETL, foram obtidas visualizações de sequência de diagnósticos obtidas por pacientes no tratamento de tuberculose. A metodologia desenvolvida neste trabalho pode ser aplicada em análises de diversos eventos, tanto para área de saúde quanto para outras áreas.

Palavras-chave: Neo4J. ETL. SINAN. Saúde Coletiva. Tuberculose. Informática Aplicada em Saúde.

ABSTRACT

The Brazilian Ministry of Health maintains the Information System for Notifiable Diseases (SINAN) which aims to collect, transmit and disseminate data routinely generated by Epidemiological Surveillance. Such data is available in standards and file formats that do not allow analysis over events. The objective of this work is to perform an Extraction, Transformation and Loading (ETL) of the data from the tuberculosis notifications database (SINAN-TB) to a graph database (Neo4J), producing an analytical environment suitable for the analysis of events to be executed.In order to extract knowledge, understanding and suggest improvements in public health. The Neo4J database was chosen, due to the various tools already developed that facilitate the construction and visualization of the analysis made. The process involves a series of steps to convert the original files, conversions and adaptations of a relational database to the format of a database in linked graphs. After the ETL process, visualizations of the diagnostic sequence obtained by patients in the treatment of tuberculosis were obtained. The methodology can be applied by any analyst interested in obtaining event analysis, either for the health area or any other areas.

Keywords: Neo4J. ETL. SINAN. Public Health. Tuberculosis. Applied Informatics in Public Health.

LISTA DE ILUSTRAÇÕES

Figura 1 – Modelo Baseado na Estrutura SVO 25
Figura 2 – Colunas Base Tubenet 31
Figura 3 – Grafo: Bairro -> Paciente -> Consultas 46
Figura 4 – Busca de Caminhos da Tuberculose Resistente 47
Figura 5 – Diagnósticos com TBDR, imagem relativa a busca do Quadro 20 . . . 48
Figura 6 – Pacientes em até 10 consultas, tiveram diagnósticos em Hospitais, Imagem relativa a busca do Quadro 21 49
Figura 7 – Caminhos com TBDR, imagem relativa a busca do Quadro 22 51
Figura 8 – Caminhos com TBDR em UBS,imagem relativa a busca do Quadro 23 52
Figura 9 – Caminhos com TBDR fora de UBS,imagem relativa a busca do Quadro 24 53
Figura 10 – Tabela agregada com caminhos,imagem relativa a busca do Quadro 25 54

LISTA DE CÓDIGOS

C.1 load.py 64
D.1 steps.py 65
E.1 stepsInNeo2.py 66

LISTA DE TABELAS

Tabela 1 – Completude dos dados para a Base Tubenet de 2012 a 2018 62
Tabela 2 – Domínio dos dados para a Base Tubenet de 2012 a 2018 63

LISTA DE ABREVIATURAS E SIGLAS

OLAP	Online Analytical Processing
ETL	Extract, Transform , Load
SINAN	Sistema de Informação de Agravos de Notificação
IESC-UFRJ	Instituto de Estudos de Saúde Coletiva
SIM	Sistema de Mortalidade
SINASC	Sistema sobre Nascidos Vivos
SIH	Sistema de Informações Hospitalares
SQL	Structured Query Language
NOSQL	Not Only SQL
SGBD	Sistema Gerenciador de Banco de Dados
CSV	Comma Separated value
ACID	Atomicity,Consistency,Isolation,Durability
URI	Uniform Resource Identifier para conexão Cypher
DDL	DataDefinition Language
DML	Data Manipulation Language
SVO	Sujeito-Verbo-Objeto

SUMÁRIO

1 INTRODUÇÃO 14
1.1 Motivação 16
1.2 Objetivo 17
1.3 Trabalhos Correlatos 18
1.4 Estrutura do documento 19

2 ESTADO DA ARTE 20
2.1 Bancos de Dados Relacional 20
2.1.1 PostgreSQL 20
2.1.2 Modelagem no PostgreSQL 21
2.2 Big Data 22
2.2.1 Neo4J 22
2.2.2 Modelando com Neo4J 23
2.2.3 Modelo do problema baseado em Eventos 24
2.3 Python 25
2.4 Pscyopg 26
2.5 Driver oficial Neo4J em Python 26
2.6 Neo4J Desktop 26
2.7 Neo4J Bloom 27
2.8 Pipeline de tratamentos de dados para importação 27

3 IDENTIFICAÇÃO DAS BASES DE DADOS UTILIZADAS . 28
3.1 Base do SINAN 28
3.2 Base do SINAN-TB 29
3.3 Base Tubenet 29
3.4 Base Auxiliar Com Informações de Unidades de Saúde . . . 31
3.5 Completude da Base Tubenet 32
3.6 Domínio da Base Tubenet 32

4 EXTRAÇÃO, TRANSFORMAÇÃO E CARGA (ETL) NO NEO4J 33
4.1 Configurações do Software 33
4.2 Importação do Arquivo Principal no Formato CSV para Servidor Local PostgreSQL 34
4.3 Criação de uma tabela Analítica a partir da tabela primária 34

4.4 Adicionando as variáveis Tipo de Estabelecimento e Tipo de Unidade 37
4.5 Criação de Tabela com Encerramentos possíveis para representar entidade Encerramento no Grafo 37
4.6 Criação de Tabela com Sequência de Eventos de Notificações para representar arestas entre Notificações Sucessivas 38
4.7 Criação de Tabela com Pacientes para representar entidade Pacientes no Grafo 39
4.8 Criação de Tabela Recebe que caracteriza os Eventos de Notificações no Grafo 40
4.9 Conexão com Neo4j e o drive Python 42
4.10 Importação para o Neo4j de Nós Relativos a Pacientes . . 42
4.11 Importação para o Neo4j de Nós Relativos a Encerramentos 43
4.12 Criação das conexões entre os nós da entidade Encerramento 44
4.13 Importação para o Neo4j de Nós Relativos a Notificação de Pacientes e suas Arestas 44
4.14 Importação para o Neo4j de Nós Relativos a Bairros 44
4.15 Importação para o Neo4j de Arestas Relativas a conexão entre Bairros e Paciente 45

5 RESULTADOS 46

6 CONCLUSÃO 56

7 TRABALHOS FUTUROS 58

REFERÊNCIAS 59

APÊNDICE A – COMPLETUDE DA BASE TUBENET **62**

APÊNDICE B – DOMÍNIO DA BASE TUBENET **63**

APÊNDICE C – CLASSE PYTHON LOAD.PY **64**

APÊNDICE D – CLASSE PYTHON STEPS.PY **65**

APÊNDICE E – CLASSE PYTHON STEPSINNEO2.PY **66**

1 INTRODUÇÃO

Grafos são estruturas de dados interessantes, que podem ser usadas para representar modelagens de redes de computadores com múltiplos roteadores e conexões entre eles, sendo os roteadores os nós e as conexões as arestas. Outro exemplo de representação em grafos é a modelagem de redes sociais, onde se tem uma coleção de entidades que participam da rede e se interagem.

Tipicamente, essas entidades são pessoas, e existe pelo menos uma relação entre elas. No Facebook, esse relacionamento é chamado de amizade. O mapeamento das redes sociais por grafos é composto por arestas direcionadas (Twitter) ou não direcionadas (Facebook). Páginas Web são hyperlinks conectados, o que também pode ser considerado outro tipo de grafo.

De fato, em um modelo Entidade-Relacionamento todas as entidades podem ser modeladas como um nó do grafo, e cada relacionamento binário pode ser modelado como uma aresta do grafo. Relacionamentos além dos binários são mais difíceis de se modelar, porém essas relações podem ser consideradas como um conjunto de relações binárias se for necessário.

Nem sempre existe somente uma ou duas relações em modelos de dados mais complexos. Modelar um banco de dados com apenas duas relações é muito simples para um banco orientado a grafos, mas quando temos um modelo mais complexo com cada entidade com seu conjunto de atributos e cada relacionamento com suas propriedades e atributos começamos a pensar em um modelo simplificado para armazenar dados.

Podemos dizer que é mais fácil armazenar os dados em bancos de dados relacionais, mas bancos de dados orientados a grafos, como o Neo4J, um dos mais utilizados, provê ferramentas extras (SILBERSCHATZ; KORTH; SUDARSHAN, 2020):

- Permite que relacionamentos sejam representados no modelo de nós e arestas, além de oferecer uma sintaxe para definir esses relacionamentos.
- Possui uma linguagem própria para expressar busca de caminhos, o que pode ser difícil de ser representado em SQL.
- Provê uma maneira eficiente de escrever as buscas, e pode executar de maneira muito mais veloz do que em uma consulta a um banco de dados relacional.
- Provê ferramentas próprias para visualização dos dados em forma de grafos.

A linguagem de busca utilizada pelo Neo4J é chamada de Cypher, é uma linguagem declarativa inspirada no SQL e que utiliza uma sintaxe ASCII art para representar padrões

de nós e relações no grafo. O Cypher possui conceito de variáveis que representam elementos e parâmetros. A linguagem permite criar, atualizar e remover nós, relacionamentos, rótulos e propriedades, além de gerenciar índices e restrições (CYPHER, 2021).

A maioria das buscas em grafos está interessada apenas em partes específicas de um grafo, começando por ou terminando em um nó específico, o que significa encontrar sub grafos. Isso implica em termos que declarar uma busca na estrutura geral do grafo, esse tipo de busca geralmente é utilizada para encontrar padrões ou transações em tempo real no banco de dados.

Na maioria das vezes quando falamos em grafos estamos buscando padrões globais ou estruturas. A entrada para um algoritmo é normalmente o banco de dados com o grafo inteiro, e a saída pode ser um grafo enriquecido ou com algum valor agregado. Categoriza-se esse processamento por global, e isso implica em termos que processar a estrutura inteira do grafo, o que geralmente requer algoritmos computacionais. Organizações tendem a usar algoritmos e buscas em grafos para modelar sistemas e prever comportamentos baseados em como as informações podem ser disseminadas, componentes importantes, identificação de grupos, e redes de relacionamento.

De maneira geral, operações no grafo inteiro são executadas por algoritmos computacionais e em sub-grafos são feitas através de consultas aos bancos de dados.Porém esses comportamentos, as vezes, podem se sobrepor quando processados por algoritmos usados para responder buscas locais e vice-versa.

Um motivo convincente, para a escolha de um banco de dados em grafos é o simples aumento de performance ao lidar com dados conectados versus bancos de dados relacionais e repositórios NOSQL. Em contraste com os bancos de dados relacionais, onde o desempenho da consulta se deteriora à medida que o conjunto de dados aumenta, com um banco de dados em grafos a complexidade tende a permanecer relativamente constante, mesmo quando o conjunto de dados cresce. Isso ocorre porque as consultas são localizadas em uma parte do grafo. Como resultado, o tempo de execução de cada consulta é proporcional apenas ao tamanho da parte do grafo percorrida para satisfazer essa consulta, em vez do tamanho do grafo geral.

Bancos de dados nos permitem que a estrutura e o esquema surjam em conjunto com a crescente compreensão do espaço do problema, em vez de ser imposta antecipadamente, quando se sabe menos sobre a forma real dos dados. Os bancos de dados em grafos atendem diretamente ao modelo de dados expresso e acomoda as necessidades de negócios de uma maneira que permite que a TI se mova na velocidade dos negócios (ROBINSON; WEBBER; EIFREM, 2015). Os grafos são naturalmente aditivos, o que significa que podemos adicionar novos tipos de relacionamentos, nós, rótulos e sub-grafos para uma estrutura sem perturbar as consultas existentes e a funcionalidade das aplicações.

Essa característica geralmente tem implicações positivas na produtividade do desenvolvedor, mas pode gerar riscos ao projeto. Devido à flexibilidade do modelo em grafo,

não é necessário modelar o domínio com detalhes exaustivos antes do tempo. O aditivo natureza dos grafos também significa que a tendência é realizar menos migrações, reduzindo a sobrecarga e o risco de manutenção.

Além da flexibilidade e da performance, bancos em grafos concedem agilidade, permitindo evoluir o modelo de dados em sintonia com o restante de qualquer aplicação, usando uma tecnologia alinhada com as opções incrementais e práticas iterativas de entrega de software. Particularmente, a natureza livre de esquema do modelo de dados em grafos não é mais uma prática controversa: mais do que o desenvolvimento relacional, o desenvolvimento de bancos de dados grafos alinha bem com as práticas de desenvolvimento de softwares ágeis e orientados a testes, permitindo que aplicativos baseados em banco de dados em grafos evoluam em sintonia com as constantes mudanças dos ambientes de negócios.

1.1 MOTIVAÇÃO

Este trabalho começa a partir de demandas do Instituto de Estudos de Saúde Coletiva da UFRJ (IESC-UFRJ) para pesquisas que desejam observar fenômenos relacionados a tuberculose, em especial, a trajetória das notificações recebidas por pacientes com Tuberculose (TB) e Tuberculose Droga Resistente (TB-DR), e o interesse em entender as particularidades desses pacientes. Acrescenta-se o fato do real interesse em realizar esse estudo em cima das capacidades de bancos de dados em grafos permitindo explorar um volume de dados mais abrangente e suas conexões, gerando resultados nítidos e conclusivos sobre o tratamento de pacientes.

O expressivo aumento do número de casos de tuberculose multirresistente, transforma a doença em um grave problema de saúde pública mundial. Segundo o Guia de Vigilância Epidemiológica TBMR do Ministério da Saúde (2007,p. 14): "A incidência dos casos multirresistentes tem crescido desde a introdução dos primeiros tratamentos para a TB em 1944. A emergência das formas multirresistentes seguiu-se ao uso amplo da rifampicina a partir da década de 70. O número de casos confirmados de TBMR no mundo notificados à Organização Mundial de Saúde (OMS) em 2005 foi de 18.422, destes, 23,8% (4.386/18.422) estavam localizados nas Américas."

Esse trabalho possui uma vasta gama de aplicabilidades em diversas situações, já que com os resultados gerados a partir das notificações, é possível observar informações não evidentes, porém importantes para o tipo de pesquisa realizado.

De acordo com o Ministério da Saúde Brasileiro (Portal do Ministério, 2017), para todos os pacientes notificados com tuberculose no Sistema Único de Saúde (SUS) é gerada uma entrada no Sistema de Informação de Agravos de Notificação (SINAN-TB) que é implementado sobre um banco de dados relacional.

Este sistema permite a realização do diagnóstico dinâmico da ocorrência de um evento

na população, podendo fornecer subsídios para explicações causais dos agravos de notificação compulsória, além de possibilitar a indicação dos riscos aos quais as pessoas estão sujeitas, contribuindo assim para a vigilância sanitária e para a identificação da realidade epidemiológica de determinada área geográfica.

