UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

INSTITUTO DA COMPUTAÇÃO

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Jorge Luís Ribeiro de Oliveira

COMPARAÇÃO DE DESEMPENHO ENTRE BANCOS DE DADOS RELACIONAIS E DE GRAFO RDF NO CONTEXTO DO ARMAZENAMENTO DE DADOS DO TRANSPORTE PÚBLICO

RIO DE JANEIRO

2024

Jorge Luís Ribeiro de Oliveira

COMPARAÇÃO DE DESEMPENHO ENTRE BANCOS DE DADOS RELACIONAIS E DE GRAFO RDF NO CONTEXTO DO ARMAZENAMENTO DE DADOS DO TRANSPORTE PÚBLICO

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

Orientadora: Profa. Giseli Rabello Lopes

RIO DE JANEIRO

2024

CIP - Catalogação na Publicação

O48c

Oliveira, Jorge Luís Ribeiro de
Comparação de desempenho entre bancos de dados relacionais e de grafo RDF no contexto do armazenamento de dados do transporte público / Jorge Luís Ribeiro de Oliveira. -- Rio de Janeiro, 2024.
106 f.

Orientadora: Giseli Rabello Lopes.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Computação, Bacharel em Ciência da Computação, 2024.

1. Gerenciamento de dados. 2. Web semântica. 3. GTFS. 4. Transporte público. I. Lopes, Giseli Rabello, orient. II. 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.

Jorge Luís Ribeiro de Oliveira

COMPARAÇÃO DE DESEMPENHO ENTRE BANCOS DE DADOS RELACIONAIS E DE GRAFO RDF NO CONTEXTO DO ARMAZENAMENTO DE DADOS DO TRANSPORTE PÚBLICO

Trabalho de conclusão de curso de graduação apresentado ao Instituto de 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 agosto de 2024.

BANCA EXAMINADORA:

gov.br Documento assinado digitalmente
GISELI RABELLO LOPES
Data: 19/08/2024 13:35:41-0300
Verifique em https://validar.iti.gov.br

Giseli Rabello Lopes, D.Sc. (UFRJ)

gov.br Documento assinado digitalmente
SILVANA ROSSETTO
Data: 19/08/2024 18:14:14-0300
Verifique em https://validar.iti.gov.br

Silvana Rossetto, D.Sc. (UFRJ)

gov.br Documento assinado digitalmente
VIVIAN DOS SANTOS SILVA
Data: 20/08/2024 10:12:05-0300
Verifique em https://validar.iti.gov.br

Vivian dos Santos Silva, Ph.D. (UFRJ)

A todos que, como eu, decidiram se dedicar à melhora do transporte público. A ferramenta de democratização das cidades.

AGRADECIMENTOS

Em primeiro lugar, agradeço aos meus pais, Luiz e Juciene. Eles são os responsáveis por eu ter chegado até aqui.

À minha namorada Yohanna, por me confortar nos momentos que acreditei que não conseguiria continuar.

Aos meus amigos, Letícia, Nicolas e Rebeca, por ouvirem todas as minhas frustrações.

À professora Giseli, por ter aceitado ser minha orientadora e me guiado até conclusão deste trabalho.

Ao professor Collier, por ter me introduzido a alegria do processo de pesquisa acadêmica.

À professora Silvana, nossas reuniões foram essenciais para que eu tivesse a ideia do tema deste trabalho.

À Bridgit Mendler, pelas músicas que me mantiveram focado e me impediram de desistir.

Por fim, a todos que caminharam comigo na jornada acadêmica.

RESUMO

Dados sobre a infraestrutura de transporte público de uma cidade geralmente são disponibilizados em formato tabular pelos órgãos responsáveis em repositórios de dados abertos. Essa forma de disponibilização facilita seu uso na carga de bancos relacionais. No entanto, nos últimos anos, foram mostrados os benefícios do armazenamento desse tipo de dado em bancos de grafos. Este trabalho tem como objetivo comparar a implementação dos dois tipos bancos de dados quando populados com os dados do GTFS (*General Transit Feed Specification*) da cidade do Rio de Janeiro. A comparação é feita avaliando três bancos, dois relacionais e um terceiro em grafos RDF construído a partir da ontologia *Linked* GTFS Ontology, sob a ótica do tempo de execução de um grupo de consultas, seguindo a metodologia apresentada em (Chaves-Fraga et al., 2020). Os resultados obtidos indicam um melhor desempenho para os bancos relacionais do que para o banco de dados em grafos RDF na execução das consultas.

Palavras-chave: Gestão de Dados; Web Semântica; Transporte Público; GTFS.

ABSTRACT

The data on a city public transportation infrastructure is usually made available in a tabular format by the responsible agencies in data repositories. By distributing the data in this format it facilitates the use of relational databases for this type of data. However, in recent years, the benefits of storing this type of data in graph databases have been demonstrated. This work aims to compare the implementation of the two types of databases when populated with GTFS (General Transit Feed Specification) data from the city of Rio de Janeiro. The comparison is made by evaluating three databases, two of the relational kind and a third RDF graph database built from the Linked GTFS Ontology, from the perspective of query execution time, following the methodology presented in (Chaves-Fraga et al., 2020). The results obtained indicate better performance for the relational databases than for the RDF graph database in executing the queries.

Palavras-chave: Data Management; Semantic Web; Public Transportation; GTFS.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS

CSV	*Comma-separated Values*
EMTU	Empresa Metropolitana de Transportes Urbanos de São Paulo
ER	Entidade Relacionamento
ETL	*Extract Transform Load*
GREL	*General Refine Expression Language*
GPS	*Global Positioning System*
GTFS	*General Transit Feed Specification*
IRI	*Internationalized Resource Identifier*
OBDI	*Ontology Based Data Integration*
RDF	*Resource Description Framework*
SPARQL	*SPARQL Protocol and RDF Query Language*
SQL	*Structured Query Language*
SGBD	Sistema Gerenciador de Banco de Dados
W3C	*World Wide Web Consortium*
UTF-8	*8-bit Unicode Transformation Format*

SUMÁRIO

1 INTRODUÇÃO

O surgimento das chamadas Cidades Inteligentes (CUNHA et al., 2016) e o movimento de Internet das Coisas (ROSE; ELDRIDGE; CHAPIN, 2015) impulsionaram a criação de fluxos de grande volume de dados. Esses dados provêm de diferentes fontes e tratam sobre diversas esferas do ambiente urbano (por exemplo, segurança, administração pública, transporte, etc.). No âmbito do transporte público, a eminência do uso de dispositivos de *Global Positioning System* (GPS) no monitoramento de ônibus concessionados tornou comum o acesso a informações, em certos momentos em tempo real, sobre a infraestrutura e os veículos presentes em uma cidade. Em geral, os dados são disponibilizados pelas prefeituras das cidades em repositórios, seguindo (ou não) um padrão de estruturação no formato tabular. Um dos padrões mais comum é o *General Feed Transit Specification*[1] (GTFS).

O GTFS, na forma aplicada a esses repositórios, contempla os dados em diferentes arquivos tabulares, que podem conter referências entre si, de forma similar à organização de um banco de dados relacional. Somado a isso, o formato de arquivo mais encontrado é o de *Comma-Separated Values* (CSV), que permite importação direta como uma única tabela, na maioria dos Sistemas de Gerenciamento de Banco de Dados (SGBD) relacionais (HEUSER, 1998). Esses fatores sugerem o modelo de banco de dados relacional como o mais simples e prático para a implementação de aplicações e análises nesse contexto. No entanto, nos últimos anos, trabalhos como (GUBERT et al., 2020), (DINIZ; DA CUNHA; LOUREIRO, 2020) e (FORTIN; MORENCY; TREPANIÉR, 2016) mostram as vantagens obtidas ao armazenar esses dados fazendo uso de bancos de dados de grafos, uma abordagem diferente do tradicional modelo relacional em que o armazenamento dos dados não é feito em tabelas, mas, sim, nos vértices e arestas de um grafo (ANGLES, 2012). Esse fluxo recente de artigos demonstra que esse modelo também é uma alternativa viável ao se tratar dos dados de sistemas de trânsito.

Uma das formas de se implementar um banco de dado de grafos é por meio da utilização do formato *Resource Description Framework*[2] (RDF), que, auxiliado por ontologias (STAAB; STUDER, 2009), permite a criação de conexões semânticas entre os dados armazenados, ou seja, os dados passam a ser entendidos em um contexto e não como

[1] https://gtfs.org.

[2] https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

entidades isoladas. O uso desse método permite integração do grafo na Web Semântica, o que facilita a conexão com outros conjuntos de dados e a realização de inferências (BERNERS-LEE; HENDLER; LASSILA, 2001).