Os dados são essenciais para a tomada de decisões e impulsionam a estratégia de negócios em geral no mundo hoje. A introdução de registros eletrônicos de saúde leva a grandes expectativas a todas as partes interessadas. Serviços que são orientados a dados têm sido amplamente sugeridos como uma solução para evitar aumento nos custos e desperdícios nos sistemas de saúde, cuidados de saúde preventivos e econômicos, bem como a detecção precoce de comportamentos anômalos. A extração de conhecimento pode resultar em diversos benefícios seja na melhora do serviço prestado, seja na qualidade de vida do paciente.

Pesquisas computacionais podem apresentar aos especialistas os detalhes sobre populações vulneráveis para que possam propor tratamentos eficazes para qualquer tipo de doença. Soluções Orientadas a dados fornecem informações que permitem melhores serviços de saúde personalizados, objetivos e econômicos, bem como a detecção precoce de doenças e de comportamentos anômalos.

No entanto, às vezes algumas características complexas e dinâmicas dos dados de saúde impedem essa utilização. Com os dados da SINAN é possível traçar uma trajetória para o paciente com tuberculose e identificar padrões para o tratamento de diferentes comunidades. Com a estrutura correta do modelo proposto é possível responder a perguntas relacionadas a trajetória do paciente, por exemplo.

O ambiente NEO4J foi escolhido para permitir analisar as trajetórias dos pacientes, já que esse tipo de análise de caminhos não é bem suportado pelos bancos de dados SQL tradicionais e NoSQL. Organizar dados em uma estrutura tabular significa fazer várias leituras de tabelas e executar funções computacionalmente complexas de window functions para organizar eventos pelo usuário e, em seguida, sequenciá-los. O mesmo é necessário para bancos de dados que armazenam as informações em documentos, onde é preciso ler muitos arquivos. Além disso, a escolha a esse tipo de banco de dados se dá pela facilidade em visualizar os dados em grafos direto da ferramenta.

1.2 OBJETIVO

O objetivo desse trabalho é exportar os dados de um banco de dados relacional (PostgreSQL) e importar os dados em um banco de dados em grafos (Neo4J), criando um ambiente fácil para consultas de trajetórias com muitas informações para área de saúde, e provendo uma interface de visualização das consultas a serem realizadas.

Deve-se considerar que a base SINAN não foi modelada para permitir uma importação direta para o Neo4J, sendo necessárias algumas etapas auxiliares nesse processo,

transformando os dados do PostgreSQL, de forma eficiente, para o Neo4J.

O presente trabalho desenvolve um modelo em grafos consistente com as análises a serem feitas, transforma as tabelas, exporta os dados do PostgreSQL de maneira adequada para um arquivo em formato csv, o qual é importado para o Neo4J. Com essa solução pode-se explorar algumas capacidades da linguagem Cypher, própria para consultas ao banco, para obter resultados analíticos.

Dessa forma, o trabalho pretende demonstrar a capacidade e eficiência ao se buscar padrões em trajetórias e permitir uma nova classe de análise para trajetórias de usuários. Na maioria das vezes, análises no estilo OLAP permitem efetuar diversas combinações e dimensões de métricas para pacientes, notificações, unidades de saúde e outras entidades. Entretanto, quando são feitas análises sobre eventos, frequentemente queremos entender a sequência em que os eventos ocorrem, por exemplo:

- Quanto tempo demora para um paciente sair de um diagnóstico de abandono e chegar a um diagnóstico de Tuberculose Droga Resistente ?
- Quais são os diferentes caminhos que um paciente tem até chegar a um diagnóstico de Tuberculose Droga Resistente ?
- Quais são os diferentes caminhos que um paciente chega a um diagnóstico de Cura?

Os bancos de dados orientados a grafos representam uma nova abordagem para armazenar e consultar os dados, por isso, é possível utilizá-los para tentar responder algumas das perguntas acima. O trabalho pretende abordar conceitos em bancos de dados orientados a grafos e explorar as capacidades de busca do Neo4J, em particular.

1.3 TRABALHOS CORRELATOS

No trabalho (SNOWPLOW, 2021) é citada uma possibilidade da análise de caminhos para investigação do comportamento de usuários em uma página da internet. Exemplifica a construção de uma gramática para modelagem de um problema no Neo4J e dá exemplo de queries sobre caminhos. O artigo usa clausulas de busca que não são compatíveis com versões modernas da aplicação, porém é bem original e dá bons exemplos do uso do método.

Outro exemplo é o trabalho de (NEO4J, 2021) demonstrando as capacidades do banco de dados e métodos de busca, utilizando funções de agregações e outros exemplos de busca para o Neo4J também em análise de navegação de usuários em uma página da internet.

Outros trabalhos na área de saúde comparando diferentes SGBDS com bancos de dados relacionais são descritos em (SAMPAIO, 2019) e (SANTOS; CARVALHO, 2017) , no primeiro é feita uma comparação de um SGBD relacional com o MongoDB e diferenças no desempenho em consulta e na disponibilidade de dados para diferentes plataformas

dadas pela diferença de formato e diferença no tipo de dado armazenado, no segundo trabalho é usada a ferramenta Pentaho porém com mesmos objetivos, que é criar um ambiente analítico no MongoDB para bases do SUS, nos dois trabalhos foram utilizadas diferentes bases de dados.

1.4 ESTRUTURA DO DOCUMENTO

O trabalho está dividido em seis capítulos: Introdução, Estado da Arte , Identificação das Bases , ETL , Resultados e Conclusão. No capítulo 1 é apresentada a motivação para o presente trabalho, além de descrever brevemente o comportamento dos bancos de dados orientados a grafos e sua relação com bancos relacionais tradicionais. No capítulo 2 são descritas as ferramentas utilizadas no presente trabalho, incluindo linguagem de programação, bibliotecas, funções e sistemas de bancos de dados. O capítulo 3 tenta descrever as bases de dados que foram utilizadas e os campos selecionados. No capítulo 4 é descrito todo o processo de ETL do arquivo em formato csv, derivado de um sistema relacional, para importação no Neo4J. No capítulo 5 é apresentado o banco de dados gerado e os resultados obtidos após o ETL. No capítulo 6 são mostrados os resultados obtidos neste trabalho e sua importância para a pesquisa realizada para a tuberculose.No capítulo 7 é apresentada possíveis trabalhos futuros sobre o tema.

2 ESTADO DA ARTE

Durante todo processo de ETL dos dados disponibilizados pelo SUS para a carga no Neo4J, foi necessário executar os estágios de ingestão, transformação das tabelas, alteração e exclusão de campos, adição de novos campos e conversão de tabelas extraídas do banco relacional para o formato csv, e por fim a importação dos arquivos em formato csv para o banco de dados Neo4J. Nesta seção são descritos os padrões dos arquivos manipulados, e as ferramentas utilizadas no processo.

2.1 BANCOS DE DADOS RELACIONAL

A abordagem relacional foi criada no final da década de 60 por Ted Codd, pesquisador da IBM, que separou os detalhes internos referentes ao desempenho do SGBD dos aspectos funcionais relevantes aos programadores. Um banco de dados relacional é composto de tabelas, ou de maneira formal por relações (HEUSER, 2009).

Uma tabela é um conjunto não ordenado de tuplas onde cada tupla é composta por uma série de campos. Cada campo é identificado por um nome de atributo, que compõe o cabeçalho da tabela. O conjunto de campos homônimos de todas as tuplas de uma tabela formam uma coluna. Em relação as diferenças entre uma tabela de banco de dados e um arquivo digital convencional, Heuser(2009, p. 121) destaca as seguintes diferenças:

- As linhas de uma tabela não têm ordenação, a ordem de recuperação é arbitrária a menos que se especifique. Em geral os arquivos digitais estão organizados em ordem.
- Os valores de campo de uma tabela são atômicos e monolavorados. Ser atômico significa que o campo não pode ser composto de outros. Ser monolavorado significa que o campo possui um único valor e não um conjunto de valores. Diferente das estruturas de dados padrões das linguagens de programação mais comuns.
- As linguagens de consulta a base de dados relacionais permitem o acesso por quaisquer critérios envolvendo os campos de uma ou mais linhas. Já em arquivos convencionais, para buscar registros com base em valores de seus campos de forma rápida, usualmente é necessário que exista algum tipo de caminho de acesso.

Em geral utilizamos Structured Query Language, ou simplesmente SQL, para realizar consultas em bancos relacionais, disponível em diversos fornecedores de sistemas de gerenciamento de bancos de dados do modelo objeto-relacional, sendo o PostgreSQL (POSTGRESQL, 2021) o escolhido para utilização no desenvolvimento do trabalho.

2.1.1 PostgreSQL

O PostgreSQL é um sistema de banco de dados relacional de objetos de código aberto que usa e estende a linguagem SQL combinada com muitos recursos, que armazenam

e escalam com segurança as cargas de trabalho mais complicadas. As origens do PostgreSQL remontam a 1986 como parte do projeto POSTGRES da Universidade da Califórnia em Berkeley, e tem mais de 30 anos de desenvolvimento ativo na plataforma principal. (POSTGRESQL, 2021) Pode ser executado em todos os principais sistemas operacionais, incluindo Linux, Unix e Windows.

O PostgreSQL é compatível com ACID (conjunto de propriedades de transação em banco de dados - Atomicidade, Consistência, Isolamento e Durabilidade) desde 2001, e possui complementos poderosos como o extensor de banco de dados geoespacial PostGIS. Possui suporte para chaves estrangeiras, associações, triggers e procedimentos armazenados. Além disso, pode ser estendido para o armazenamento de grandes objetos binários, o que inclui imagens, sons e vídeo. Possui interfaces em diversas linguagens Java, Python, C/C++, entre outras. Uma característica a ser destacada é a possibilidade de armazenar grandes quantidades de dados com acesso simultâneo a muitos usuários.

2.1.2 Modelagem no PostgreSQL

Em termos simples, o modelo relacional define um conjunto de relações análogas a tabelas e descreve os relacionamentos ou conexões entre eles para determinar como os dados armazenados neles podem interagir. O uso do modelo relacional em banco de dados representa apenas uma tabela plana e bidimensional, onde podemos descrever os dados de uma maneira mais complexa e detalhada, ajudando a reduzir dados duplicados.

O modelo de banco de dados relacional foi um enorme salto em relação aos modelos de bancos de dados anteriores. O modelo permite que qualquer arquivo esteja relacionado a qualquer outro por meio de um campo comum, denominado chave. Desta forma a complexidade do design foi bastante reduzida, pois era possível fazer alterações no esquema do banco de dados sem afetar o acesso aos dados. E como o acesso não era por meio de caminhos para arquivos, mas de um relacionamento direto entre arquivos, novas relações entre esses arquivos podiam ser facilmente adicionadas.

Em 1970, quando Codd desenvolveu o modelo relacional, considerou-se impraticável. A maior facilidade de uso do modelo trouxe uma grande penalidade de desempenho e o hardware naquela época não era capaz de implementar o modelo. Desde então, o hardware avançou bastante até hoje, mesmo os computadores mais simples podem executar sistemas sofisticados de gerenciamento de banco de dados relacional.(MARIADB, 2021)

Os bancos de dados relacionais tiveram um desenvolvimento muito relacionado ao desenvolvimento da linguagem SQL. A simplicidade do SQL - onde podemos aprender a executar consultas básicas em um curto período de tempo em grande parte é o motivo da popularidade do modelo relacional.

2.2 BIG DATA

Aplicativos modernos de gerenciamento de dados geralmente precisam lidar com dados que não estão necessariamente em forma relacional. O Big Data contrasta com os bancos de dados relacionais nos seguintes fatores (SILBERSCHATZ; KORTH; SUDARSHAN, 2020):

- Volume: a quantidade de dados a serem armazenados e processados é muito maior que bancos de dados tradicionais.
- Velocidade: a taxa de chegada de dados é muito maior no mundo em rede de hoje do que nos dias anteriores. Os sistemas de gerenciamento de dados devem poder ingerir e armazenar dados a taxas muito altas.
- Variedade: a representação relacional de dados, linguagens de consulta relacional e sistemas de banco de dados têm tido muito sucesso nas últimas décadas, e eles formam o núcleo da representação de dados da maioria das organizações. Contudo, claramente, nem todos os dados são relacionais.

Embora muitos dados de hoje possam ser representados com eficiência de forma relacional, existem muitas fontes de dados que possuem outras formas de representação, como dados semiestruturados, dados textuais e dados em grafos. A linguagem de consulta SQL é adequada para especificar uma variedade de consultas sobre dados relacionais e foi estendida para lidar com dados semiestruturados. No entanto, muitas consultas não podem ser facilmente expressas em SQL ou avaliadas eficientemente se representadas usando SQL.

Desse modo, uma nova geração de linguagens para consulta e ferramentas foram desenvolvidas por especialistas ao longo dos anos para lidar com essa complexidade crescente e para executar consultas mais complexas sobre um novo modelo de dados.

2.2.1 Neo4J

O Neo4j (NEO4J, 2021) é um banco de dados de grafos nativo de código aberto, que fornece uma estrutura de suporte para aplicações transacionais compatível com ACID. O código-fonte é escrito em Java e Scala, e disponível gratuitamente no GitHub ou como um download de aplicativo. O Neo4j é referido como um banco de dados em grafos nativo porque implementa com eficiência o modelo de grafos até o nível de armazenamento. Isso significa que os dados são armazenados exatamente como modelados, usando ponteiros para navegar e percorrer o grafo. Ao contrário do processamento de grafos ou de bibliotecas em memória, o Neo4j também fornece características completas de banco de dados, incluindo conformidade com transações ACID, suporte a cluster e tolerância a falhas em tempo de execução, o que pode também torná-lo adequado para o desenvolvimento de aplicações.

O Neo4J além de contar com uma linguagem própria para consulta, nos garante flexibilidade, suporta consultas em tempo constante e possui diversos drives para integração com variadas ferramentas, sendo algumas das suas ferramentas listadas abaixo:

- Cypher, linguagem de consulta declarativa semelhante ao SQL, mas otimizada para grafos. Também usado por outros bancos de dados, como o SAP HANA Graph e Redis, por meio do projeto openCypher.
- Buscas em tempo constante para diferentes tamanhos de grafo, tanto em profundidade quanto em largura, devido à representação eficiente de nós e relacionamentos. Permite expansão para bilhões de nós em hardware moderado.
- Drivers para linguagens de programação populares, incluindo Java, JavaScript, .NET, Python, entre outras.

2.2.2 Modelando com Neo4J

A modelagem de dados em grafos é um processo no qual se descreve um domínio arbitrário como um grafo conectado de nós e relacionamentos com propriedades e rótulos. Um modelo de dados em grafos do Neo4j é projetado para responder perguntas na forma de consultas Cypher e resolver problemas técnicos e de negócios, organizando uma estrutura de dados compatível para grafos.

Nós e relacionamentos formam as unidades fundamentais de um grafo, os nós em geral representam entidades e podem conter propriedades no formato de pares chave-valor. Os nós podem assumir papéis ou tipos usando um ou mais rótulos. Os rótulos, por sua vez, servem para categorizar ou agrupar os nós. É usado para criar grupos de nós, sendo os nós com mesmo rótulo considerados da mesma categoria.

Em geral, os bancos de dados em grafos são projetados para procurar nesses grupos ao invés de ter que pesquisar por todo grafo, o que torna as consultas mais eficientes e mais fáceis de escrever. Um nó pode ser rotulado por um número qualquer de rótulos, incluindo nenhum, o que torna opcional na construção em grafos.

Um relacionamento conecta dois nós e permite encontrar nós a partir de seus relacionamentos. Possui um nó de origem e um nó de destino que mostra a direção da seta. Embora armazene um relacionamento em uma direção específica, o Neo4j tem desempenho de travessia igual em qualquer direção, para que você possa consultar o relacionamento sem especificar a direção. Porém uma regra fundamental para construir relacionamentos é que não podemos ter um relacionamento que não termina em um ponto específico, assim quando um nó é excluído é necessário excluir também seus relacionamentos.

Propriedades são pares de dados chave-valor que são armazenadas em nós ou em relacionamentos. A maioria dos tipos de dados padrão é suportada como propriedades, isso permite armazenar dados relevantes sobre o nó ou relacionamento com a entidade que

ele descreve. Eles geralmente podem ser encontrados sabendo quais tipos de perguntas seu caso de uso precisa fazer com seus dados.

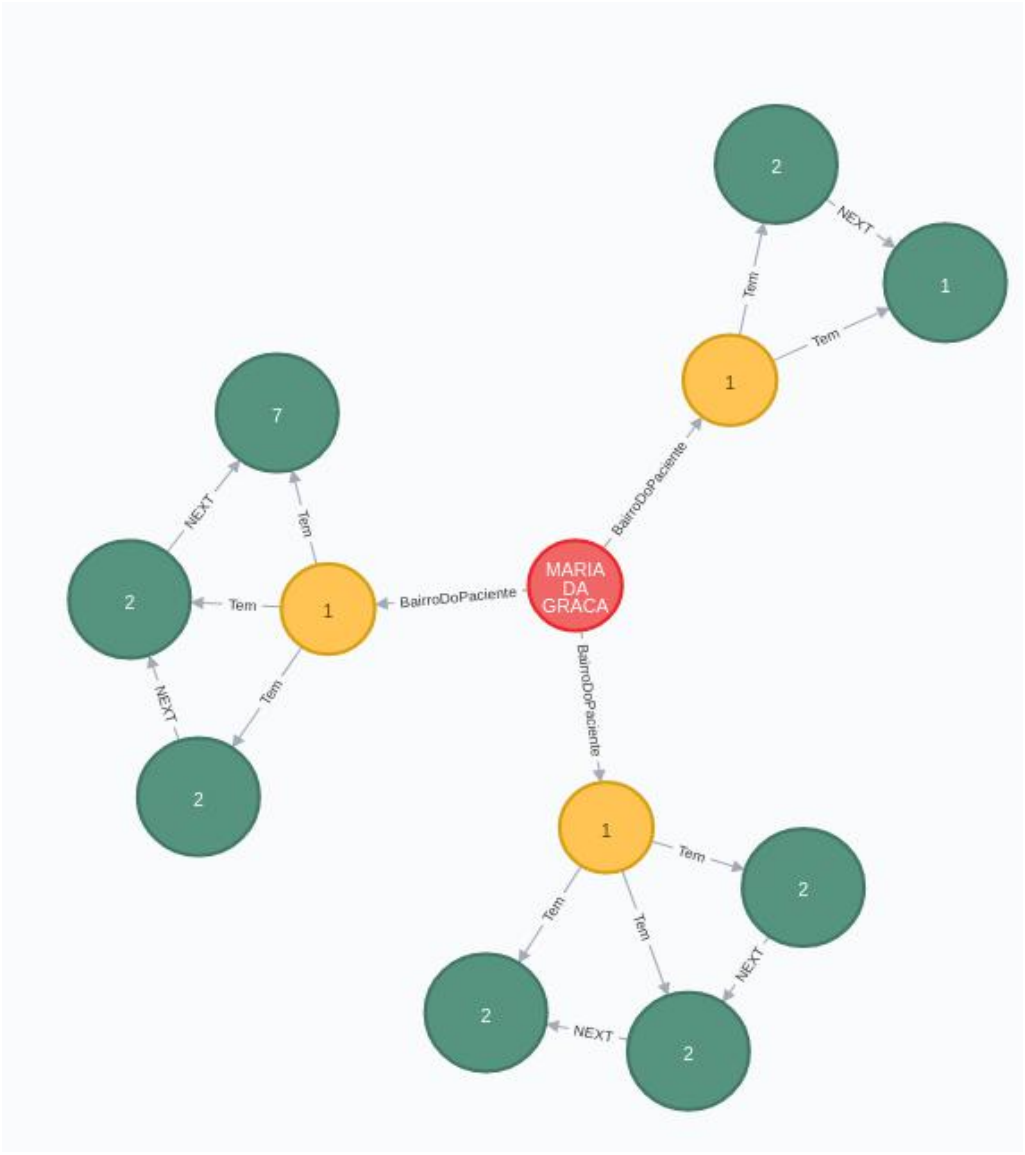
2.2.3 Modelo do problema baseado em Eventos

O trabalho pretende descrever a sequência de notificações do paciente em tratamento da tuberculose. Devido a complexidade do problema foi escolhido o modelo baseado em grafos para traçar uma trajetória. O modelo consiste em pacientes, encerramentos e bairros. Para obter a trajetória do paciente é fundamental entender a ordem dos eventos, o modelo mais simples contaria com um paciente com aresta para todas as suas notificações, porém isso não permitiria uma análise em cima da trajetória.

Portanto, além das arestas que ligam paciente a notificações, também é necessário arestas que ligam notificações a notificações criando então um modelo sequencial para notificações. O modelo é baseado na própria gramática onde temos uma estrutura linguística de Sujeito-Verbo-Objeto(SVO) na qual o sujeito surge como a primeira instância da oração, seguido de verbo e objeto, juntamente com a ordem Sujeito-Objeto-Verbo, a ordem SVO é uma das duas mais comuns, e as duas incluem mais de 75% das línguas do mundo (CRYSTAL, 1997) sugerindo que pode ser a mais simples para psicologia humana. Um modelo da gramática utilizada é exemplificado na figura 1.

Como exemplo de linguagens SVO podemos citar (Inglês , Francês , Português).

Figura 1 – Modelo Baseado na Estrutura SVO



2.3 PYTHON

Define-se Python como uma linguagem de programação de uso geral que muitas vezes é utilizada em scripts, mas também comumente definida como uma linguagem de orientação a objetos por oferecer suporte a essa modalidade de programação. Por isso Python é definido de forma mais comum como uma linguagem de programação de uso geral que utiliza conceitos de linguagens procedurais, funcionais e de orientação a objetos, essa definição é talvez a que melhor capture a riqueza e as aplicações da linguagem atualmente.

O uso da linguagem se dá por uma necessidade muito natural de automatizar as tarefas que são realizadas de forma repetitiva, tais como adição de colunas em bancos de dados, transformação de dados, inserção de dados e extração. Neste trabalho utiliza-se a linguagem Python para configuração do ambiente utilizado, isso inclui gerenciamento de acesso ao servidor local com uso de senhas e login, configuração de host local, diretórios

utilizados pelo programa, nome de arquivos e outras configurações que definem nosso ambiente de programação.

Entre outras tarefas em que se mostra necessário a utilização de scripts, está a criação de tabelas, normalização da tabela, retirar duplicadas, exportar arquivos em formato CSV para que seja intercambiável entre os databases, importação de arquivos para os databases do PostgreSQL, assim como nos databases do Neo4J. Além disso toda cadeia de ETL sobre os dados brutos foi feita por programação.

2.4 PSCYOPG

Pscyopg é o mais famoso driver de conexão com o banco de dados PostgreSQL em linguagem Python. Entre suas principais ferramentas é a conexão por threads que podem utilizar uma mesma conexão. Essa sua característica nos permite ter aplicações multi-thread com a possibilidade de termos um número grande de operações INSERT'S e UPDATE'S em simultâneo.

Os dados brutos possuem um formato próprio que exigiu procedimentos de conversão para um formato compatível com o Neo4J. Essas transformações do formato padrão de uma tabela relacional para um formato baseado em rótulos de nós do Neo4J exigiram diversos scripts Python que são executados no PostgreSQL através do Pscyopg.

Essa biblioteca permite acessar e manipular dados no PostgreSQL em linguagem Python, entre as mais diversas funções podemos realizar consultas, manipulação de dados e tarefas rotineiras como criação de tabelas, updates e deletes.

2.5 DRIVER OFICIAL NEO4J EM PYTHON

Python é uma das linguagens que são oficialmente suportadas pelo Neo4J, os drivers de conexão oficiais conectam com os bancos de dados usando um protocolo chamado BOLT. Esse tipo de conexão nos permite criar os nós, arestas e fazer consultas diretamente com linguagem de programação Python otimizando tempo de desenvolvimento e tornando as soluções reutilizáveis.

Através deste drive podemos manipular as bases de dados no Neo4J. Esse acesso nos permite manipular as conexões entre os nós, suas propriedades e rótulos. Além da criação de novos nós, importação de nós e arestas por arquivo CSV, importar as propriedades dos nós, deletar nós, deletar arestas, assim como realizar consultas , e tarefas administrativas como criar índices e atualizar algumas propriedades dos nós.

2.6 NEO4J DESKTOP

Esta ferramenta é uma IDE para desenvolvedores que permite gerenciar vários projetos e bancos de dados locais, além do uso em servidores. Cada um dos bancos de dados

existentes na ferramenta pode ser gerenciado através da própria máquina local, assim como configurações, atualizações esse gerenciamento pode ser feito tanto pela interface gráfica promovida pela ferramenta como pela linha de comando do aplicativo. Além disso a ferramenta nos permite instalar diversos pacotes extras como APOC e o Graph Data Science de forma muito simples.

2.7 NEO4J BLOOM

É uma ferramenta gráfica que permite imprimir o grafo persistente em um banco de dados do Neo4J de maneira visual para nos permitir investigar e explorar o grafo da forma desejada. Isso pode ser útil nas mais variadas perspectivas de negócios, além de nos permitir fazer buscas nos grafos de maneira simplificada sendo útil para desenvolvedores iniciantes. Essa forma de visualização ainda pode permitir uma maior integração entre o time de desenvolvimento e de análise pela forma visual gerada pela ferramenta.

2.8 PIPELINE DE TRATAMENTOS DE DADOS PARA IMPORTAÇÃO

A forma como os dados são armazenadas em um banco de dados em grafos, é através de conexões e propriedades de arestas e nós. Desta forma é necessário seguir uma rotina de tarefas, que prepare os dados para adaptação ao novo modelo. Algumas das tarefas realizadas:

- LEAD - Essa função é utilizada para acessar uma linha que é sucessora da linha especificada por um argumento. Isso significa que a próxima linha também poderá acessar dados da próxima e assim sucessivamente. Esta função é bem útil quando queremos comparar os valores entre as linhas. No caso utilizamos essa função para organizarmos uma sequência de datas de tratamento de pacientes organizadas por ID.

- LAG – Essa função é utilizada para acessar uma linha que é predecessora da linha especificada por um argumento. Isso significa que a linha anterior também poderá acessar dados da anterior e assim sucessivamente. Esta função é bem útil quando queremos comparar os valores entre a linha atual e as suas predecessoras.

- LPAD – Essa função pode ser utilizada para preencher uma determinada string com uma sub-string em um determinado comprimento. Útil no trabalho para tornar os formatos de datas e identificadores intercambiáveis entre si.

3 IDENTIFICAÇÃO DAS BASES DE DADOS UTILIZADAS

O Sistema Único de Saúde (SUS) é o sistema público de saúde brasileiro criado pela Constituição Federal de 1988 (SAÚDE, 2021a). O SUS é formado pelo conjunto de todas as ações e serviços de saúde prestados por órgãos e instituições públicas federais, estaduais e municipais, da administração direta e indireta e das fundações mantidas pelo Poder Público (SAÚDE, 2021b).

O SUS atua: na assistência em saúde aos indivíduos, por meio de serviços de saúde de baixa, média e alta complexidade; na vigilância epidemiológica e sanitária; assistência farmacêutica; atenção hospitalar; serviços de urgência e emergência; distribuição gratuita de medicamentos e pesquisas na área da saúde (REZENDE, 2021). Os estudos sobre problemas de Saúde Pública e sobre resultados de ações em saúde requerem dados e abordagens metodológicas com certo nível de complexidade. Costuma-se afirmar que há muitos dados, porém pouca informação para apoiar os processos decisórios em saúde pública. O Brasil conta com diversos sistemas de informação em saúde, destacando-se os principais a seguir.

SINASC (Sistema sobre Nascidos Vivos) e SIM (Sistema de Mortalidade) explorado no trabalho de (SANTOS; CARVALHO, 2017) e SIH (Sistema de Informações Hospitalares) explorada no trabalho de (SAMPAIO, 2019). No presente trabalho, buscamos explorar as bases de dados SINAN (Sistema de Informação de Agravos de Notificação) para tuberculose também explorado previamente no trabalho de (ROCHA; BARTHOLOMAY et al., 2019) porém além de uma visão do sistema o presente trabalho pretende encontrar padrões nas ligações entre os dados.

3.1 BASE DO SINAN

No trabalho de (ROCHA; BARTHOLOMAY et al., 2019) , é descrito a base da seguinte forma:

> Esse sistema de informação em saúde foi desenvolvido com o objetivo de padronizar a coleta e o processamento dos dados sobre doenças e agravos de notificação em todo o território brasileiro, disponibilizando informações para análise do perfil da morbidade dos residentes, de forma a contribuir com a tomada de decisão nas esferas municipal, estadual e federal. O sistema também tem como objetivos realizar o monitoramento da saúde da população e prever a ocorrência de eventos, identificar a realidade epidemiológica de determinada área geográfica e auxiliar o planejamento em Saúde, a definição de prioridades de intervenção e a avaliação do impacto das ações de controle desenvolvidas.