Nesse contexto, objetivo deste trabalho é entender a diferença entre o impacto de implementações relacionais e de grafos RDF no desempenho de um banco de dados contendo dados relativos à infraestrutura e a veículos do sistema de ônibus concessionados de uma cidade. O desempenho foi avaliado considerando apenas o tempo de execução de consultas individualmente, sem o contexto das necessidades da aplicação em que estariam inclusas. Para a comparação, foram realizadas três implementações distintas: (i) relacional seguindo diretamente o padrão GTFS, (ii) relacional objetivando alcançar uma estrutura mais normalizada (HEUSER, 1998) e (iii) em grafos RDF fazendo uso de uma ontologia desenvolvida para o padrão GTFS[3]. Em cada banco, executou-se um conjunto de consultas adaptadas para suas respectivas estruturas, porém com retornos equivalentes. As consultas almejam testar o desempenho de certos elementos comuns à linguagem de consultas (junções, agrupamentos, ordenações, etc.) e/ou simular recuperações de dados relevantes para análises do tema (baldeações necessárias para viajar entre duas paradas, paradas acessíveis a partir do local atual, etc.). Os resultados encontrados apontam para a superioridade dos modelo de banco relacional em relação ao modelo em grafo RDF sob essas circunstâncias.

O restante deste trabalho está organizado da seguinte forma. No capítulo 2, são apresentadas as fundamentações teóricas que embasam este trabalho, são incluídas definições e explicações dos conceitos utilizados, contextualizando-os ao escopo e objetivo deste trabalho. No capítulo 3, são descritos os processos realizados para estabelecer o ambiente em que foram realizados os experimentos, passando por todas as etapas do processo de ETL (*Extract Transform Load*) e apontando o raciocínio que levou às escolhas de implementação tomadas. No capítulo 4, é apresentada uma comparação entre as implementações relacionais e em grafo para um mesmo conjunto de dados, avaliando consultas, em ambos os bancos de dados, por meio de métricas estabelecidas. Por fim, no capítulo 5, conclui-se o trabalho apontando as principais contribuições, as dificuldades encontradas e as propostas para continuação deste trabalho.

[3] https://github.com/OpenTransport/linked-gtfs/blob/master/spec.md.

2 REFERENCIAL TEÓRICO

Para o melhor entendimento dos experimentos descritos nos próximos capítulos, é preciso ter familiaridade com alguns conceitos. Neste capítulo, conceitos como os de Web Semântica e Bancos de Dados são apresentados e discutidos, levando em consideração como eles podem ser aplicados no contexto deste trabalho. O capítulo também introduz o funcionamento das ferramentas que são aplicadas ao longo do trabalho.

2.1 WEB SEMÂNTICA

Berners-Lee, Hendler e Lassila (2001) apresentam a Web Semântica como uma extensão da Web clássica, marcada por *hiperlinks*, em que a informação é estruturada de forma a ser entendida por máquina, isto é, torna-se possível para um *software* realizar conexões semânticas de maneira similar a uma pessoa. No entanto, para se alcançar esse nível de interconexão é necessária uma série de padrões para a representação de dados. Esta seção apresenta esses padrões, primeiramente, focando na representação estrutural e, posteriormente, na semântica.

2.1.1 *Resource Description Framework* (RDF)

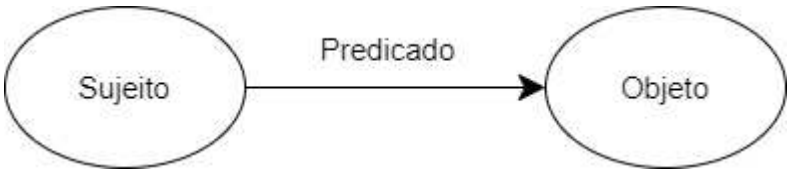
Na Web Semântica, para que a informação seja processada por um computador, é necessário encontrar formas de organizá-la que sejam voltadas para esse objetivo (BERNERS-LEE; HENDLER; LASSILA, 2001). Assim, modelos de estruturação foram pensados para suprir essa necessidade, entre eles, o *Resource Description Framework* (RDF) despontou como um dos mais populares.

O RDF, desenvolvido pela organização *World Wide Web Consortium*[4] (W3C), propõe uma abordagem para a estruturação dos dados. De forma mais específica, a informação é organizada em triplas contendo sujeito, predicado e objeto (KLYNE; CARROLL; MCBRIDE, 2014). Essa organização tem como objetivo apresentar uma estrutura facilmente entendida tanto por máquinas quanto por seres humanos. Nessa estruturação, sujeito e objeto representam elementos enquanto o predicado é a representação de um relacionamento entre eles. A tripla RDF é uma declaração modelada por um grafo em que o sujeito e o objeto são

[4] https://www.w3.org/.

vértices e o predicado é a aresta conectando-os, conforme mostrado na Figura 1. O grafo gerado por um conjunto de uma ou mais triplas RDF é chamado grafo de conhecimento.

Figura 1 – Estrutura de uma tripla RDF



Para o formato RDF, um vértice pertence a um dos seguintes 3 tipos: IRI (*Internationalized Resource Identifier*), vértice em branco ou literal. Nesse contexto, IRIs são os identificadores de elementos na Web e podem ocupar qualquer posição na tripla. É por meio deles que a conexão entre grafos de conhecimento é possível, pois se dois vértices em grafos diferentes são representados por um mesmo IRI, eles são, em realidade, um único vértice em um grafo maior formado pela união dos diferentes grafos de conhecimento que contêm aquela IRI. Já os vértices em branco são identificadores intra-grafos perdendo seu sentido fora do contexto onde é definido, na tripla, ocupam as posições de sujeito ou objeto. Por fim, literais são valores que não são referências a IRIs, geralmente números, datas e sequências de caracteres. Para eles, a única posição possível na tripla é a de objeto.

Além de facilitar o entendimento dos dados por máquina, o formato RDF também possibilita a obtenção de novas informações por meio de inferências. Isso é possível através da definição de conjuntos de regras associadas às ontologias. Assim, um computador é capaz de inferir conclusões que não foram explicitadas no conjunto de dados original.

2.1.2 Ontologia

Enquanto o RDF estipula uma estrutura em comum para que a informação seja facilmente entendida por máquina, a responsabilidade de atribuir significado aos elementos recai sobre as ontologias. Embora o termo seja aplicado em diversas áreas, para Berners-Lee, Hendler e Lassila (2001), uma ontologia, no contexto da Web Semântica, apresenta-se como as definições dos tipos dos elementos, relacionamentos e as regras lógicas aplicáveis entre termos de uma área e são comumente representadas através de documentos. São formas de modelar estruturas de um sistema, suas entidades e seus relacionamentos (STAAB; STUDER, 2009). Dessa forma, as ontologias agem na definição de domínios, garantindo que dados gerados por fontes diferentes possam ser entendidos em função de seu significado em vez de em função de aspectos superficiais da sua forma de apresentação.

Contudo, uma ontologia não se limita a apenas uma lista de definições; as ontologias também são responsáveis por indicar como os termos descritos se relacionam dentro de uma tripla RDF e possíveis ligações com outras ontologias. Portanto, uma forma mais completa de entender os propósitos e responsabilidades de uma ontologia é por meio das características da ontologia como especificação de domínio apontadas por Grimm *et al*. (2011).

Ainda segundo Grimm *et al*. (2011), cinco características aparecem ao se observar ontologias por esse ângulo: formalidade, explicitude, consenso, conceptibilidade e especificidade ao domínio. As três primeiras estão relacionadas ao seu objetivo de tornar compreensível nuances e ideias intuitivas a uma pessoa para uma máquina. Uma ontologia deve apresentar de forma lógica (formalidade) todos os aspectos relevantes de um domínio sem depender de inferências (explicitude) e, como o objetivo do campo da Web Semântica é a conexão entre diferentes fontes de dados, uma ontologia adotada por uma minoria na comunidade do domínio que representa é de pouco valor. As duas últimas características falam sobre outro papel importante das ontologias: descrever um domínio. O foco delas não está em refletir instâncias específicas, mas sim as essências mais gerais das entidades componentes (conceptibilidade). Dessa forma, há, também, mérito em se manter restrito o domínio alvo (especificidade ao domínio), dado que, ao se observar uma área muito abrangente, torna-se mais complexo resumi-la aos seus principais conceitos.

2.2 BANCOS DE DADOS

Segundo Heuser (1998), um banco de dados é a integração de um conjunto de dados contemplando os objetivos de um grupo de usuários. Essa organização pode ser feita de diversas maneiras, relacionadas à estrutura do dado que armazena. Os diferentes modos de integração desses conjuntos de dados são os fatores que distinguem os tipos de bancos de dados. Neste trabalho, serão abordados dois tipos de bancos de dados: relacional e de grafos.

2.2.1 Banco de Dados Relacional

Um banco de dados relacional é caracterizado por dispor as informações armazenadas por meio de relações. Essas relações, comumente conhecidas como tabelas, são compostas por um esquema, e por instâncias. O primeiro é o molde com os atributos da relação, enquanto o último é o agrupamento de registros armazenados de acordo com o esquema definido. Segundo Heuser (1998), esquema de um banco de dados relacional é composto por um

conjunto de relações, seus atributos e as restrições de integridade que regem seus atributos. Uma regra de integridade delimita os possíveis valores de uma propriedade da relação, de forma a manter o banco internamente consistente. Tais regras abrangem: limitações ao tipo do dado de uma propriedade, se a propriedade deve apresentar valores únicos, se os valores estão restritos a algum conjunto, entre outras delimitações.

Nesse tipo de banco, o dado é acessado e manipulado por meio de consultas escritas em *Structured Query Language*[5] (SQL). Uma consulta permite a recuperação de informação armazenada em um banco de dados (instâncias) e requer conhecimento prévio da organização do mesmo banco (esquema). No entanto, o uso do SQL não está limitado a isso, também é possível alterar os dados contidos em uma instância, elaborar cálculos sobre esses dados, criar tabelas no banco, etc.

2.2.2 Banco de Dados de Grafos

Sendo outro paradigma para o armazenamento de dados, um banco de dados de grafos é definido por Monteiro, Sá e Bernardinho (2023) como um banco que, seguindo um modelo de dados em grafos, representa a informação por meio de vértices e arestas. A estrutura dos vértices e arestas determina o tipo do banco de dados de grafos. Se os vértices e as arestas contêm pares nome-valor, o banco é do tipo de grafos de propriedades. Caso os vértices e as arestas forem construídos seguindo o formato RDF, o banco é do tipo de grafos RDF (*triplestore*) (ANGLES; THAKKAR; TOMASZUK, 2020). Nos últimos anos, esse tipo de representação destacou-se nos campos das redes sociais, aprendizado de máquina e, mais relevante para o contexto deste trabalho, Web Semântica.

A popularização desse paradigma nessas áreas está relacionada com a compatibilidade dos problemas encontrados nelas (grande volume de informação, estruturas inconstantes, etc.) com as características desses bancos. Dentre essas características, para Monteiro, Sá e Bernardinho (2023), as seguintes podem ser consideradas como as três principais: desempenho, flexibilidade e agilidade. Desempenho refere-se à capacidade desses bancos em manter a velocidade de consulta relativamente constante a medida que o número de conexões e a profundidade dos relacionamentos aumenta, diferentemente dos bancos relacionais. Já a característica da flexibilidade trata da possibilidade do esquema de um banco de grafos ser alterado à medida que novos dados são inseridos, sem afetar seu funcionamento. Por fim, a agilidade é sobre a capacidade desses bancos se adequarem às práticas de desenvolvimento

[5] https://pt.wikipedia.org/wiki/SQL.

ágil, adaptando-se às necessidades mais recentes dos usuários. As características como desempenho e flexibilidade facilitam a utilização e a inserção de um grande volume de dados sem preocupações com a disrupção da estrutura prévia do banco. Dessa forma, o paradigma que faz uso de grafos é mais adequado para situações em que se precisa de escalabilidade.

Como dito anteriormente, um banco de dados de grafos RDF é comumente usado no escopo da Web Semântica e também chamado de *triplestore*. Um *triplestore* armazena os dados obtidos de arquivos em formatos de serialização do modelo RDF (ex.: N3, Turtle, RDF/XML, JSON-LD, etc.). Esses, por sua vez, podem ser manipulados a partir da linguagem de consulta *SPARQL Protocol and Query Language*[6] (SPARQL). De forma análoga ao SQL para os bancos relacionais, a linguagem SPARQL permite a realização de operações ligadas à consulta e alteração de informações. Contudo, como o SPARQL, diferentemente do SQL, atua sobre grafos e não tabelas, as opções de recuperação são mais flexíveis, permitindo a busca por padrões de conexões, não se limitando a estruturas fixas.

2.3 PADRÃO GTFS

O movimento da Internet das Coisas e o surgimento das chamadas Cidades Inteligentes levou à criação de iniciativas que visam resolver ou reduzir os problemas enfrentados nas cidades. Sendo uma das maiores tribulações enfrentadas pelas metrópoles, é natural que iniciativas também surjam no campo da mobilidade urbana. No entanto, soluções pensadas nessa escala geram um grande volume de dados que necessitam ser geridos. Logo, é natural que o primeiro passo para essa gestão seja estabelecer padrões para o armazenamento desses dados. Entre eles, um dos mais populares, principalmente para dados relacionados à infraestrutura de linhas de ônibus, é o *General Traffic Feed Specification*[7] (GTFS).

O GTFS é um padrão aberto desenvolvido pela empresa Google e dispõe de padrões tanto para representar os dados estruturais do transporte público (linha, paradas, horários, etc.), o GTFS *Schedule* (estático), quanto para representar as informações em tempo real (localização atual, imprevistos, etc.), o GTFS *Realtime* (tempo real). No entanto, para o escopo abordado no presente projeto, será considerado apenas o GTFS *Schedule*, que será referido apenas como GTFS, como é comum na literatura.

No formato GTFS, a informação é armazenada em arquivos tabulares, que são classificados de acordo com a necessidade de sua presença. Essa última podendo ser

[6] https://www.w3.org/TR/sparql11-query/.

[7] https://gtfs.org/pt-BR/.

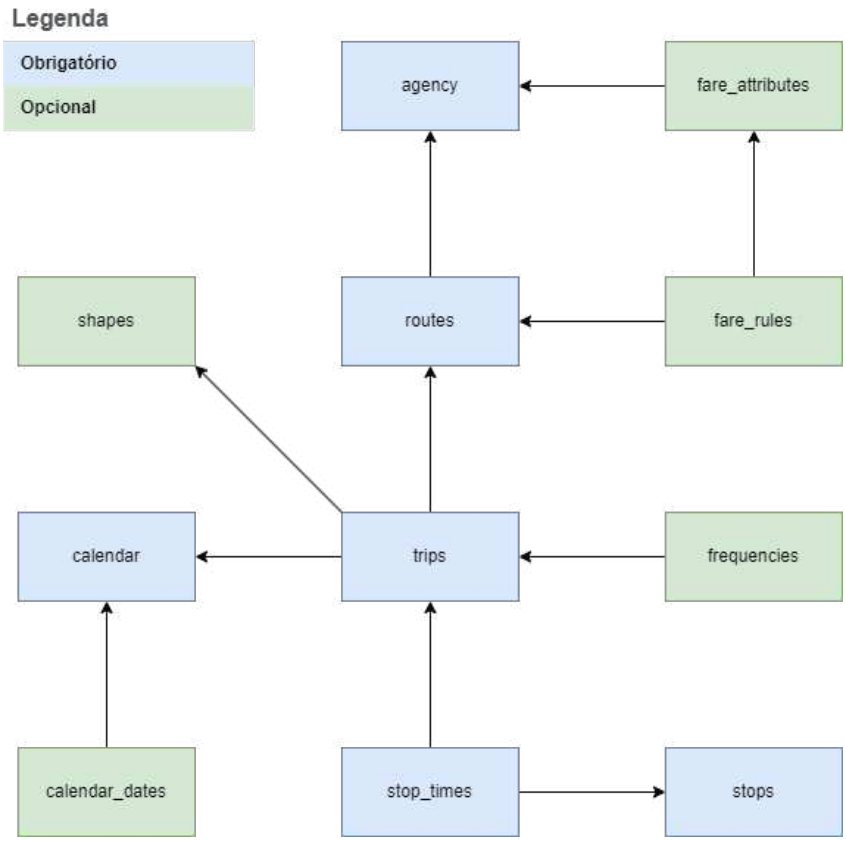
obrigatória, obrigatória sob certas condições, ou opcional. A escolha dos arquivos considerados nesse trabalho (Quadro 1) foi em razão de serem os disponibilizados pela fonte de dados utilizada, como será descrito em mais detalhes na seção 3.1.

Quadro 1– Arquivos do padrão GTFS.

ARQUIVO	CONTEÚDO
agency.txt	Dados das organizações responsáveis pelas linhas contidas no conjunto de dados.
calendar.txt	Informações sobre os dias da semana e período de tempo que um dia de serviço atua.
calendar_dates.txt	Apresenta exceções nas operações dos dias de serviço.
fare_attributes.txt	Contém os tipos de tarifas que podem ser cobradas em uma linha.
fare_rules.txt	Relação de uma linha ao tipo de tarifa cobrado nela.
feed_info.txt	Dados sobre o *feed* onde à implementação do GTFS foi disponibilizada.
frequencies.txt	Frequência das viagens.
routes.txt	Informações sobre as linhas, grupo de percursos, contidas no conjunto dados.
shapes.txt	Contém as coordenadas que compõe os trajetos das viagens.
stop_times.txt	Sequência e momento na viagem que o veículo para em uma parada.
stops.txt	Informação sobre as paradas, pontos de acesso a uma linha.
trips.txt	Descrição das viagens, percurso contidos em uma linha.

O diagrama mostrado na Figura 2 apresenta o recorte de arquivos relevantes para este trabalho e a forma que eles interagem. Na Figura 2, uma seta partindo de um arquivo “A” para um arquivo “B” indica que um dos atributos do arquivo “A” referencia o atributo identificador do arquivo “B”.