O SINAN é responsável pela notificação, e no caso de doenças transmissíveis para acompanhamento do tratamento. As ocorrências de doenças e agravos são definidos por uma Lista Nacional definida por especialistas. A notificação é feita por profissionais atuantes em todas esferas do sistema de saúde, do mais baixo ao mais alto escalão. O

sistema do tipo relacional permite análises em dimensões de tempo e espaço, permitindo identificar tendências e padrões de doenças.

O download da base SINAN sem identificação e da respectiva documentação estão disponíveis no portal do DATASUS em: http://www2.datasus.gov.br/DATASUS/index.php. Navegando em Início > Serviços > Transferência/Download de Arquivos > Arquivos de Dados > SINAN – DADOS FINAIS. As bases são disponibilizadas no portal mensalmente e em formato dbc não explorado neste trabalho.

Na mesma página também é possível fazer o download de uma documentação geral que vem em formato .zip, a documentação é composta por um dicionário de dados, com os atributos do sistema e informa o tipo de variável, além de uma descrição do atributo.

Para solicitação de dados nacionais com identificação nominal, os interessados devem seguir o fluxo definido no sítio eletrônico da Secretaria de Vigilância em Saúde do Ministério da Saúde (SVS/MS): http://portalms.saude.gov.br/vigilancia-em-saude Dados estaduais e municipais devem ser solicitados às respectivas secretarias.

3.2 BASE DO SINAN-TB

Para cada doença que consta na lista de notificação compulsória existe uma base de dados diferente com as variáveis específicas daquela doença no sistema SINAN, no presente trabalho buscamos explorar a base de dados do SINAN-TB. A ficha de notificação/investigação de tuberculose deve ser utilizada para notificar todas as pessoas com TB ativa, ou seja, os casos novos, as recidivas, os reingressos após abandono e as transferências. As pessoas que nunca foram registradas no SINAN e foram descobertas após a morte, em decorrência da realização de investigação epidemiológica, devem ser notificados com o tipo de entrada 'pós-óbito'. (ROCHA; BARTHOLOMAY et al., 2019)

A ficha individual é composta por duas partes: (i) a notificação, incluindo dados de identificação da pessoa notificada e dados sobre a unidade notificadora, com campos padronizados para todas as doenças e agravos; e (ii) a investigação, contemplando o tipo de entrada da pessoa no sistema, forma clínica, características individuais (populações vulneráveis; e agravos e doenças associadas), presença de coinfecção TB-HIV e resultados de exames, entre outros.(ROCHA; BARTHOLOMAY et al., 2019)

3.3 BASE TUBENET

A Secretaria Municipal de Saúde (SMS) faz a vinculação dos dados, a partir de um procedimento no Sistema de Informação SINAN.

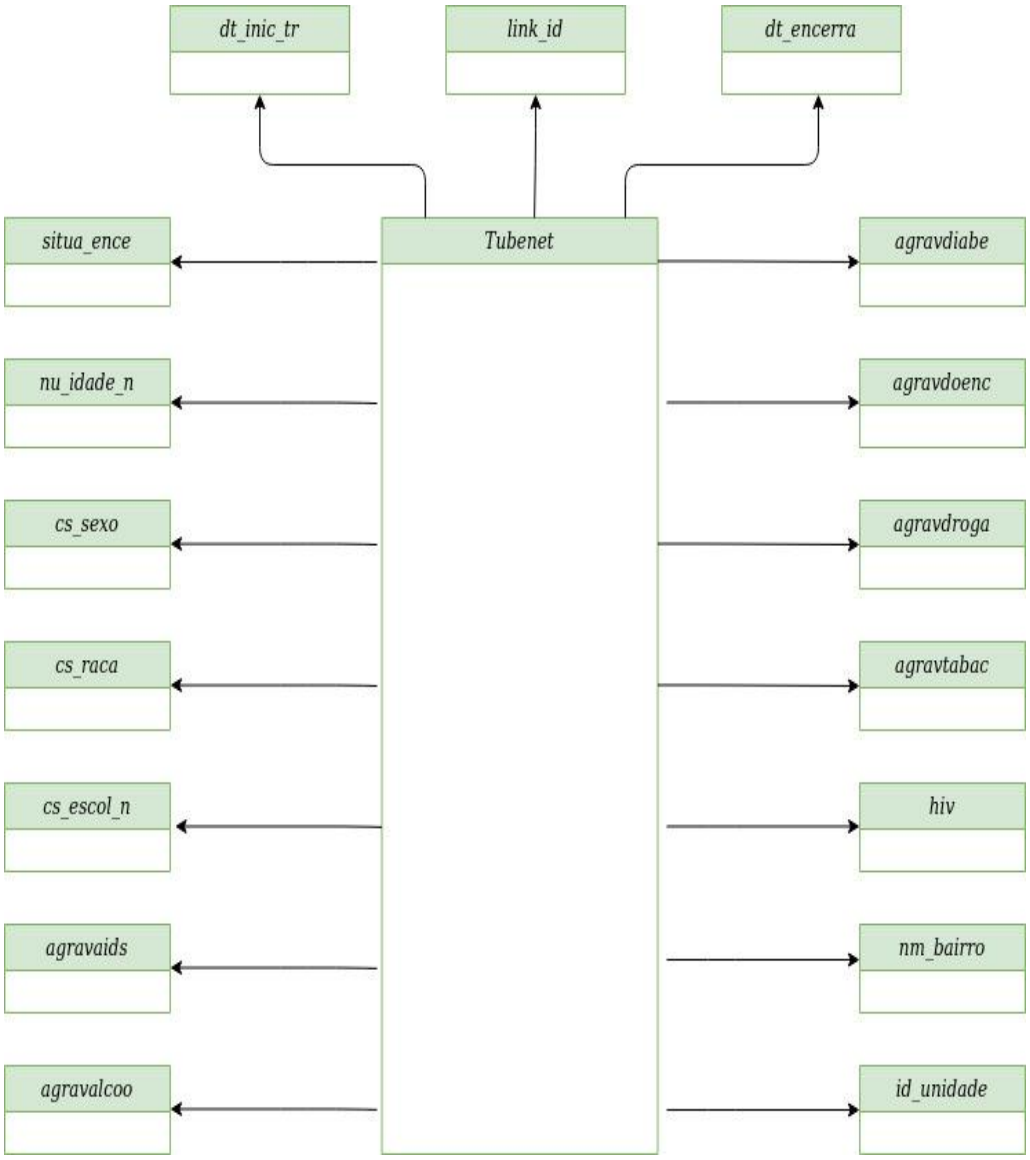
Esse procedimento une os registros que tem situação de encerramento por transferência com os que tem entrada por transferência, por que são considerados o mesmo paciente. A 1ª parte do registro refere-se à notificação do caso, onde constam dados do paciente e

dados relativos a primeira entrada do paciente e a 2ª parte do registro contém os dados de acompanhamento do caso.

Os registros que são vinculados têm unidade de notificação diferente da unidade de tratamento, porém os dados são providos pelo mesmo sistema o SINAN. Os registros para chegarem na forma apresentada nesta base passam por um procedimento de vinculação na Secretaria Municipal de Saúde (SMS) e após esse procedimento a base é descrita por este nome Tubenet e o presente trabalho se desenvolve com estes registros. Para trabalharmos com os indivíduos reincidentes da forma que trabalhamos, necessitamos de um processo de linkage, ou seja, um processo de deduplicação com os dados da SINAN. Com este processo é criado e incluído variáveis à base de dados.

Considerando a base original temos 164 colunas, e 111636 registros de notificações entre o ano de Janeiro de 2005 e Junho de 2019, dessas colunas retiramos apenas as colunas importantes para a análise. Destacando aqui as colunas utilizadas para análise de caminhos na base Tubenet em um modelo conceitual:

Figura 2 – Colunas Base Tubenet



3.4 BASE AUXILIAR COM INFORMAÇÕES DE UNIDADES DE SAÚDE

Uma tabela auxiliar foi necessária durante o processo de ETL para identificar o tipo de estabelecimentos de unidade de saúde em nossa tabela analítica. A tabela em específico é chamada de tpTipoEstabelecimento.csv , e tem o formato de arquivo csv.

Esta tabela faz um de-para entre códigos de tipo de estabelecimento em saúde originados do SINAN para nome de tipos de Unidade de Saúde. O seu download pode ser encontrado em http://cnes.datasus.gov.br/pages/downloads/arquivosBaseDados.jsp site oficial do DataSus.

3.5 COMPLETUDE DA BASE TUBENET

Foi feita uma análise de completude de todos os campos da base Tubenet. A completude é expressa em percentagem unitária de registros preenchidos. O resultado se encontra no apêndice A.

3.6 DOMÍNIO DA BASE TUBENET

Foi feita uma análise do domínio de todos os campos relevantes da base Tubenet. O domínio é expresso pelos valores que cada atributo pode assumir. Todos os campos foram interpretados como strings. O resultado se encontra no apêndice B.

4 EXTRAÇÃO, TRANSFORMAÇÃO E CARGA (ETL) NO NEO4J

Neste trabalho foi elaborado um software para lidar com as diferenças do modelo relacional para um modelo em grafo, e uma automatização para tarefas executadas várias vezes. Dessa forma, foi criada uma rotina principal que executa diversos scripts que compõem as tarefas necessárias para extrair os dados de um formato para o banco desejado. Esse processo é chamado de pipeline de dados.

Com bancos orientados a grafos tem-se a possibilidade de analisar dados de eventos sob uma nova perspectiva, especialmente quando o interesse é entender a sequência em que esses eventos ocorrem. Pretende-se com esse processo de ETL construir um modelo em grafo que permita a análise de caminhos a partir de eventos de notificações em tuberculose.

4.1 CONFIGURAÇÕES DO SOFTWARE

No software desenvolvido, foi passado um arquivo texto com algumas configurações, que, ao início da rotina principal, inicializa algumas definições de ambiente para posterior execução do script. Existem várias definições permitidas, porém, as que foram utilizadas estão descritas abaixo:

- DIRETORIO_BASE , diretório onde se encontram os arquivos base do software.
- CONFIGURACOES_DIR, diretório onde se encontram as configurações do software.
- INBOX_DIR, diretório em que arquivos com dados de entrada para o software são armazenados.
- OUTBOX_DIR, diretório em que arquivos com dados de saída para o software são armazenados.
- SQL_DIR, diretório com scripts em SQL.
- CYPHER_DIR, diretório com scripts em linguagem Cypher.
- CSV_FILE, nome do arquivo csv principal para entrada de dados.
- TABLENAME, nome da tabela primária para entrada de dados.
- HOSTNAME, nome de entrada para conexão com o servidor PostgreSQL.
- PORT, porta para conexão com o servidor PostgreSQL.
- USERNAME, nome do usuário de conexão com o servidor PostgreSQL.

- PASSWORD, senha do usuário de conexão com o servidor PostgreSQL.
- URI, Uniform Resource Identifier para conexão Cypher.
- AuthLogin, login para conexão com o servidor Neo4j.
- AuthPass, senha para conexão com o servidor Neo4j.
- CSVFILEAUX, nome de arquivo auxiliar primário para dados auxiliares.
- TABLENAMEAUX, nome de tabela auxiliar primária para armazenar dados auxiliares.

4.2 IMPORTAÇÃO DO ARQUIVO PRINCIPAL NO FORMATO CSV PARA SERVIDOR LOCAL POSTGRESQL

O formato padrão csv do arquivo com dados da base Tubenet dificulta a manipulação de dados e a uniformização com o formato de dados em grafos. Para uniformizar o modelo relacional com o modelo em grafos foi necessário carregar a base principal no servidor PostgreSQL e, em seguida, fazer a importação para o Neo4J.

Essa etapa no processo de ETL, foi executada com uma classe em Python chamada load.py, apresentada no apêndice C, que importa as seguintes bibliotecas da linguagem Python: psycopg2 e sys. Essa classe contém um método chamado pg_load_table que recebe como parâmetros o arquivo de configuração inicial, um arquivo em formato CSV, e o nome da tabela no servidor PostgreSQL.

Esse método é responsável por estabelecer uma conexão com servidor PostgreSQL, a partir do método connect da biblioteca psycopg2. Com a conexão estabelecida é possível utilizar a linguagem SQL dentro do código em Python para executar as tarefas. Nesta etapa em específico são retirados quaisquer dados que existam na tabela e todos os dados que existirem no arquivo CSV são carregados. Para a execução deste método é necessário criar a tabela previamente com o mesmo número de colunas que existe no arquivo CSV. As tabelas também devem estar padronizadas quanto ao formato dos campos. Neste trabalho todos os campos foram tratados com formatação String para dar flexibilidade ao processo de importação e extração das bases com diferentes bancos de dados.

4.3 CRIAÇÃO DE UMA TABELA ANALÍTICA A PARTIR DA TABELA PRIMÁRIA

Neste passo foi criada uma tabela analítica no PostgreSQL, através da classe "steps", desenvolvida em Python, que se encontra no apêndice D. A classe steps tem um único método chamado step, responsável por estabelecer uma conexão com servidor PostgreSQL

a partir do método connect citado anteriormente, e executar um script SQL localizado no diretório SQL_DIR do ambiente utilizado, executado dentro do servidor PostgreSQL.

No script de criação da tabela analítica são executados comandos para DDL (Data Definition Language) - CREATE, ALTER e DROP, usados para a criação das tabelas, e comandos para DML (Data Manipulation Language) - INSERT, UPDATE e DELETE, usados para manipular os dados das tabelas.

1º Passo – Drop table

Deleta a tabela analítica se ela existir previamente (Quadro 1). Esse comando é executado para garantir que não restam dados de uma importação anterior a cada nova importação.

```
drop table if exists tbl_tubenet_analitica;
```

Quadro 1

2º Passo – Create Table a partir da Tabela primária

Nesta etapa é criada a tabela analítica a partir da tabela primária anteriormente citada. O interesse principal para análise de caminhos de pacientes com tuberculose era entender a situação de cada paciente a cada nova notificação compulsória no Sistema SINAN. Para isso foram excluídos os registros cujas situações de encerramento não foram registradas, ou seja, a cláusula where situa_ence != 'NA' não considera os registros cujo atributo situa_ence tenha o valor "NA".