Figura 2 – Diagrama dos Arquivos do Padrão GTFS



2.4 FERRAMENTAS DE ETL (*EXTRACT TRANSFORM LOAD*)

Durante o processo de instanciação de um banco de dados, é comum se buscar formas de automação, seja pelo volume dos dados a serem inseridos, seja porque a forma que os dados estão contidos na fonte não está de acordo com as especificações do banco. Chamamos de *Extract Transform Load* (ETL) o conjunto de passos tomados para automatizar a extração, a carga e a transformação de múltiplas fontes de dados.

2.4.1 A Ferramenta Talend Open Studio

Um exemplo de ferramenta que pode ser usada na implementação de processo ETL é o Talend Open Studio[8], ou, simplesmente, Talend. Neste trabalho, os fluxos que estão ligados aos bancos relacionais para extração, transformação e carga dos dados contidos nos arquivos GTFS de entrada utilizados são construídos por meio dessa ferramenta. Esse *software*, desenvolvido para processos de ETL, oferece uma interface de programação gráfica para a criação

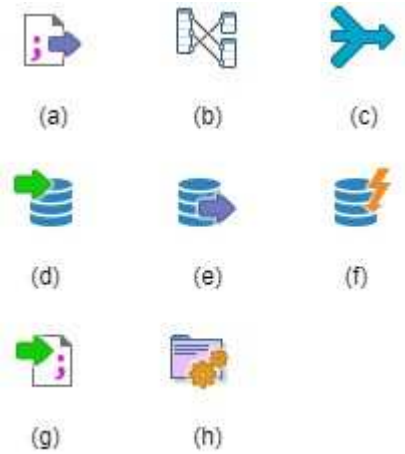
[8] https://www.talend.com/products/talend-open-studio/

de uma rotina que será traduzida para linguagem Java[9] e executada. O foco principal da ferramenta é a integração entre diferentes fontes de dados.

A estrutura das rotinas, ou *jobs*, como são chamados pelo Talend, é organizada da seguinte forma: um *job* é composto por *subjobs* que, por sua vez, são compostos por um ou mais componentes. Os *subjobs* são delimitações de um grupo de componentes que interagem entre si e se relacionam com outros *subjobs* com o intermédio de *triggers* (gatilhos). Já os componentes, são elementos do Talend que realizam alguma ação, abrangendo desde o acesso a um banco de dados até a criação e escrita de arquivos.

O Talend disponibiliza diversos componentes, no entanto, para os fins deste trabalho, apenas sete foram utilizados na construção dos fluxos. Foram eles: `tFileInput_Delimited` (Figura 3a), `tMap` (Figura 3b), `tUnite` (Figura 3c), `tDBOutput` (Figura 3d), `tDBInput` (Figura 3e), `tDBRow` (Figura 3f), `tFileOutput_Delimited` (Figura 3g) e `tJavaFlex` (Figura 3h).

Figura 3 – Componentes do Talend Open Studio



O `tFileInput_Delimited` é um componente que tem como função a leitura de arquivos externos que têm seus campos demarcados por algum separador e seus itens separados por marcador de fim de linha, como, por exemplo, o formato CSV. Para seu funcionamento, são necessários o caminho onde se encontra o arquivo e um esquema (Figura 4). O esquema é algo comum a todos os componentes do Talend que lidam com dados. Ele dita os campos que aquele componente recebe como entrada ou, em algumas situações, retorna como saída, bem como o tipo de dado esperado em cada um, além de algumas outras informações a variar de

[9] https://www.java.com/pt-BR/.

acordo com a tipagem. Também é possível determinar propriedades do arquivo lido, tais como: caracter de separador, caracter de fim de linha, o número de linhas a serem consideradas como cabeçalho, etc. (Figura 5).

Figura 4 – Exemplo de esquema no Talend

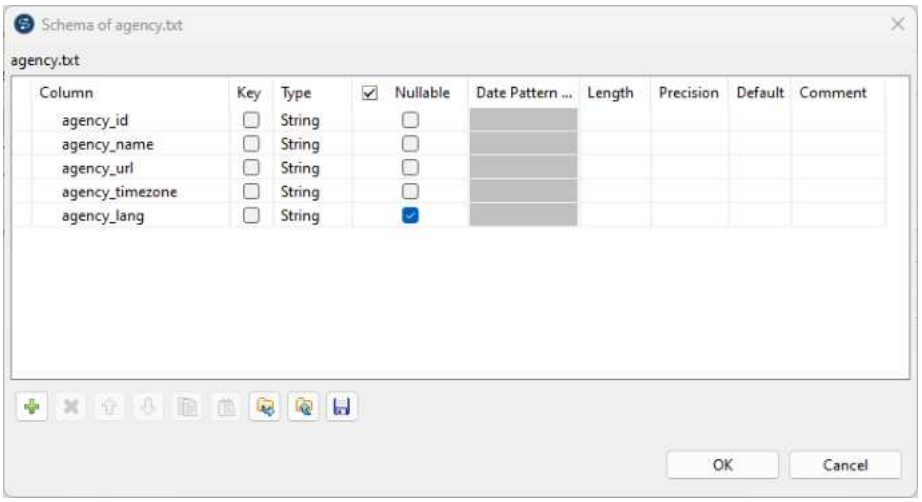


Figura 5 – Interface principal do componente "tFileInput_Delimited"

Em seguida, o componente `tMap` é um dos principais quando se tratam dados de origens distintas. Esse componente é capaz de realizar operações de álgebra relacional (interseção, união, etc.) entre fontes diferentes, filtragem de dados (Figura 6) e replicação de fluxo de dados (Figura 7). A principal função do `tMap` no processo de ETL foi a de um papel similar ao *join* (por padrão *left join*) entre as distintas possíveis origens dos fluxos de dados (bancos ou arquivos). Isso é feito utilizando a interface "arrasta e solta" do componente (Figura 8) para relacionar campos de fluxos de dados diferentes como chaves desses *joins* (similar à cláusula *on do SQL*). É ainda possível realizar manipulações com os dados vindos de um fluxo, utilizando funções nativas da ferramenta ou código da linguagem Java.

Figura 6 – Campo de filtro do componente "tMap"

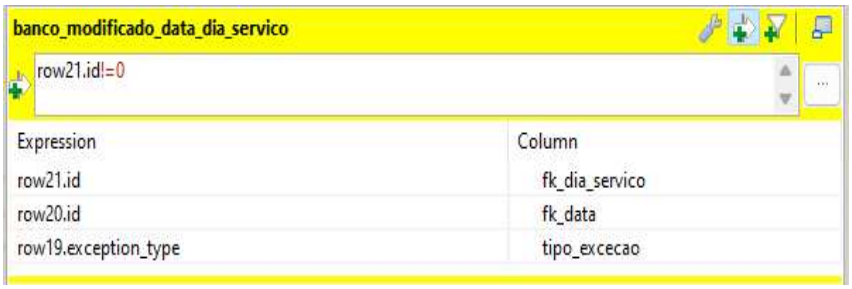
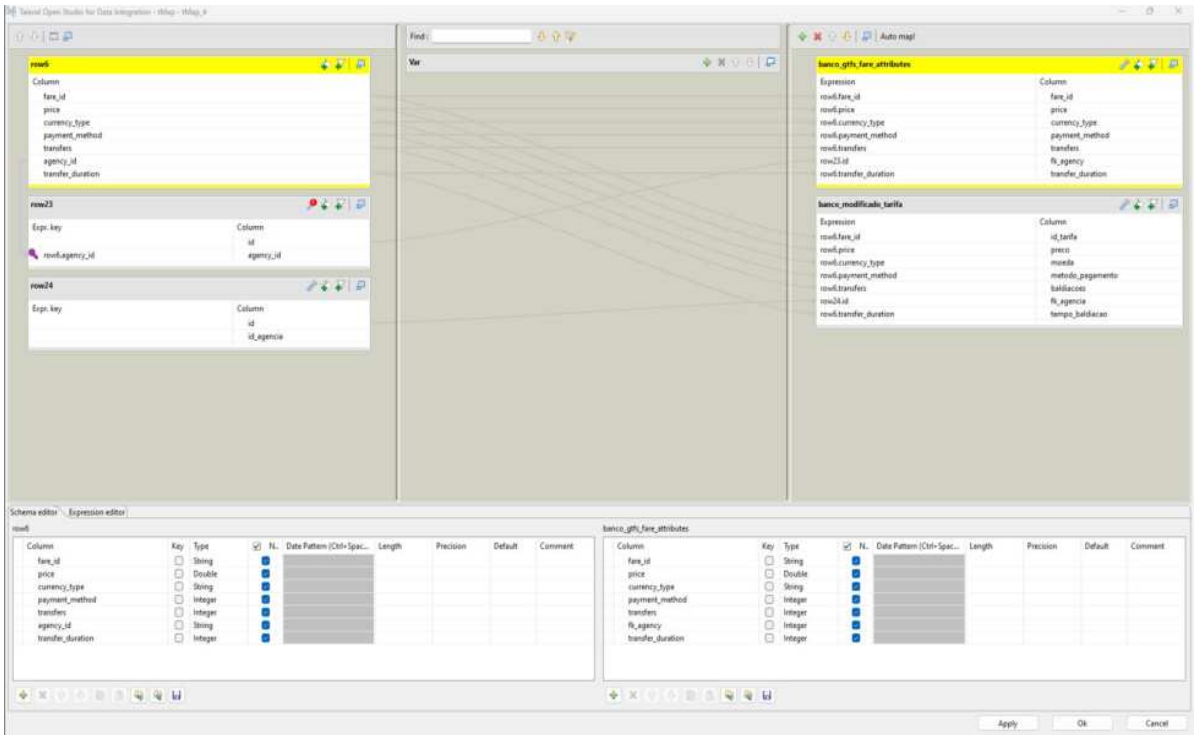


Figura 7 – Componente “tMap” sendo utilizado para duplicar um fluxo de dados

Figura 8 – Interface do componente “tMap”



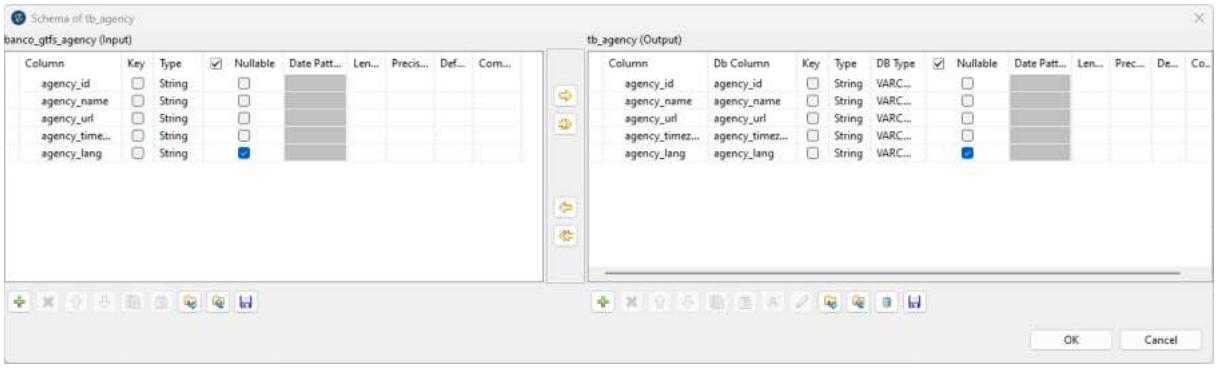
O componente `tUnite` também lida com múltiplas fontes de dados, porém seu foco são naquelas que compartilham o mesmo esquema. A operação que ele realiza é similar ao *union all* do SQL, isto é, uma união que permite repetição dos elementos dos dois conjuntos. Com exceção do esquema, o componente não possui nenhuma outra propriedade.

Já o componente `tDBOutput` atua na escrita de dados em um banco. O Talend fornece variações desse componente para diferentes SGBDs. Portanto, para este trabalho, foi utilizada a variação específica para o PostgreSQL. Para conexão e escrita bem-sucedidas no banco, o componente necessita das informações de conexão como: endereço do servidor onde o banco está hospedado, nome do banco, nome do esquema contendo a tabela, nome de usuário, senha para acesso e nome da tabela (Figura 9). Assim como nos outros componentes, também é preciso especificar o esquema de entrada e de saída do componente. Embora o esquema de entrada seja similar aos já vistos anteriormente, para o `tDBOutput`, o esquema de saída deve descrever a tabela a ser escrita, utilizando os mesmos nomes de campos e tipos

que o SGBD (Figura 10). Além disso, também deve ser especificado o tipo de alteração a ser realizada no banco (Figura 9): “inserir”, “atualizar”, “inserir ou atualizar”, “atualizar ou inserir” e “deletar”. As operações associadas aos tipos “inserir”, “atualizar” e “deletar” são auto-descritivas. Já, no caso dos tipos “inserir ou atualizar” e “atualizar ou inserir”, estes realizam uma das operações, associadas em seus nomes, dependendo se o valor do campo escolhido como chave já existe (ou não) na tabela alvo. A diferença entre elas é uma questão de otimização. Deve-se escolher o tipo que menciona primeiramente a operação de escrita que tem maior probabilidade de ocorrer, ou seja, caso seja esperado que mais inserções aconteçam do que atualizações, a melhor escolha é a opção “inserir ou atualizar”.

Figura 9 – Interface do componente “tDBOutput”

Figura 10 – Esquema do componente “tDBOutput”



O próximo dos componentes a ser apresentado, o `tDBInput`, é, até certo ponto, similar ao `tDBOutput`. Ambos necessitam das informações para conexão com o SGBD (Figura 12), seus esquemas são análogos (Figura 11). No entanto, a principal diferença é que esse componente, diferentemente do anterior, é utilizado para importar os dados do banco apontado no esquema. De maneira mais específica, o `tDBInput` retorna o resultado da execução de uma consulta (Figura 12). Como consequência, não é necessário especificar qual a tabela alvo na interface, dado que essa informação já está contida na consulta.

Figura 11 – Esquema do componente “tDBInput”

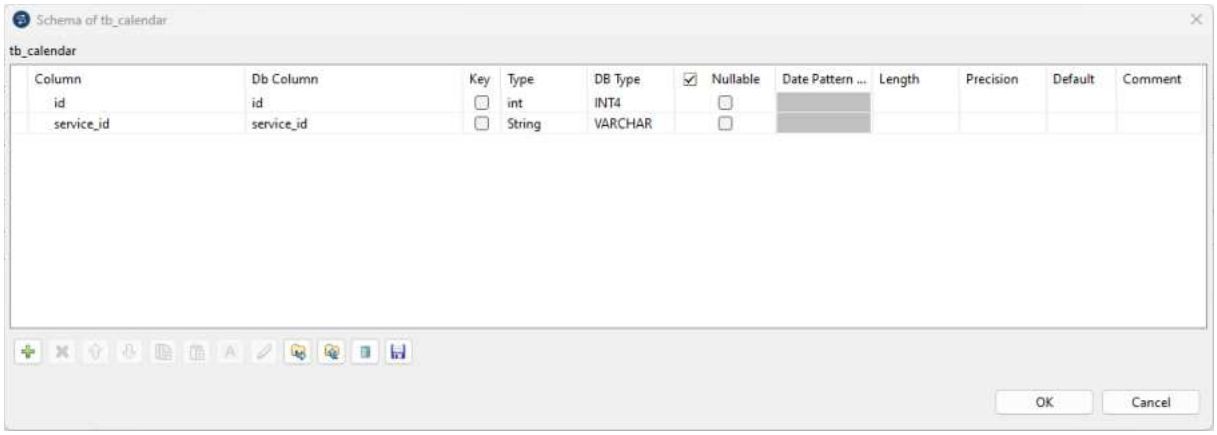


Figura 12 – Interface do componente “tDBInput”

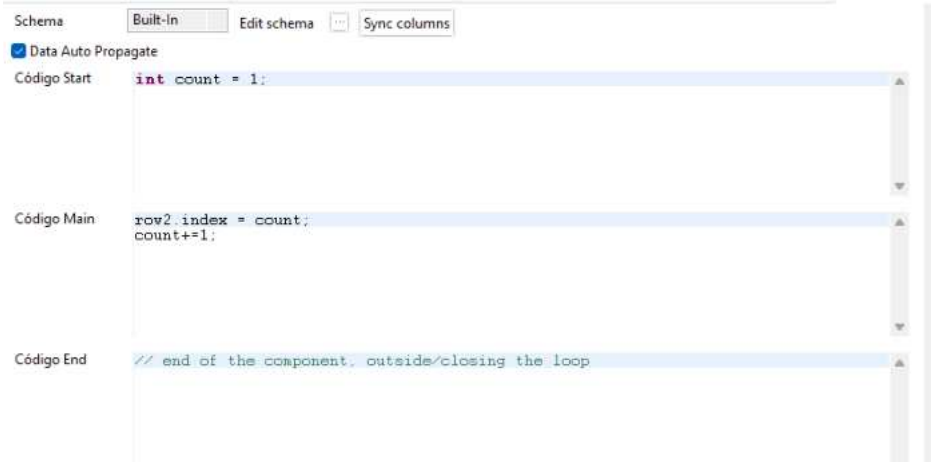
O terceiro dos componentes a lidar diretamente com o banco de dados, o `tDBRow` pode ser visto como uma generalização dos componentes `tDBOutput` e `tDBInput`. Em vez de executar uma ação pré-determinada no banco de dados, o componente permite que o usuário escreva um código SQL a ser executado. No mais, estruturalmente, é muito similar aos anteriores, é necessário especificar as informações de conexão com o banco e o esquema segue o mesmo padrão que no `tDBInput`.

O componente `tFileOutput_Delimited` age de forma oposta ao `tFileInput_Delimited`. Sua função é escrever ou sobrescrever um arquivo tabular com o conteúdo recebido e o cabeçalho especificado (Figura 13). Estruturalmente, esse componente é bastante similar ao `tFileInput_Delimited`, tanto na interface quanto em seu esquema.