Para compor a trajetória dos pacientes no decorrer do tempo foi necessário identificar quem são esses pacientes. Na rotina de vinculação descrita anteriormente, os pacientes recebem um link_id único e para os casos em que o paciente tem apenas uma notificação ele recebe uma string 'NA'. Esta situação também foi eliminada, pois não seria possível estabelecer uma trajetória com mais de um passo para o paciente que tem uma entrada única (considerado como caso trivial), e sendo a motivação principal deste trabalho os pacientes com uma trajetória maior que um passo.

A população restrita de liberdade também não foi considerada neste trabalho, pois poderia criar uma influência nas análises que não reflete a situação geral de todos os pacientes. Dessa forma, foi executado o comando do quadro 2:

```
create table tbl_tubenet_analitica as
(select * from stg_tubenet_anonimo_deduplicado
where situa_ence != 'NA'
and link_id != 'NA' and pop_liber != '1');
```

Quadro 2

3º Passo – Organização da trajetória dos pacientes em ordem cronológica:

A trajetória dos pacientes foram organizadas de acordo com as datas de início de tratamento, e assim, para os casos em que esta data não constava organizamos a sequência de notificações no grafo de cada paciente de acordo com as datas de encerramento de tratamento.

E o caso em que as duas datas estavam sem serem preenchidas. O registro foi eliminado da análise.

```
DELETE FROM tbl_tubenet_analitica
WHERE dt_inic_tr = 'NA' ;
```

Quadro 3

Foram considerados os casos a partir de 2012, pois antes desta data os registros no sistema não eram confiáveis, seja por falha no diagnóstico ou falha na inserção dos registros. Foi atribuída a cada paciente uma ordem cronológica após sucessivas notificações, com o comando definido no Quadro 4:

```
alter table tbl_tubenet_analitica
ADD COLUMN id SERIAL PRIMARY KEY;

ALTER TABLE tbl_tubenet_analitica
ADD COLUMN nm_encerra TEXT;

with cte as(
            select
               id,
                row_number() OVER
                (PARTITION BY link_id ORDER BY dt_inic_tr asc)
                as nm_encerra
            from tbl_tubenet_analitica
)
update tbl_tubenet_analitica set nm_encerra = cte.nm_encerra
from cte
where cte.id = tbl_tubenet_analitica.id;
```

Quadro 4

4º Passo – Uniformização de registros para Análise:

E por fim, foi atribuída a situação de encerramento 2, considerada abandono de tratamento, nos casos em que a situação de encerramento era 5 (transferência), situação essa onde é considerado que o paciente não continuou o tratamento como esperado.

4.4 ADICIONANDO AS VARIÁVEIS TIPO DE ESTABELECIMENTO E TIPO DE UNIDADE

Foi necessário criar uma tabela auxiliar durante o processo de ETL, para identificar o tipo de estabelecimentos de unidade de saúde na tabela analítica utilizada. A tabela auxiliar é chamada de tpTipoEstabelecimento.csv, e tem o formato de arquivo csv. Esta tabela faz um de-para entre códigos de tipo de estabelecimento em saúde para nome de tipos de Unidade de Saúde. O seu download pode ser encontrado em http://cnes.datasus.gov.br/pages/downloads/arquivosBaseDados.jsp site oficial do DataSus.

De forma resumida, executa-se um passo com o script SQL que realiza um *update* na tabela analítica, conforme apresentado no Quadro 5:

```
UPDATE tbl_tubenet_analitica AS tu
SET aux_tp_estabelecimento = aux.aux_tp_estabelecimento,
aux_tp_unidade = aux.aux_tp_unidade
FROM tbl_aux_unidades AS aux
WHERE tu.tpuninot = aux.aux_tp_estabelecimento ;
```

Quadro 5

4.5 CRIAÇÃO DE TABELA COM ENCERRAMENTOS POSSÍVEIS PARA REPRESENTAR ENTIDADE ENCERRAMENTO NO GRAFO

Os bancos de dados em grafos usam nós para armazenar entidades de dados e arestas para armazenar relacionamentos entre entidades. Uma aresta sempre tem um nó inicial, um nó final, um tipo e uma direção, e no geral descrevem um relacionamento entre entidades com um verbo, como gostar, pertencer, e relações hierárquicas.

Para criação da entidade Encerramento no banco em grafos foi preciso definir quais tipos de entidade Encerramento existem nos dados. Essa definição é feita criando-se uma tabela para essa entidade, onde cada linha da tabela corresponde a um tipo, que é representada no grafo por um nó no modelo de grafos baseado em *labels*.

Por fim essa definição é feita da seguinte forma:

Executa-se o código SQL do quadro 6 abaixo, para garantir que não existam dados na tabela a ser criada:

```
drop table if exists tbl_encerramento_nodes;
```

Quadro 6

Em seguida, foi criada a tabela "tbl_encerramento_nodes"garantindo que cada tipo diferente da entidade Encerramento existente na tabela analítica seja única, e que faça uma varredura completa nesta tabela.

```
create table tbl_encerramento_nodes as (
select situa_ence from tbl_tubenet_analitica group by 1
);
```

Quadro 7

No último passo é efetuada a extração da tabela "tbl_encerramento_nodes"para um arquivo com formato CSV, para que seja possível criar o nó responsável por representar a entidade e seus respectivos *labels*, responsáveis por representar seus tipos.

Podemos fazer isso com o comando SQL apresentado no Quadro 8:

```
COPY tbl_encerramento_nodes TO '/home/gustavo/data.pop.engine/
data/data.input/tbl_encerramento_nodes.csv'
DELIMITER ',' CSV HEADER;
```

Quadro 8

4.6 CRIAÇÃO DE TABELA COM SEQUÊNCIA DE EVENTOS DE NOTIFICAÇÕES PARA REPRESENTAR ARESTAS ENTRE NOTIFICAÇÕES SUCESSIVAS

Para estabelecer uma sequência cronológica dos eventos precisamos numerar cada um deles de acordo com as datas de início de tratamento para cada paciente. No grafo, essa sequência será representada por arestas com uma origem e com destino. Podemos representar as sequências por uma tabela, com origem e destino.

Esta representação é feita executando um passo no programa principal, com o script SQL do Quadro 9:

```
drop table if exists tbl_proxima_edge;

create table tbl_proxima_edge as (
select * from
(select
id as evento_id,
LEAD(id,1) OVER
(PARTITION BY link_id    ORDER BY dt_inic_tr)
as proximo_evento
from tbl_tubenet_analitica
) as t where proximo_evento is not null);

COPY tbl_proxima_edge TO
'/home/gustavo/data.pop.engine/data/
data.input/tbl_proxima_edge.csv'
DELIMITER ',' CSV        HEADER;
```

Quadro 9

4.7 CRIAÇÃO DE TABELA COM PACIENTES PARA REPRESENTAR ENTIDADE PACIENTES NO GRAFO

Para a criação da entidade Paciente no banco em grafos, foi necessário definir quem são os pacientes. A entidade Paciente existente nos dados, é definida por uma tabela, onde cada linha da tabela representa um Paciente, que é representado no grafo por um nó representativo da entidade Paciente no modelo de grafos baseado em *labels*.

Isso é feito criando-se uma tabela após a escolha das propriedades de interesse para análise pelos especialistas. O script de criação é apresentado no Quadro 10:

```
drop table if exists tbl_paciente_node;


CREATE TABLE tbl_paciente_node as(
select
link_id,
nu_idade_n,
cs_sexo,
cs_raca,
```

```
        cs_escol_n,
        agravaids,
        agravalcoo,
        agravdiabe,
        agravdoenc,
        agravdroga,
        agravtabac,
        hiv,
        nm_bairro
        from tbl_tubenet_analitica
        where nm_encerra = '1'
        );
```

Quadro 10

Estas variáveis foram escolhidas por representarem melhor as características de interesse nas populações pesquisadas.

Por fim, é realizada a extração desta tabela para importação do banco em grafos, com o seguinte comando no script SQL (Quadro 11):

```
COPY tbl_paciente_node TO '/home/gustavo/data.pop.engine/
data/data.input/tbl_paciente_node.csv' DELIMITER ',' CSV HEADER;
```

Quadro 11

4.8 CRIAÇÃO DE TABELA RECEBE QUE CARACTERIZA OS EVENTOS DE NOTIFICAÇÕES NO GRAFO

Para criação de uma sequência de notificações no banco em grafos foi necessário definir quais instâncias da entidade Notificação existem nos dados. Além de ser necessário definir essa entidade também foi preciso definir as arestas que fazem o relacionamento NEXT entre estas.

Essa definição é feita da mesma forma, criando-se uma tabela para essas instâncias da entidade Encerramento, e usando a numeração criada anteriormente para definir onde serão colocadas as arestas que ligam a origem ao destino. Cada linha da tabela representa uma nova notificação, que é representada no grafo por um nó representativo da entidade de notificação e também indica os relacionamentos entre elas, a partir do número gerado para a sequência.

De forma geral, esta tabela é criada com um passo no programa principal, que executa o script SQL do Quadro 12:

```
drop table if exists tbl_recebe_nodes;

create table tbl_recebe_nodes as (select
id as event_id,
link_id,
extract(EPOCH from dt_inic_tr::timestamp) as date_timestamp,
dt_inic_tr,
dt_encerra,
situa_ence,
nu_idade_n,
cs_sexo,
cs_raca,
cs_escol_n,
agravaids,
agravalcoo,
agravdiabe,
agravdoenc,
agravdroga,
agravtabac,
hiv,
nm_bairro,
id_unidade,
aux_tp_estabelecimento,
aux_tp_unidade
from tbl_tubenet_analitica
);

COPY tbl_recebe_nodes TO '/home/gustavo/
data.pop.engine/data/
data.input/tbl_recebe_nodes.csv' DELIMITER ',' CSV HEADER;
```

Quadro 12

4.9 CONEXÃO COM NEO4J E O DRIVE PYTHON

Para conectividade com o banco de dados Neo4j, através da linguagem Python utiliza-se a biblioteca oficial da empresa. A biblioteca neo4j contém um drive que permite executar comandos na linguagem Cypher e outras funções administrativas dentro do banco.

Foi desenvolvida uma classe StepsInNeo.py, presente no trabalho no apêndice E, com um único método loadStep, que tem como funções abrir uma conexão com o banco de dados, criar uma sessão e executar scripts na linguagem Cypher. Essa classe recebe com parâmetro a classe com as configurações de execução, citadas no item 4.1, e os nomes de scripts para execução.

Essa conexão é fundamental para automatizar a construção do banco de dados para os tipos de análise realizadas neste trabalho.

4.10 IMPORTAÇÃO PARA O NEO4J DE NÓS RELATIVOS A PACIENTES

A rotina principal chama o método loadStep da classe StepsInNeo.py, responsável por executar o script na linguagem Cypher (Quadro 13):

```
        USING PERIODIC COMMIT
        LOAD CSV WITH HEADERS FROM
        "file:/home/gustavo/
data.pop.engine/data/data.input/
tbl_paciente_node.csv" AS row
        CREATE (:Paciente {
        link_id : row.link_id,
        nu_idade_n: row.nu_idade_n,
        cs_sexo:row.cs_sexo,
        cs_raca: row.cs_raca,
        cs_escol_n: row.cs_escol_n,
        agravaids:      row.agravaids,
        agravalcoo: row.agravalcoo,
        agravdiabe: row.agravdiabe,
        agravdoenc:     row.agravdoenc,
        agravdroga: row.agravdroga,
        agravtabac: row.agravtabac,
        hiv:    row.hiv,
        nm_bairro: row.nm_bairro});
```

Quadro 13

Esse script é responsável por criar todos os nós representativos da entidade Paciente no banco de dados em grafos, fundamental para representar o caminho de cada um deles. Esses nós são criados a partir do csv extraído da tabela de nós dos pacientes (item 4.7).

4.11 IMPORTAÇÃO PARA O NEO4J DE NÓS RELATIVOS A ENCERRAMENTOS

Da mesma forma, a rotina principal executa o método loadStep para executar o script de criação dos nós representativos da entidade Encerramento, que são extraídos da tabela de Notificações (Quadro 14).

```
        USING PERIODIC COMMIT
        LOAD CSV WITH HEADERS FROM "file:/home/gustavo/
data.pop.engine/data/data.input/
tbl_recebe_nodes.csv" AS row
        CREATE (e:Encerramento {
        event_id: row.event_id,link_id: row.link_id,
        date_timestamp: row.date_timestamp,
        dt_inic_tr: row.dt_inic_tr,
        dt_encerra: row.dt_encerra,
        dt_notific: row.dt_notific,
        situa_ence: row.situa_ence,nu_idade_n: row.nu_idade_n,
        cs_sexo: row.cs_sexo,cs_raca: row.cs_raca,
        cs_escol_n: row.cs_escol_n,agravaids:row.agravaids,
        agravalcoo: row.agravalcoo,agravdiabe: row.agravdiabe,
        agravdoenc: row.agravdoenc,agravdroga: row.agravdroga,
        agravtabac: row.agravtabac,hiv: row.hiv,
        nm_bairro: row.nm_bairro,id_unidade:row.id_unidade,
        aux_tp_estabelecimento:row.aux_tp_estabelecimento,
        aux_tp_unidade: row.aux_tp_unidade})
```

Quadro 14

Aqui são consideradas todas as características de interesse para análise, onde cada linha é a propriedade de um nó.