Figura 13 – Interface principal do componente “tFileOutputDelimited”

O último dos componentes, `tJavaFlex` assume uma função “coringa”. O componente reproduz o código Java contido nele no código final gerado pela ferramenta Talend. A interface principal do componente (Figura 14) é divida em três partes: *Start* (início), *Main* (principal) e *End* (fim). O posicionamento do código na interface indica o momento de sua execução no *subjob*. Um código escrito na seção *Start* é a primeira ação executada pelo *subjob*, independente de sua posição no fluxo. Analogamente, códigos localizados na seção *End* tem sua execução adiada até o fim do *subjob*. Para a seção *Main,* códigos presentes nela são executados todas as vezes que o fluxo do *subjob* passa por esse componente.

Figura 14 – Interface principal do componente “tJavaFlex”



Ainda há uma outra estrutura relevante para o processo de ETL, chamada `row`. As *rows* (Figura 15) são conexões, entre componentes, responsáveis pela transferência de dados entre eles.

Figura 15 – Exemplo de conector *row*

2.4.2 Triplificação de Dados com OpenRefine e RDF Extension

Uma aplicação de ETL específica ao se lidar com Web Semântica e o formato RDF é a automatização da triplificação de dados. O processo de triplificação consiste em converter dados em um formato qualquer para o formato RDF com a criação de triplas (sujeito, predicado, objeto), seguindo uma ou mais ontologias.

Esse processo pode ser implementado por diferentes ferramentas. Uma delas é o OpenRefine[10], uma ferramenta para a limpeza e transformação de dados. O software, com sua extensão para o formato RDF, RDF Extension[11], permite a conversão automática dos arquivos-fonte, caso um mapeamento seja oferecido.

Em mais detalhes, a ferramenta oferece uma interface para a criação de triplas a partir dos campos do arquivo-fonte (Figura 16). Cada nó da tripla pode ser gerado por meio da interface RDF *Node* (Figura 17). Essa interface disponibiliza as opções de campos para gerarem o nó, no caso de nós objetos que são literais, descrever o tipo do dado e, para IRIs, definir o *namespace* utilizado. Os *namespaces* disponíveis devem ser previamente adicionados indicando os prefixos e IRI a que servem como abreviação (Figura 18). Além disso, o OpenRefine permite manipular o valor de um nó de diferentes formas além de simplesmente a escolha da origem do valor pela seção *content* (Figura 17). Isso é possível por meio da GREL (*General Refine Expression Language*), a linguagem de manipulação de expressões nativa do software. Ela permite que sejam realizadas operações mais complexas na criação de nós, tais como: atribuição condicional, concatenação de campos e termos, etc.

[10] https://openrefine.org/.
[11] https://github.com/stkenny/grefine-rdf-extension.

Figura 16 – Interface RDF *Transform*

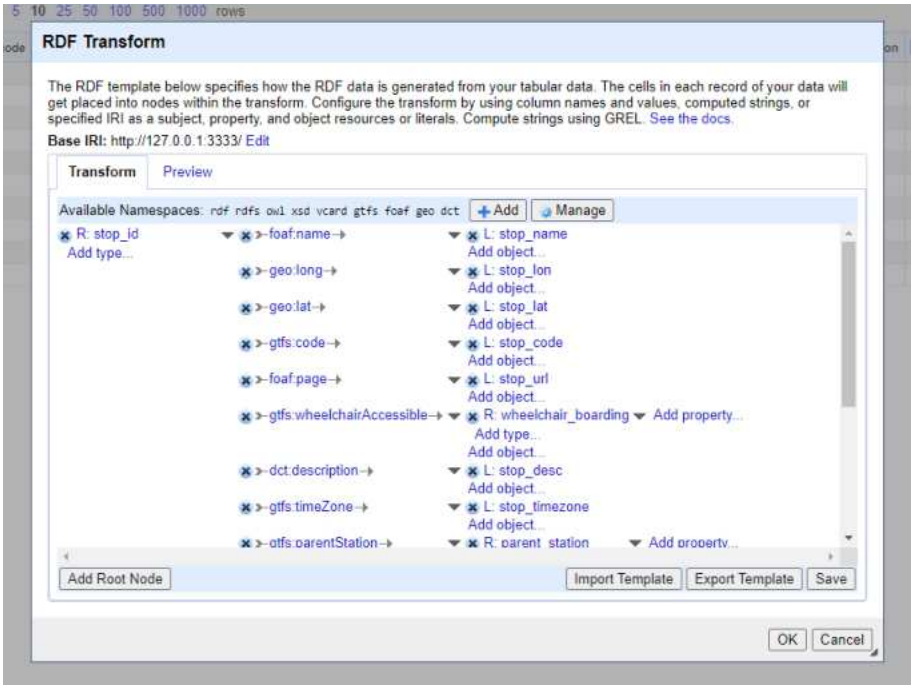


Figura 17 – Interface RDF *Node*

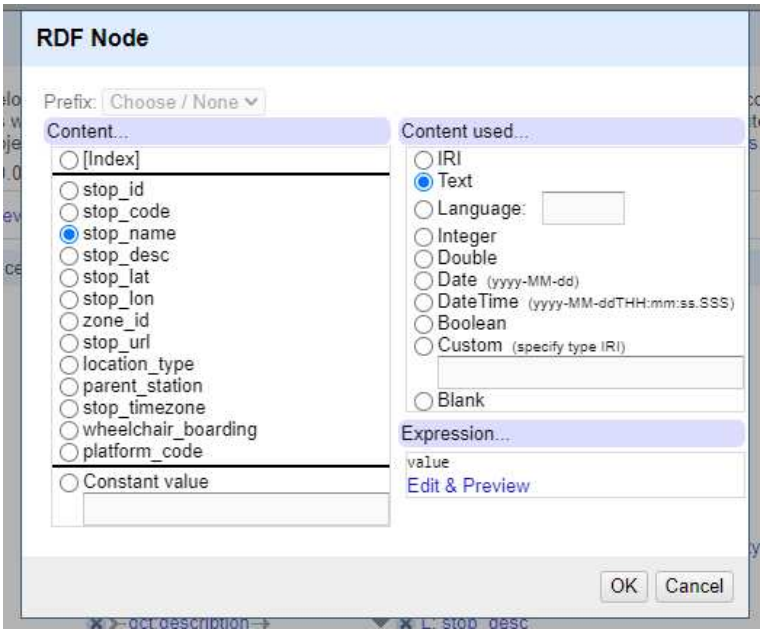
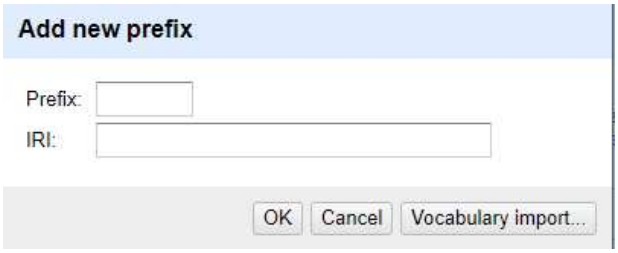


Figura 18 – Interface *Add New Prefix*



No fim, é possível exportar o arquivo gerado em diferentes formatos de serialização (Turtle, RDF/XML, JSON-LD, etc.). Neste trabalho, o formato escolhido foi o Turtle em razão da legibilidade de arquivos desse padrão e a possibilidade de importação direta deste formato pelo *triplestore* adotado (GraphDB).

2.5 TRABALHOS RELACIONADOS

Ao se estudar a literatura, podem ser encontrados artigos discutindo iniciativas similares ao experimento abordado neste trabalho. Esses artigos estão relacionados aos temas abordados em, pelo menos, uma das duas formas: (i) evidenciando as aplicações de modelos de grafos para análises de dados relacionados à mobilidade urbana ou (ii) comparando o uso de banco de dados de grafos e relacionais.

No caso de trabalhos sobre o uso de grafos em iniciativas envolvidas com o estudo e análise de dados da mobilidade urbana em cidades, podemos citar (GUBERT et al., 2020), por exemplo. Os autores demonstram o uso de um grafo de mobilidade como ponto de partida para uma análise multicamadas dos padrões espaço temporais do deslocamento, no transporte público da cidade de Curitiba, no estado do Paraná.

Em uma linha similar, Diniz, Cunha e Loureiro (2020) apontam para os usos de diferentes tipos de grafos no entendimento da maneira que as redes responsáveis pela troca de informação entre veículos se comportam. Embora os últimos dois artigos citados não entrem em detalhes na forma e implementação dos bancos de dados que suportaram as análises feitas, os experimentos descritos demonstram o potencial do armazenamento na forma de grafos para a exploração de dados do campo da mobilidade urbana.

Já Fortin, Morency e Trépanier (2016) trazem uma abordagem mais focada na implementação de um modelo de grafos no armazenamento de dados. Assim como no presente trabalho, o foco é em adaptar o padrão GTFS para esse tipo de modelo, no entanto, a implementação dos autores é direcionada ao Neo4J[12] um banco de dados de grafos de propriedade.

Além disso, no âmbito das comparações entre modelos relacionais e de grafos, Vicknair et al. (2010) provêm estratégias para comparação entre um banco de dados de grafos e um relacional. Contudo, o foco do artigo está nos bancos de dados de grafos de propriedade, um paradigma diferente do modelo em RDF descrito anteriormente.

[12] https://www.neo4j.com.

Mais similar a este trabalho, Chaves-fraga et al. (2020) fazem a comparação entre diferentes modelos de bancos de dados, incluindo *triplestores* e bancos relacionais, com o objetivo de avaliar diferentes mecanismos de OBDI (*Ontology Based Data Integration*), que permitem a realização de consultas em um grafo de conhecimento virtual, onde os dados podem estar armazenados em bancos diferentes, de paradigmas distintos. Embora o foco do trabalho não seja o desempenho isolado de cada paradigma de armazenamento de dados, mas sim o desempenho em conjunto, em um contexto de integração, ele faz uso tanto de um conjunto de dados no padrão GTFS tabular como de uma versão desse conjunto mapeada utilizando a mesma ontologia a ser aplicada no presente trabalho. Sendo assim, as métricas e consultas elaboradas pelos autores para testes de desempenho serviram como base para os experimentos futuramente descritos no presente trabalho.

3 PREPARAÇÃO DOS BANCOS DE DADOS

Para comparação entre a forma de armazenamento relacional e a utilizando grafos, é preciso criar diferentes ambientes adaptados para cada situação. Esta seção foca nos passos tomados para que esses ambientes sejam estabelecidos. São detalhados os processos desde a escolha da fonte de dados, passando pela criação dos bancos de dados, relacional e de grafos, para contê-los, e finalizando com as etapas para as cargas de dados nesses bancos.

3.1 FONTE DE DADOS

Antes de se pensar em qualquer estrutura para os bancos que seriam criados para a realização dos testes, foi preciso fazer um levantamento e estudar como os dados da infraestrutura de modais de transporte são disponibilizados. Nesse levantamento, foram considerados, como possíveis fontes, os portais de dados abertos disponibilizados por entidades governamentais. Nesta seção, são destacados três desses portais que foram analisados: Portal Brasileiro de Dados Abertos[13], Governo Aberto SP[14] e DATA.RIO[15]. Sendo o portal DATA.RIO, o escolhido como fonte de dados principal para esse trabalho. A seguir, são discutidos estes portais e o formato dos dados sendo tratados.

3.1.1 Portal Brasileiro de Dados Abertos (dados.gov.br)

O portal é uma iniciativa a nível federal, criado a partir da promulgação da Lei de Acesso à Informação de 2011, e que tem como objetivo ser o ponto central para a busca e acesso de dados de diferentes níveis da federação (BRASIL,2019). Com isso, o site dados.gov.br apresentava-se, teoricamente, como uma boa opção para se estudar aplicações do formato GTFS em diferentes regiões do Brasil, a fim de se melhor entender os processos necessários no carregamento e armazenamento dos dados.

Entretanto, mesmo contendo 13.141 conjuntos de dados até a data de 12 de junho de 2024, os únicos conjuntos contendo dados seguindo o padrão GTFS encontrados foram os referentes à infraestrutura de ônibus de Belo Horizonte, disponibilizados pela prefeitura da

[13] https://dados.gov.br/.

[14] https://www.governoaberto.sp.gov.br/.

[15] https://www.data.rio/.

cidade. Esses conjuntos contemplavam tanto o sistema convencional quanto suplementar e continham todos os arquivos requeridos do formato e mais alguns opcionais. Porém, posteriormente, foi encontrado no Data.rio, que será discutido futuramente, um conjunto de dados mais detalhado e mais recentemente atualizado. Dessa forma, o GTFS de Belo Horizonte não foi escolhido como fonte para popular os bancos de dados criados neste trabalho.

3.1.2 Governo Aberto SP

No caso deste próximo portal abordado, o Governo Aberto SP[16], ele foi escolhido como uma possível fonte dada, assim como no repositório anterior, a expectativa de haver um detalhamento dos conjuntos de dados de uma grande metrópole. Contudo, o portal operado pelo governo do estado de São Paulo não continha diretamente o conjunto de dados procurado e sim um *link* para o repositório da Empresa Metropolitana de Transporte Urbano de São Paulo (EMTU).

Os dados presentes no repositório seguem o padrão GTFS e contemplam a infraestrutura de ônibus das 5 metrópoles paulistas. No entanto, não há especificação de quais cidades são contempladas por essa terminologia. Também há questões sobre a atualidade dos dados, pois sua última atualização até a data de observação, 24 de abril de 2024, aconteceu no ano de 2020. Esses fatores contribuíram para a não escolha deste portal como fonte dos dados.

3.1.3 Portal DATA.RIO

Por fim, o portal escolhido como fonte para o experimento foi o DATA.RIO que disponibiliza uma série de conjunto de dados sobre diversos aspectos da administração pública da cidade do Rio de Janeiro. Entre eles, o GTFS contendo os dados de mobilidade urbana da cidade. O repositório é mais completo em relação ao das 5 Metrópoles paulistas, quando observado do ponto de vista dos arquivos GTFS que armazena, contendo 15 arquivos (Quadro 1) enquanto o de São Paulo contém 11. Isso permite que uma maior variedade de consultas possam ser executadas sobre o conjunto de dados. Além disso, no momento de elaboração deste trabalho, o primeiro semestre de 2024, os registros são atualizados mensalmente.

[16] https://www.governoaberto.sp.gov.br/.

3.2 BANCOS DE DADOS

Escolher um conjunto de dados permitiu definir para quais dos arquivos do padrão GTFS seria necessário projetar estruturas para seu armazenamento. Esse processo envolve tanto a adaptação do arquivo para modelos de banco de dados quanto a escolha de sistemas de bancos de dados (SGBD) a serem utilizados para cada um deles. Nesse trabalho, foi decidido pela implementação de bases de dados seguindo os modelos relacional e de grafos (triplas RDF). Assim, foram projetados três bancos. Dois deles seguindo o modelo relacional e o terceiro implementado utilizando grafos de triplas RDF.

3.2.1 Bases Relacionais

Como dito anteriormente, duas implementações relacionais foram pensadas. A diferença entre elas está na forma de representação do modelo GTFS. A primeira abordagem faz apenas alterações mínimas (criação de "*dumb keys*" e criação de restrições de chaves estrangeiras) à estrutura já existente das tabelas dos arquivos disponibilizados. Já a outra buscou um nível maior de normalização. Os bancos gerados por essas duas abordagens foram nomeados "Banco GTFS" e "Banco Modificado" respectivamente.

3.2.1.1 Banco GTFS

O Banco GTFS foi construído seguindo a mesma estrutura que a disponibilização dos arquivos GTFS. No modelo da Figura 19, é possível perceber que existe uma tabela de mesmo nome para cada arquivo do padrão, com exceção de `feed_info.txt`. Os campos de cada tabela, em geral, se mantiveram em relação ao arquivo GTFS correspondente. Contudo, duas modificações foram feitas. Primeira, dado que as entidades do GTFS possuem identificadores principalmente textuais, foram criados campos "id" de valor numérico para agirem como "*dumb keys*" por motivos de otimização. A outra mudança foi a criação da restrição de chave estrangeira para certos campos que fazem referência a identificadores de outras relações (tabelas), assim como a substituição desses identificadores pelos usados no banco.