4.12 CRIAÇÃO DAS CONEXÕES ENTRE OS NÓS DA ENTIDADE ENCERRAMENTO

Este script é executado para criar as conexões entre os nós que se relacionam. Essa informação foi obtida a partir da tabela proxima_edge criada anteriormente (item 4.6). Com esse tipo de relacionamento foi possível estabelecer as arestas direcionadas do grafo. O script em linguagem Cypher é apresentado no Quadro 15:

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/home/gustavo/
data.pop.engine/data/
data.input/tbl_proxima_edge.csv" AS row
MATCH (novo:Encerramento {event_id: row.proximo_evento}),
(antigo:Encerramento{event_id:row.evento_id})
CREATE (antigo)-[:NEXT]->(novo)
```

Quadro 15

4.13 IMPORTAÇÃO PARA O NEO4J DE NÓS RELATIVOS A NOTIFICAÇÃO DE PACIENTES E SUAS ARESTAS

Além de relacionar os sucessivos encerramentos em ordem cronológica, precisa-se relacionar o paciente e suas notificações, através do script Cypher (Quadro 16), responsável por relacionar os nós dessas duas entidades.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/home/gustavo/
data.pop.engine/
data/data.input/tbl_recebe_nodes.csv" AS row
MATCH (p:Paciente{link_id : row.link_id})
MATCH (e:Encerramento{event_id : row.event_id})
MERGE (p)-[:Tem]->(e)
```

Quadro 16

4.14 IMPORTAÇÃO PARA O NEO4J DE NÓS RELATIVOS A BAIRROS

É importante também definir os nós pertencentes a Entidade Bairro para a análise. Isso é feito extraindo os nós existentes da tabela Bairro:

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/home/gustavo/
data.pop.engine/data/
data.input/tbl_paciente_bairro.csv" AS row
CREATE (b:Bairro {nm_bairro: row.nm_bairro})
```

Quadro 17

4.15 IMPORTAÇÃO PARA O NEO4J DE ARESTAS RELATIVAS A CONEXÃO ENTRE BAIRROS E PACIENTE

Por fim, foi necessário identificar as ligações do Paciente com o Bairro, para executar a análise dos relacionamentos de Bairros e Pacientes. Esses relacionamentos foram construídos com o script em linguagem Cypher, apresentado no Quadro 18:

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:/home/gustavo/
data.pop.engine/data/
data.input/tbl_paciente_bairro.csv" AS row
MATCH (b:Bairro{nm_bairro : row.nm_bairro})
MATCH (p:Paciente{nm_bairro : row.nm_bairro})
CREATE (b)-[:BairroDoPaciente]->(p)
```

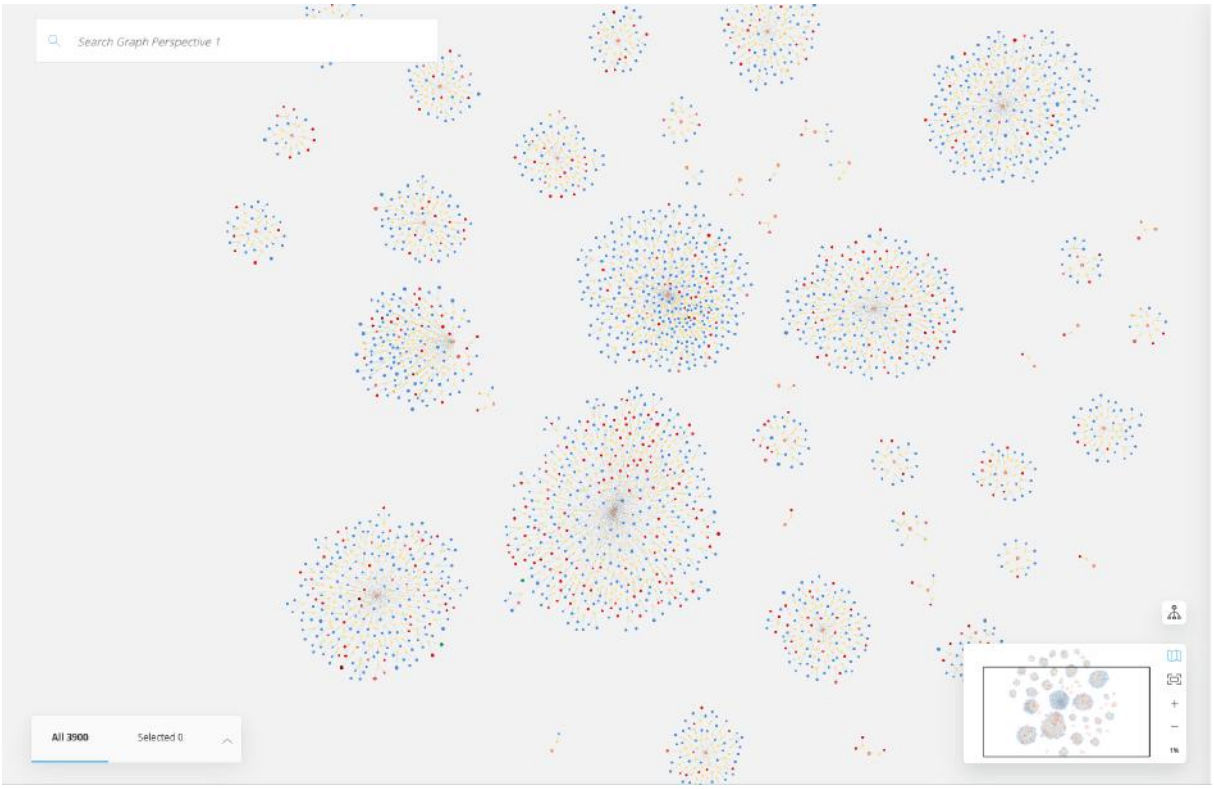
Quadro 18

5 RESULTADOS

Após todos os passos descritos no capítulo anterior, tem-se a tabela Tubenet de 2012 a 2018 carregada no Neo4J de forma automatizada. O procedimento foi realizado com as restrições descritas no capítulo 3, somente com os dados da cidade do Rio de Janeiro e considerando apenas registros vinculados, porém, a metodologia estabelecida pode ser reutilizada para análises futuras com novos conjuntos de dados apenas fazendo pequenas adaptações.

Cada paciente se tornou um nó e todas as notificações recebidas por esse paciente foram também considerados nós. Numa forma mais geral, todos os pacientes estão ligados a um Bairro, e assim todos os Bairros também estão ligados às consultas efetuadas pelos pacientes. Esse cenário é exemplificado na figura 3.

Figura 3 – Grafo: Bairro -> Paciente -> Consultas



Com essa estrutura de grafo foi possível fazer buscas locais de caminhos sem precisar percorrer todo o grafo. Para ilustrar o uso e a utilidade da base obtida, foram realizadas consultas que podem ocorrer em situações práticas, e para este procedimento foi usado o Neo4J Bloom.

Conversando com os analistas, pudemos identificar algumas buscas interessantes, por exemplo, encontrar os tipos de diagnóstico que o paciente recebeu antes de ter um diag-

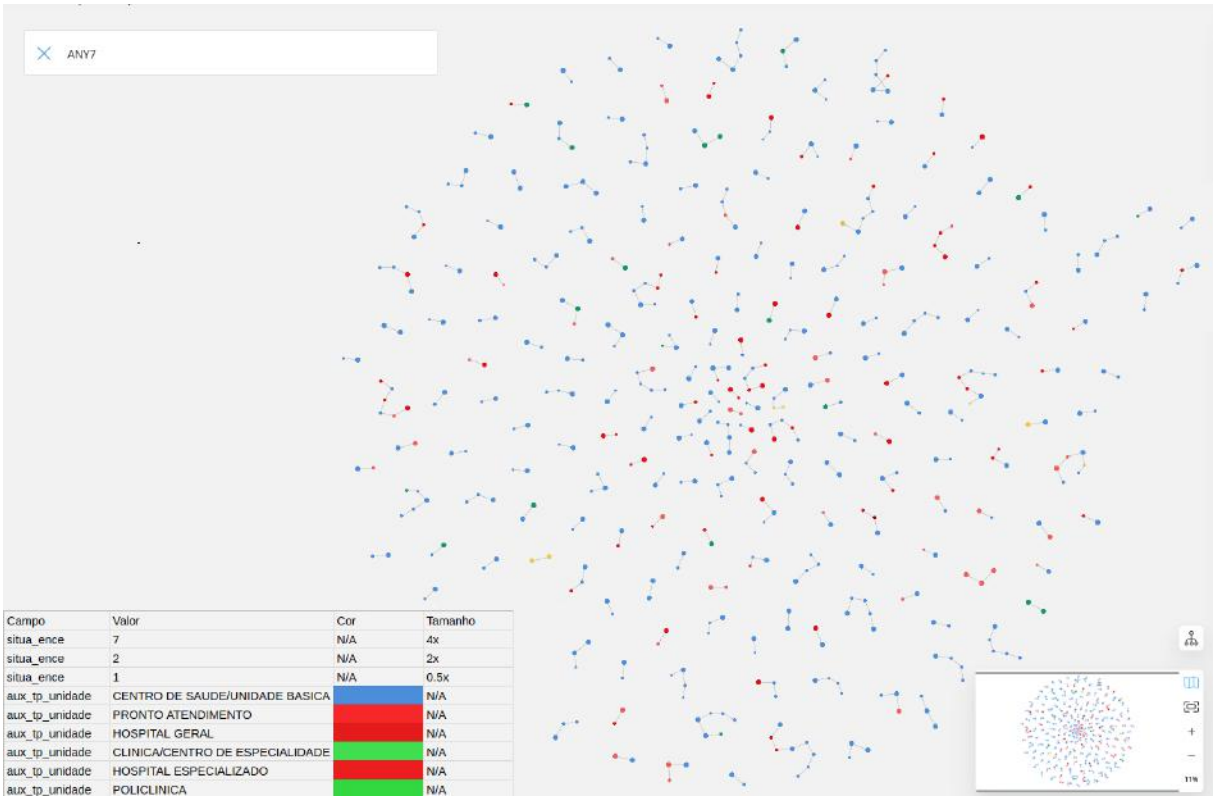
nóstico de tuberculose droga resistente. Essa busca pode ser feita utilizando-se o seguinte código Cypher (Quadro 19):

```
MATCH path = (inicio:Encerramento)-[:NEXT*1..]->(fim:Encerramento)
WHERE Any(node in NODES(path) WHERE node.situa_ence = "7")
AND NOT ()-[:NEXT]->(inicio:Encerramento)
AND NOT (fim:Encerramento)-[:NEXT]->()
RETURN path
```

Quadro 19

A linguagem de consulta utilizada é intuitiva e autoexplicativa, relacionando os nós em um caminho onde o primeiro nó não tem predecessor e o último nó não tem sucessor, sendo o desfecho procurado quando algum nó possui um encerramento como tuberculose droga resistente. A busca do Quadro 19 é ilustrada na figura 4 abaixo:

Figura 4 – Busca de Caminhos da Tuberculose Resistente



A ferramenta Neo4J Bloom permite destacar no grafo propriedades através de cor ou de tamanho dos nós. Na busca acima, as cores foram relacionadas ao tipo de unidade de saúde onde o paciente se tratou, e o tamanho está associado ao tipo de diagnóstico que ele obteve nos atendimentos médicos realizados.

A Figura 4 é uma visão geral das trajetórias de pacientes que tiveram ao menos um diagnóstico de tuberculose droga-resistente ao longo do período. A visualização permite identificar que a maioria dos pacientes recebeu diagnóstico de tuberculose droga-resistente em unidades básicas de saúde (Centros Municipais de Saúde e Clínicas da Família) e há trajetórias de pacientes que contém atendimento em hospital.

Outro exemplo interessante é descobrir em qual tipo de unidade de saúde o paciente obteve logo no primeiro diagnóstico a tuberculose resistente primária. Podemos realizar essa busca com o seguinte código Cypher (Quadro 20):

```
Match path = (inicio:Encerramento)-[:NEXT*1..]
->(fim:Encerramento)
WHERE Any(node in NODES(path)
WHERE inicio.situa_ence = "7")
AND NOT ()-[:NEXT]->(inicio:Encerramento)
AND NOT (fim:Encerramento)-[:NEXT]->()
RETURN path
```

Quadro 20

A busca Cypher do Quadro 20 tem como resultado a figura 5 abaixo:

Figura 5 – Diagnósticos com TBDR, imagem relativa a busca do Quadro 20

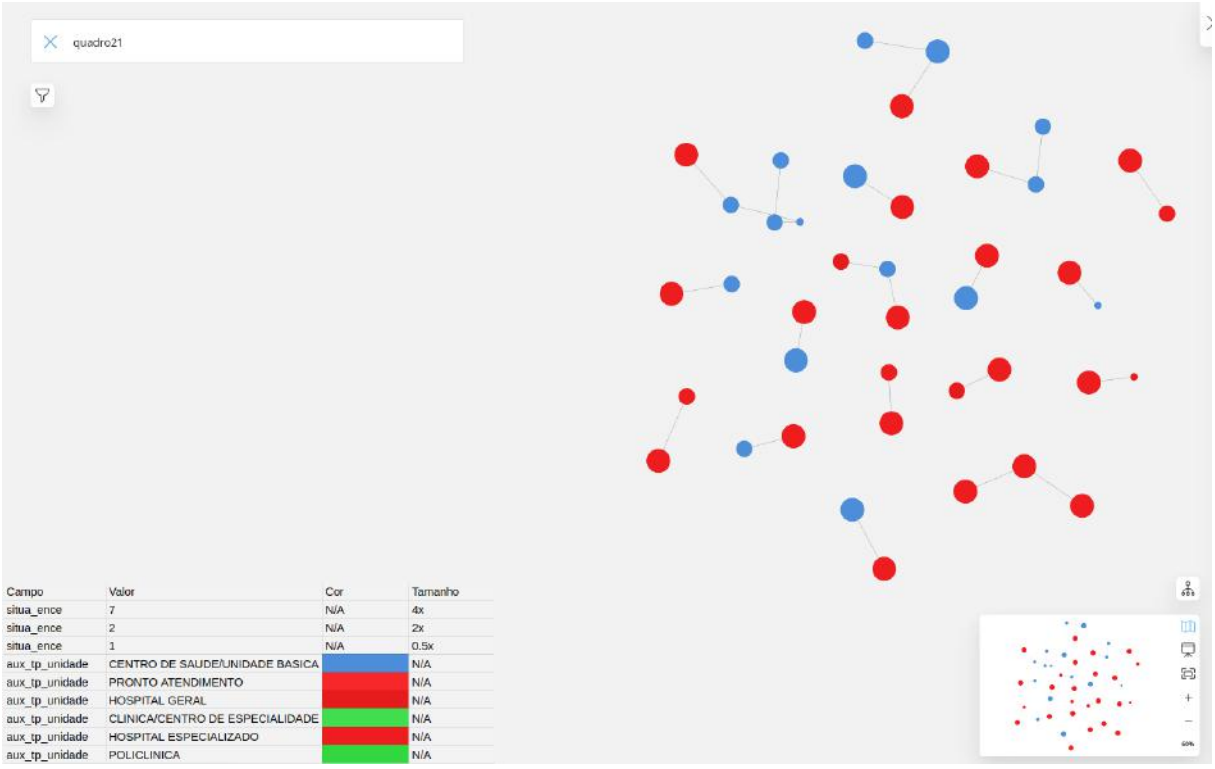
As propriedades de cada nó sempre são destacadas com a característica de cada rótulo, ou seja, pode-se destacar as características que o usuário desejar, de acordo com uma chave de cada nó ou entidade. Para finalizar a ilustração de outra análise solicitada pelos especialistas, que mostra como descobrir quais pacientes tiveram, em até dez consultas, um diagnóstico de droga resistente em um hospital geral ou hospital especializado.

```
MATCH path = (abandono:Encerramento)-[:NEXT*..10]->
(resistencia:Encerramento {situa_ence:"7"})
WHERE NONE(node in NODES(path)[0..LENGTH(path)+1]
WHERE node.situa_ence = resistencia.situa_ence)
and resistencia.aux_tp_estabelecimento = "05"
or resistencia.aux_tp_unidade = "HOSPITAL_ESPECIALIZADO"
AND NOT (resistencia:Encerramento)-[:NEXT]->()
AND NOT ()-[:NEXT]->(abandono:Encerramento)
RETURN path
```

Quadro 21

A busca Cypher do Quadro 21 tem como resultado a figura 6 abaixo:

Figura 6 – Pacientes em até 10 consultas, tiveram diagnósticos em Hospitais, Imagem relativa a busca do Quadro 21



Nesse caso, seria difícil utilizar uma estrutura fixa como os bancos de dados tradicionais. Através de uma estrutura mais flexível fornecida pelo Neo4J, orientado a grafos, ou por bancos de dados não relacionais é possível efetuar tais pesquisas. Além disso, no caso do Neo4J, esse tipo de abordagem garante em geral um desempenho superior quando é sabido de antemão o sub-grafo desejado, pois as buscas são realizadas de forma local. Essa nova forma de análise permite prever padrões e encontrar informações escondidas em grande volume de dados. Com isso, os exemplos apresentados nas consultas realizadas neste trabalho mostra que, mesmo com grande variedade de tipos de dados, a consulta necessita de novas formas diferentes daquelas realizadas nos bancos de dados relacionais tradicionais.

Outra consulta de interesse dos especialistas é identificar os trajetos com pelo menos um atendimento com diagnóstico de droga-resistência, considerando o bairro de residência no primeiro atendimento. Esta consulta foi apresentada em outro formato, que sugere maior facilidade para visualização. Por esta consulta, pode-se identificar os bairros com mais pacientes com tuberculose droga-resistente, os com pacientes com trajetórias de múltiplos atendimentos e, ao contrário, os com pacientes que tenham tido apenas um atendimento, com desfecho cura, por exemplo, anterior ao atendimento que teve a tuberculose droga-resistente como diagnóstico.:

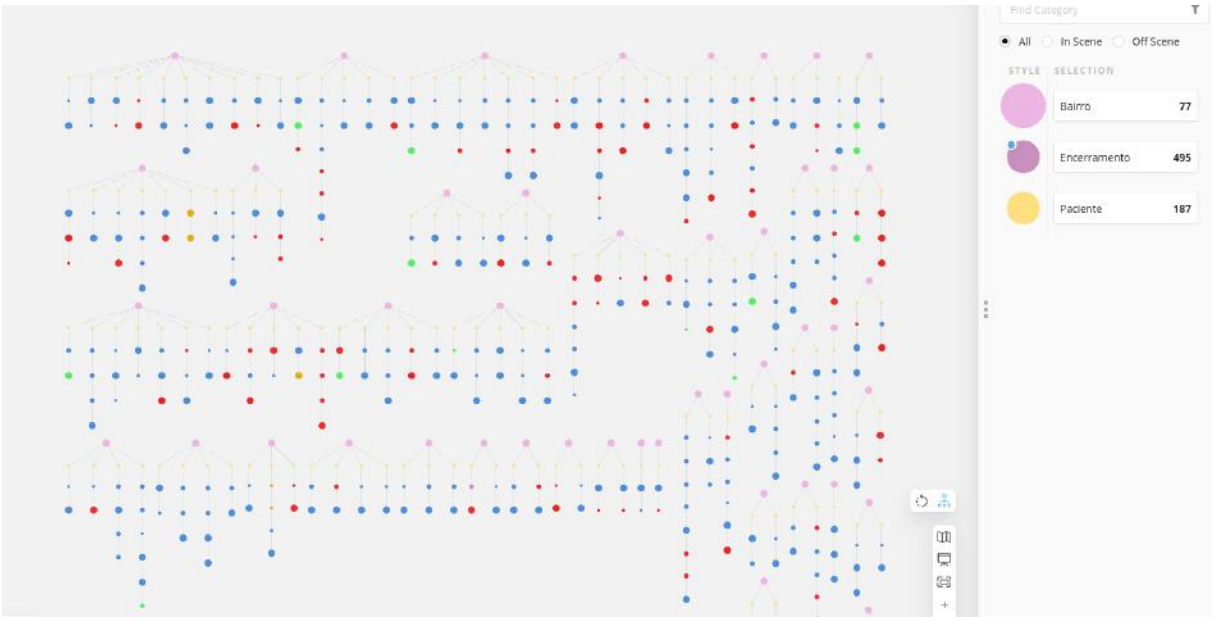
Quadro 22: Trajetos com pelo menos um encerramento droga resistente:

```
Match path = (bairro:Bairro)-[:BairroDoPaciente]
->(paciente:Paciente)-[:Tem]->
(inicio:Encerramento)-[:NEXT*1..]
->(fim:Encerramento)
WHERE Any(node in NODES(path)
WHERE node.situa_ence = "7")
AND NOT ()-[:NEXT]->(inicio:Encerramento)
AND NOT (fim:Encerramento)-[:NEXT]->()
RETURN path
```

Quadro 22

Na figura 7 , é ilustrada a busca do quadro 22, os bairros são nós em rosa, os pacientes são nós menores em amarelo, e a trajetória de consultas são os nós sub-sequentes categorizados pela cor e tamanho, a cor azul tem o significado de atendimento em uma Unidade Básica de Saúde(UBS) e a cor vermelha significa atendimento em hospital, o tamanho categoriza o tipo de diagnóstico sendo nós menores para abandono e nós maiores para tuberculose droga resistente. Nessa busca encontra-se o caminho de todos pacientes que tiveram diagnóstico de TBDR.

Figura 7 – Caminhos com TBDR, imagem relativa a busca do Quadro 22



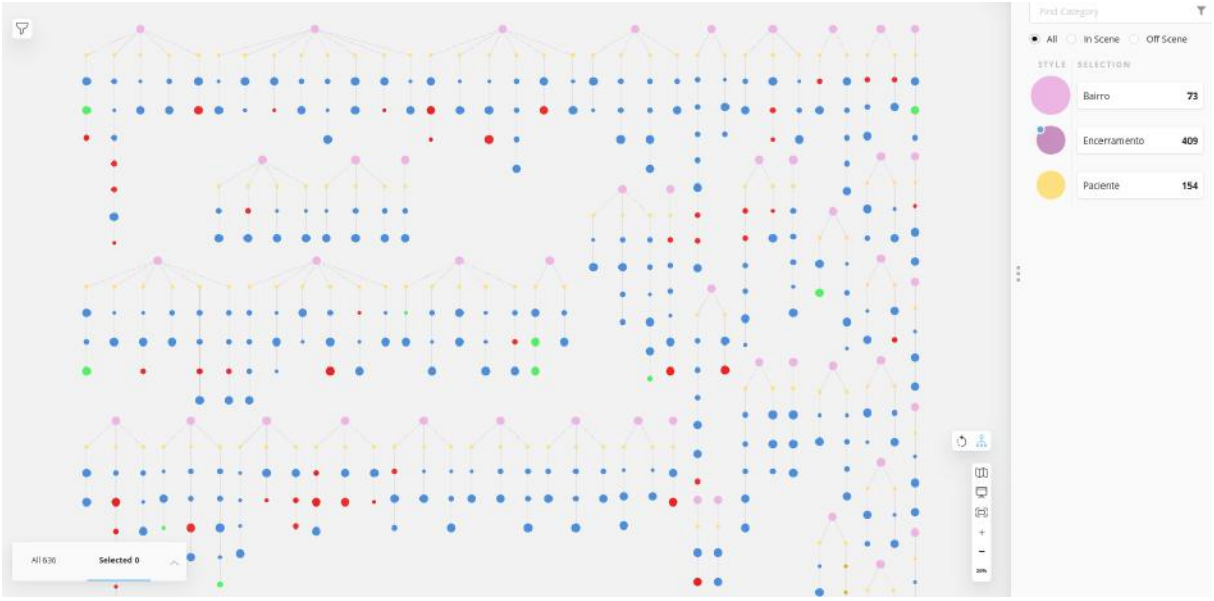
Na figura 8 , é ilustrada a busca do quadro 23, de forma semelhante a busca anterior os nós são categorizados da mesma forma, essa busca ilustra o caminho de pacientes que tiveram pelo menos um diagnóstico de TBDR em Unidades Básicas de Saúde (UBS).

Quadro 23: Trajetos com pelo menos um encerramento droga resistente em Unidade Básica:

```
Match path = (bairro:Bairro)-[:BairroDoPaciente]
->(paciente:Paciente)-[:Tem]->(inicio:Encerramento)
-[:NEXT*1..]->(fim:Encerramento)
WHERE Any(node in NODES(path)
WHERE node.aux_tp_unidade =
"CENTRO_DE_SAUDE/UNIDADE_BASICA"
AND node.situa_ence = "7")
AND NOT (fim:Encerramento)-[:NEXT]->()
AND NOT ()-[:NEXT]->(inicio:Encerramento)
RETURN path
```

Quadro 23

Figura 8 – Caminhos com TBDR em UBS,imagem relativa a busca do Quadro 23



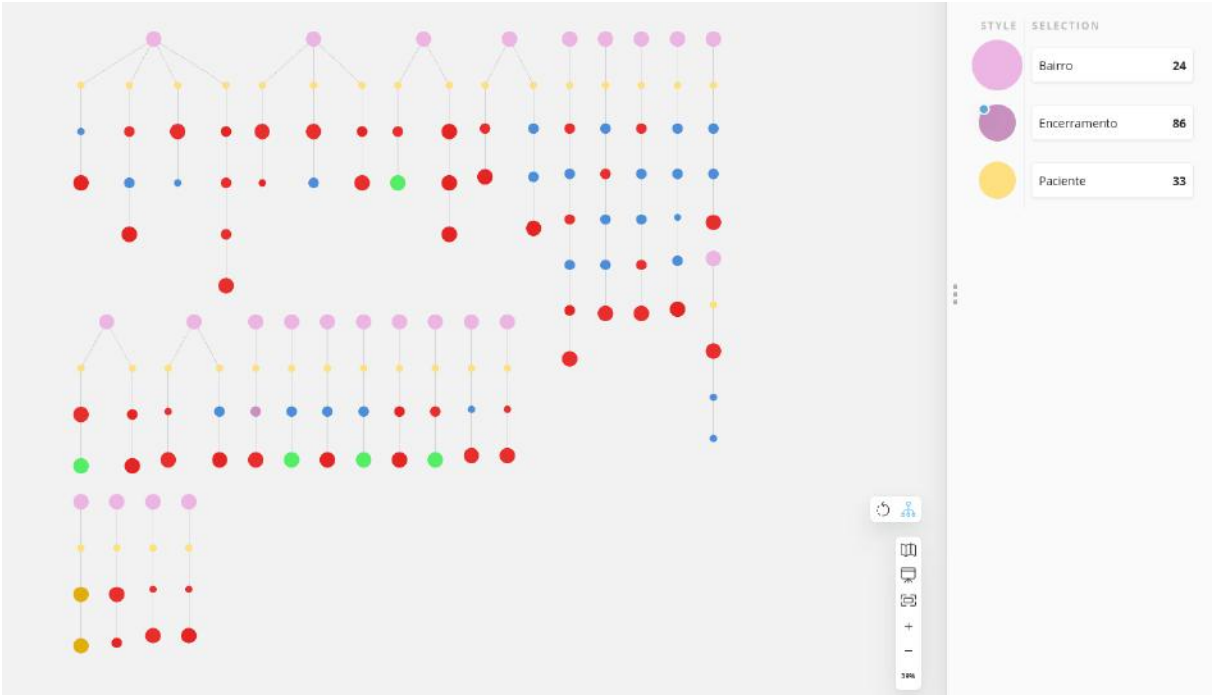
Por último a figura 9, ilustra a busca do quadro 24, com a mesma categorização, essa busca apresenta o caminho de pacientes que tiveram nenhum encerramento droga resistente em Unidade Básica.

Quadro 24: Trajetos com nenhum encerramento droga resistente em Unidade Básica:

```
Match path = (bairro:Bairro)-
[:BairroDoPaciente]->(paciente:Paciente)
-[:Tem]->(inicio:Encerramento)
-[:NEXT*1..]->(fim:Encerramento)
WHERE Any(node in NODES(path)
WHERE node.situa_ence = "7")
and NONE(node in NODES(path)[2..LENGTH(path)+1]
WHERE node.aux_tp_unidade = "CENTRO_DE_SAUDE/UNIDADE_BASICA"
AND node.situa_ence = "7")
AND NOT ()-[:NEXT]->(inicio:Encerramento)
AND NOT (fim:Encerramento)-[:NEXT]->()
RETURN path
```

Quadro 24

Figura 9 – Caminhos com TBDR fora de UBS,imagem relativa a busca do Quadro 24



Outra forma de análise simplificada com um banco de dados em grafos é uma consulta agregada em cima de uma propriedade do grafo, um exemplo dessa capacidade é uma consulta sobre o tempo médio em dias em que cada paciente leva em cada caminho que inclua um diagnóstico de tuberculose Droga Resistente, a consulta agregada do quadro 25 foi executada no terminal do Neo4J, e tem como resultado a seguinte tabela da Figura 10:

```
Match path = (inicio:Encerramento)-[:NEXT*1..]
->(fim:Encerramento)
WHERE Any(node in NODES(path)
WHERE node.situa_ence="7")
AND NOT ()-[:NEXT]->(inicio:Encerramento)
AND NOT (fim:Encerramento)-[:NEXT]->()
RETURN [p in NODES(path)[0..LENGTH(path)+1]
|p.situa_ence] as encerramentos,
toInteger(avg((toInteger(last([v in NODES(path)
[1..LENGTH(path)+1]|
fim.date_timestamp]))-toInteger(head([v in NODES(path)
[1..LENGTH(path)+1]|
inicio.date_timestamp])))/(3600*24))) as tempo,
count(path) as pacientes
ORDER BY count(path) desc AS query
```

Quadro 25

Figura 10 – Tabela agregada com caminhos,imagem relativa a busca do Quadro 25

	A	B	C
1	encerramentos	tempo	pacientes
2	["1","7"]	708	40
3	["2","7"]	493	37
4	["7","7"]	737	19
5	["7","1"]	518	13
6	["7","2"]	583	10
7	["2","2","7"]	725	8
8	["2","2","2","7"]	954	5
9	["1","7","7"]	1294	4
10	["7","4"]	618	3
11	["2","1","7"]	1236	3
12	["7","7","2"]	873	3
13	["7","7","7"]	1364	3
14	["7","2","2"]	948	3
15	["1","7","2"]	1028	2
16	["1","1","1","7"]	1687	2
17	["1","7","1"]	973	2
18	["2","7","7"]	258	2
19	["1","1","7"]	1615	2
20	["2","2","7","7"]	877	2
21	["1","2","7"]	1146	2
22	["2","2","2","2","7"]	1252	2
23	["7","3"]	1009	2
24	["10","7","7"]	741	1
25	["2","2","2","1","7","7"]	1919	1
26	["2","7","2"]	1817	1
27	["2","2","2","7","7","2"]	1206	1
28	["7","1","1"]	1977	1
29	["2","1","2","2","2","7","4"]	1838	1
30	["2","2","2","2","7","2","2","7"]	2290	1
31	["2","2","2","2","7","1"]	890	1
32	["7","7","3"]	1693	1
33	["7","2","2","2","7"]	1021	1
34	["2","2","2","7","7"]	1974	1
35	["1","2","2","7"]	1139	1
36	["7","2","7"]	1661	1
37	["7","2","4"]	565	1
38	["1","2","7","10"]	1938	1
39	["2","2","7","9","2"]	1554	1
40	["2","2","2","1","7"]	1524	1
41	["2","7","2","2"]	488	1
42	["2","2","1","7","7","2"]	2169	1
43	["10","1","7"]	648	1
44	["2","2","1","2","7"]	2508	1
45	["2","2","2","2","2","7"]	1783	1
46	["2","7","2","7","3","1"]	2104	1

Essa forma de análise não se restringe ao tempo médio permitindo também comparar-se os caminhos de acordo com a unidade de saúde que diagnosticou e com caraterísticas do paciente por exemplo: idade e escolaridade. Além das variáveis também permite-se adicionar restrições no problema, acima tem-se que não tem nenhum nó antes do início e nenhum nó depois do final no trajeto, isso significa excluir os sub-caminhos.

6 CONCLUSÃO

Frequentemente, um banco de dados relacional na sua forma original não possui o formato para a análise desejada, seja pelo grande volume de dados, devido ao próprio formato do dado ou devido à habilidade do analista de trabalhar com a ferramenta. Este trabalho pretende demonstrar a dificuldade em estabelecer uma análise de caminhos em um banco de dados relacional comum, devido ao próprio formato do dado.

Bancos de dados orientados a grafos podem permitir analisar dados de novas maneiras, especialmente quando há interesse em entender a sequência em que eventos ocorrem, e com isso tem-se a oportunidade de descobrir novas formas de análise. A transformação da base para um formato adotado no Neo4J é uma evolução que permite o uso mais amplo de análises de caminhos, produzindo informações e gerando novas aplicações.

O ETL construído, assim como a própria base, permite a visualização de dados e geração de informação a partir das ferramentas apresentadas no ambiente do Neo4J, pelo Neo4J Bloom. A visualização permite um grande número de análises sobre os dados, por exemplo: descobrir em que bairro ocorreram mais problemas na atenção primária de saúde, qual bairro merece mais recursos para investir em atenção primária, em quais locais faltam campanhas de conscientização e políticas em saúde coletiva.

Uma aplicação interessante do uso dos dados nesse formato é descobrir padrões na estrutura dos atendimentos, identificando a ordem em que os eventos aconteceram e assim procurar políticas para mitigar o avanço do problema em determinado estágio, antes que avance para um estado crítico. No caso do presente trabalho, descobrir o momento em que o paciente é diagnosticado com a tuberculose droga resistente ou identificar sua localização e tipo de unidade de saúde onde ocorreu o diagnóstico é muito importante. A análise de dados com esta estrutura em tabelas é dificultada pelo grande número de dados que são exibidos, pela forma de visualização, pela complexidade da busca e pela dificuldade de expressar em linguagem de consulta a busca desejada.

Destaca-se no trabalho a facilidade da linguagem Cypher, mais próxima da linguagem de consulta usada em bancos de dados relacionais. Apesar das linguagens de consulta ou ferramentas próprias permitirem análises mais complexas, o próprio analista pode ter dificuldade em identificar uma forma de identificar caminhos ou sequência de eventos em uma linguagem de consulta tradicional.

Além dos dados do SINAN esse tipo de análise é comumente utilizada para descobrir padrões de consumidores em E-Commerce, compra de viagens ou pela simples navegação Web. No caso da saúde, o exemplo da análise sobre tuberculose droga-resistente com o uso do SINAN apontou potencial de uso promissor para apoiar ao monitoramento, priorização e elaboração de ações. Pode-se adaptar os passos descritos no trabalho para qualquer tipo de análise de caminho em que o usuário tenha posse do horário em que o evento aconteceu

e o resultado.

A análise sobre caminhos ainda é um tema recente e com poucas referências, porém sua utilidade nas mais diversas áreas de conhecimento é algo que merece atenção e o trabalho pretende evoluir o conhecimento tanto sobre o tipo de análise quanto pela informação sobre a TB-DR no Brasil.

Finalmente, este trabalho permite a investigação pelos especialistas em saúde coletiva para detectar falhas no tratamento de doenças a partir de diagnósticos obtidos em unidades de saúde, disponibilizando consultas específicas por bairros, perfis de pacientes que sofrem com a TB-DR, e outras características existentes na base de dados, permitindo a descoberta de conhecimentos em novas pesquisas.

7 TRABALHOS FUTUROS

O trabalho pretende apresentar uma visualização de dados que permita a geração de informação para o utilizador porém, o trabalho depende do nível de conhecimento do especialista sobre o tema para geração de conclusões.

Sugestões de trabalhos futuros seria relacionar o risco de desenvolver a TB-DR dadas certas variáveis como em (DEBANNE ROGER A. BIELEFELD; ROWLAND, 2000). O risco de desenvolver TB-DR dadas certas variáveis, calculado de forma automatizada, permite a criação de aplicações automatizadas para alarmes ou softwares de geração de informações. Tal ferramenta poderia prover relatórios semanais ou mensais à equipe de vigilância epidemiológica.

Outra dificuldade encontrada no trabalho foi a necessidade de realizar o ETL em um servidor local, dadas as restrições de utilização pela empresa proprietária do Neo4J. Também não é de conhecimento acessível a informação sobre sistemas em nuvem que aceitem a construção de um banco de dados em grafos como utilizado no trabalho. Tal dificuldade restringe o acesso a informação desenvolvido no ETL descrito.

Além de muitos provedores de sistemas em nuvens não oferecerem suporte a ferramenta, drivers de conexões do Neo4J Bloom e do próprio Neo4J a tais sistemas são limitados e não foram investigados neste trabalho, porém, o acesso a informação como disponibilizado no servidor local pelo Neo4J Desktop no navegador de qualquer usuário da internet seria algo complementar a este trabalho e isso envolveria investigar e trabalhar com a versão restrita do Neo4J, ou seja o Neo4J Server.

REFERÊNCIAS

CRYSTAL, D. **The Cambridge Encyclopedia of Language**. 2nd edition. ed. Cambridge: Cambridge University Press, 1997.

CYPHER. **Introduction to Cypher**. 2021. <https://neo4j.com/developer/cypher/>. Acesso em: 4 Fev.2021.

DEBANNE ROGER A. BIELEFELD, G. M. C. T. M. D. S. M.; ROWLAND, D. Y. Multivariate markovian modeling of tuberculosis: Forecast for the united states. **Emerging Infectious Diseases**, v. 6, n. 2, p. March–April, 2000.

HEUSER, C. A. **Projeto de banco de dados**. 6. ed. Porto Alegre: Bookman, 2009.

MARIADB. **Understanding the Relational Database Model**. 2021. <https://mariadb.com/kb/en/understanding-the-relational-database-model/>. Acesso em: 2 Jan.2021.

NEO4J. **Neo4j Online Meetup 28: User path analysis in Neo4j**. 2021. <https://www.youtube.com/watch?v=JValprxgn0w>. Acesso em: 12 Jan.2021.

NEO4J. **What is a Graph Database?** 2021. <https://neo4j.com/developer/graph-database/>. Acesso em: 2 Jan.2021.

POSTGRESQL. **What is PostgreSQL?** 2021. <https://www.postgresql.org/about/>. Acesso em: 2 Jan.2021.

REZENDE, M. de O. **Sistema Único de Saúde (SUS)**. 2021. <https://mundoeducacao.uol.com.br/sociologia/sistema-unico-de-saude-sus.htm>. Acesso em: 30 Jan.2021.

ROBINSON, I.; WEBBER, J.; EIFREM, E. **Graph Databases**. 2. ed. Sebastopol: O'Reilly Media, Inc., 2015.

ROCHA, S. M.; BARTHOLOMAY, P. et al. Sistema de informação de agravos de notificação (sinan): principais características da notificação e da análise de dados relacionada à tuberculose. **Epidemiol. Serv. Saude**, v. 29, n. 1, p. 1–13, 2019.

SAMPAIO, R. S. **Ambiente de Dados do SIHSUS com MongoDB**. Rio de Janeiro: Monografia (Trabalho de Conclusão de Curso) — Departamento de Ciência da Computação-Instituto de Matemática, Universidade Federal do Rio de Janeiro, 2019.

SANTOS, I. A. d.; CARVALHO, R. O. d. **Ambiente de exploração de dados de saúde usando um Banco de Dados NoSQL**. Rio de Janeiro: Monografia (Trabalho de Conclusão de Curso) — Departamento de Ciência da Computação-Instituto de Matemática, Universidade Federal do Rio de Janeiro, 2017.

SAÚDE, C. N. D. **20 anos do SUS** . 2021. <https://conselho.saude.gov.br/web_sus20anos/sus.html>. Acesso em: 30 Jan.2021.

SAÚDE, M. D. **Princípios do SUS**. 2021. <https://bvsms.saude.gov.br/bvs/publicacoes/sus_principios.pdf>. Acesso em: 30 Jan.2021.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Database System Concepts**. 7. ed. New York: McGraw-Hill, 2020.

SNOWPLOW. **Using graph databases to perform pathing analysis – initial experiments with Neo4J**. 2021. <https://snowplowanalytics.com/blog/2014/07/31/using-graph-databases-to-perform-pathing-analysis-initial-experimentation-with-neo4j/>. Acesso em: 12 May.2021.

APÊNDICES

APÊNDICE A – COMPLETUDE DA BASE TUBENET

Tabela 1 – Completude dos dados para a Base Tubenet de 2012 a 2018

CAMPO	COMPLETUDE	CAMPO	COMPLETUDE
link_id	1,00	agravalcoo	1,00
dt_inic_tr	1,00	agravdiabe	1,00
dt_encerra	0,99	agravdoenc	1,00
situa_ence	1,00	agravdroga	0,65
nu_idade_n	1,00	agravtabac	0,64
cs_sexo	1,00	hiv	1,00
cs_raca	0,98	nm_bairro	0,96
cs_escol_n	0,99	id_unidade	1,00
agravaids	1,00		

APÊNDICE B – DOMÍNIO DA BASE TUBENET

Tabela 2 – Domínio dos dados para a Base Tubenet de 2012 a 2018

CAMPO	Mínimo	Máximo
link_id	10	9998
dt_inic_tr (ano-mês)	2012-01	2019-04
dt_encerra (ano-mês)	2012-01	2019-06
situa_ence	1	9
nu_idade_n	1026	4101
cs_sexo	F	M
cs_raca	1	9
cs_escol_n	0	9
agravaids	1	9
agravalcoo	1	9
agravdiabe	1	9
agravdoenc	1	9
agravdroga	1	9
agravtabac	1	9
hiv	1	4
nm_bairro	25 DE AGOSTO	ZUMBI
id_unidade	0	9442251

APÊNDICE C – CLASSE PYTHON LOAD.PY

Código C.1 – load.py

```
import psycopg2
import sys
import csv

def pg_load_table(ctx, csv_file, table_name, **kwargs):

    file_path = ctx.inbox_dir + "/" + csv_file
    delimiter = kwargs.get('delimiter', '|')
    try:
        conn = psycopg2.connect(dbname=ctx.db_name,\
            host=ctx.hostname\
            port=ctx.port,\
            user=ctx.username, password=ctx.password)
        print("Connecting to Database")
        cur = conn.cursor()
        f = open(file_path, 'r',\
            encoding="utf8", errors='ignore')
        cur.execute("Truncate {} Cascade;".format(table_name))
        print("Truncated {}".format(table_name))
        cur.copy_expert("copy {} from STDIN delimiter '{}'\
           CSV HEADER".format(table_name, delimiter), f)
        cur.execute("commit;")
        cur.execute("select count(*) from {}".format(table_name))
        rows = cur.fetchone()
        print('{} rows in table {}'.format(rows[0], table_name))
        print("Loaded data into {}".format(table_name))
        conn.close()
        print("DB connection closed.")

    except Exception as e:
        print("Error: {}".format(str(e)))
        sys.exit(1)
```

APÊNDICE D – CLASSE PYTHON STEPS.PY

Código D.1 – steps.py

```
import psycopg2
import sys
import psycopg2
import csv

def step(ctx, table_name):

    file_path = ctx.sql_dir + "/" + "dml" + "/"
    file_name = table_name+".sql"
    print("Executing {}".format(file_name))
    try:
        conn = psycopg2.connect(dbname=ctx.db_name,\
            host=ctx.hostname,\
            port=ctx.port,\
            user=ctx.username,\
            password=ctx.password)
        print("Connecting to Database")
        cur = conn.cursor()
        text = open(file_path + file_name, "r")\
            .read().split(";")
        length = len(text)
        for i in range(length-1):
            print(text[i] + ';')
            cur.execute(text[i] + ';')
            try:
                row = cur.fetchone()
                while row is not None:
                    print('{} rows'.format(row))
                    row = cur.fetchone()
            except Exception as e:
                print()
        cur.execute("commit;")
        conn.close()
        print("Finish {}".format(file_name))

    except Exception as e:
        print("Error: {}".format(str(e)))
        sys.exit(1)
```

APÊNDICE E – CLASSE PYTHON STEPSINNEO2.PY

Código E.1 – stepsInNeo2.py

```
import neo4j
import sys


def loadStep(ctx, table_name):
    file_path = ctx.cypher_dir + "/"
    file_name = table_name + ".cy"

    try:
        driver = neo4j.GraphDatabase.\
            driver(uri=ctx.uri,\
            auth=(ctx.authlogin, ctx.authpass),
        encrypted=False)
        session = driver.session()

        print("Connecting to Neo4J")
        record = session\
            .run(open(file_path + file_name, "r")\
            .read())
        print("Executed {}".format(file_name))
        driver.close()

    except Exception as e:
        print("Error: {}".format(str(e)))
        sys.exit(1)

    return record
```