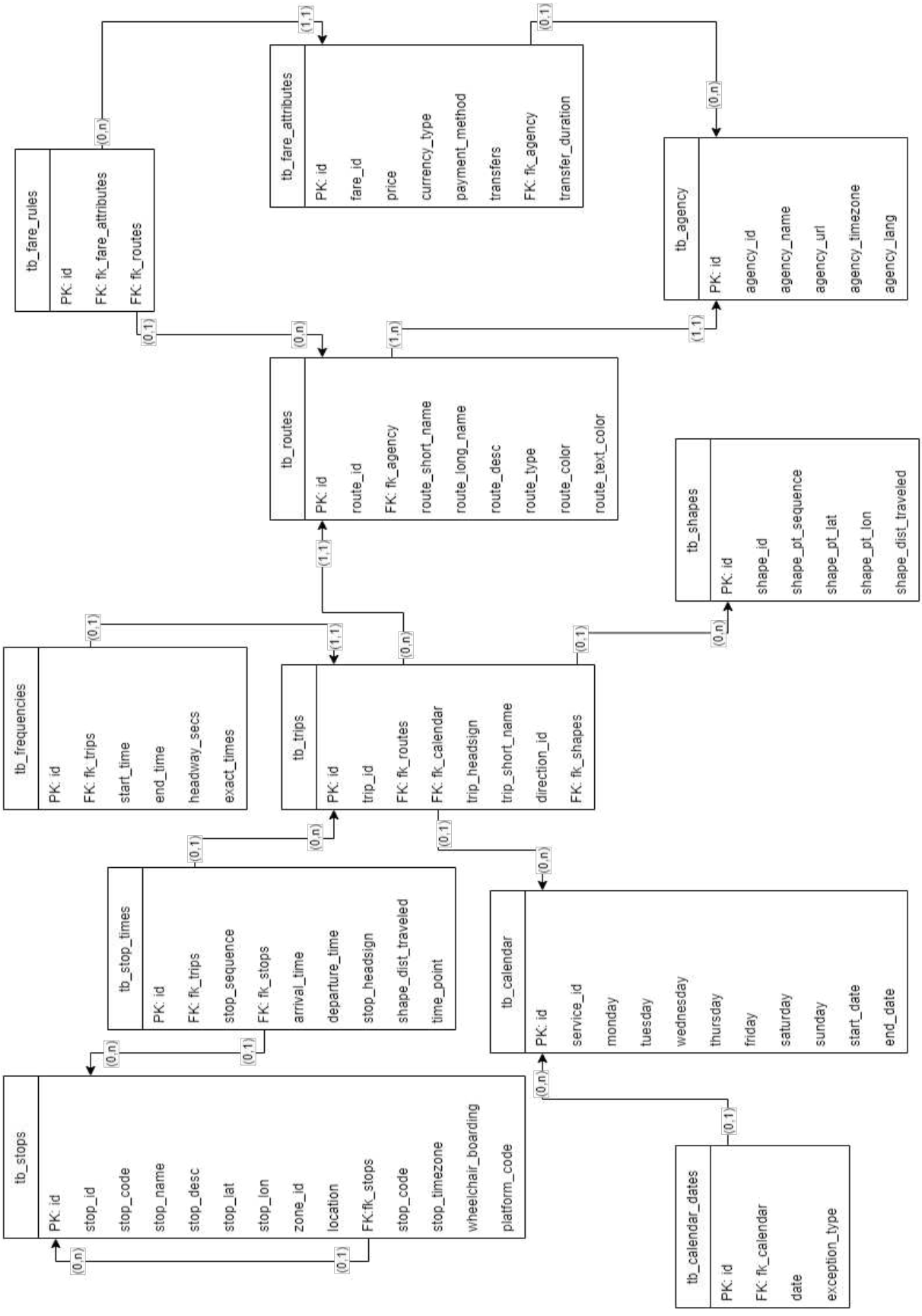


Figura 19 – Modelo lógico de banco GTFS

3.2.1.2 Banco Modificado

O Banco Modificado surge com o objetivo de apresentar uma representação mais normalizada para o GTFS. O objetivo do processo de normalização era reduzir redundâncias e avaliar, na comparação final, o impacto de diferentes implementações relacionais. Assim, os dados dos arquivos originais foram reestruturados em tabelas como apresentado no diagrama da Figura 20. No Quadro 2, se observa quais arquivos do GTFS deram origem a quais tabelas do banco Modificado.

Pode-se notar que, na maioria dos casos presentes no Quadro 2, o conteúdo de um arquivo do padrão original é dividido em múltiplas tabelas do banco Modificado. Isso é decorrência do esforço de normalização. Dados como as coordenadas geográficas, por exemplo, presentes na `tb_ponto_espaco` originam-se dos arquivos `stops.txt` e `shapes.txt` e, muitas vezes, o mesmo conjunto de coordenadas se repetia em diferentes instâncias, o que motivou a separação desse dado em uma tabela a parte. Outros casos similares, presentes no quadro, tem explicações análogas.

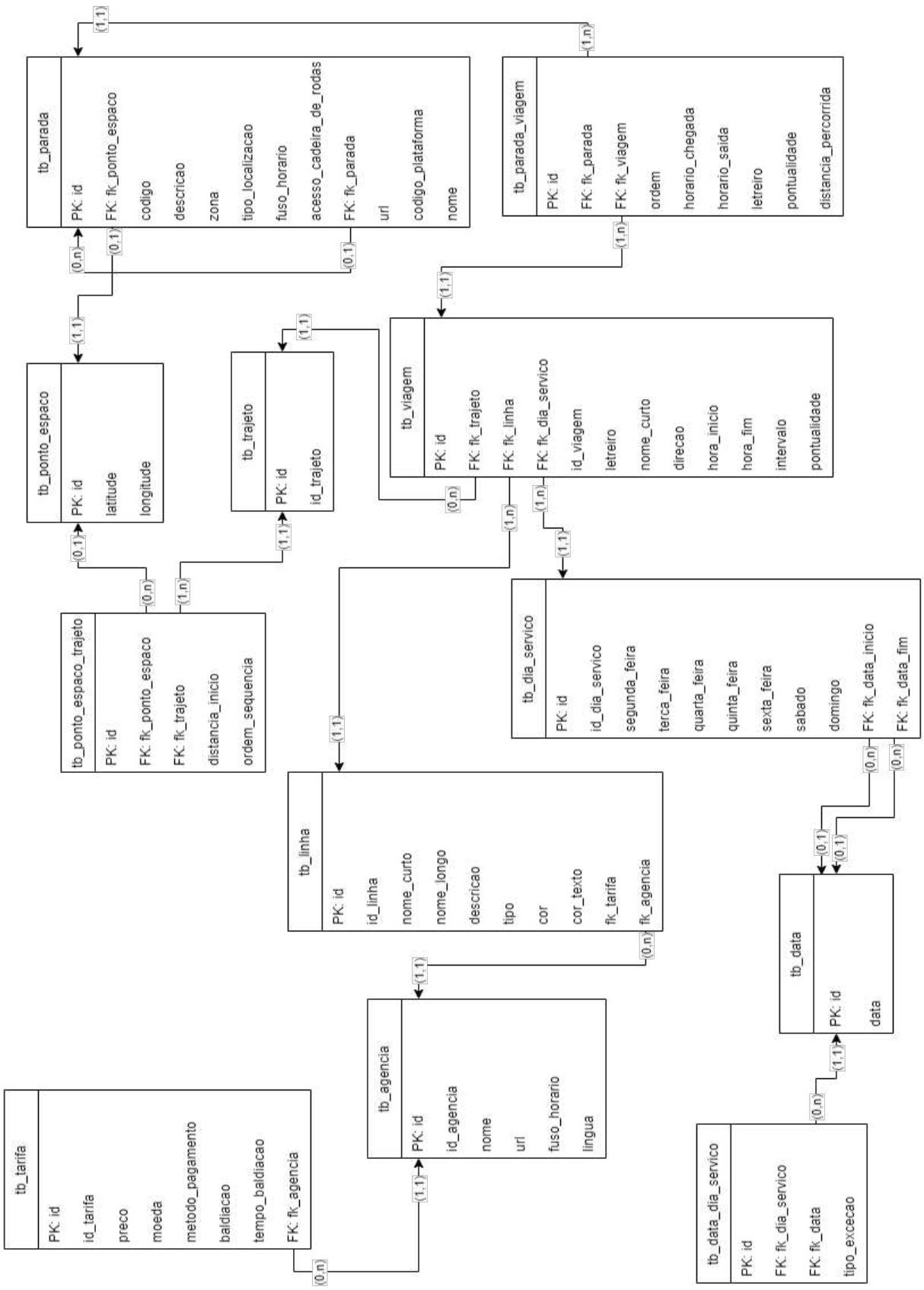


Figura 20 – Modelo lógico de banco Modificado

Quadro 2 – Mapeamento entre os Arquivos do padrão GTFS e as tabelas do Banco Modificado.

ARQUIVO	TABELA
agency.txt	tb_agencia
calendar.txt	tb_data
calendar.txt	tb_dia_servico
calendar_addon.txt	tb_data
calendar_addon.txt	tb_dia_servico
calendar_dates.txt	tb_data
calendar_dates.txt	tb_data_dia_servico
fare_attributes.txt	tb_tarifa
fare_rules.txt	tb_linha
frequencies.txt	tb_viagem
routes.txt	tb_linha
Shapes.txt	tb_trajeto
Shapes.txt	tb_ponto_espaco
Shapes.txt	tb_ponto_espaco_trajeto
Stops.txt	tb_ponto_espaco
stops.txt	tb_parada
stop_times.txt	tb_parada_viagem
trips.txt	tb_viagem

3.2.1.3 Implementação Física

O próximo passo no processo de criação dos bancos são suas implementações. Para isso, é necessária a escolha de um SGBD. Entre as possíveis escolhas, o selecionado foi o PostgreSQL[17]. O sistema foi escolhido em razão de ser código aberto. Além disso, decidiu-se que não seriam criados índices em nenhum dos bancos, além dos automaticamente criados pelo SGBD para as chaves primárias. Os códigos criados para gerar os bancos GTFS e Modificado estão em sua completude nos Apêndices A e B, respectivamente.

3.2.2 Base em Grafo

No caso do banco de grafos, como a abordagem escolhida faz uso de triplas construídas com base em uma ontologia, o esquema do banco é moldado a partir dos dados

[17] https://www.postgresql.org/.

que são inseridos. Dessa forma, a estrutura de um banco em triplas é dependente das ontologias escolhidas. Assim, esta seção será dedicada a apresentar a ontologia utilizada neste trabalho, a *Linked* GTFS Ontology[18], ou simplesmente GTFS *Ontology*.

Esse vocabulário foi criado com a intenção de possibilitar a adaptação do padrão GTFS para o contexto de Web Semântica. A ontologia traz classes representando os arquivos mais comuns disponibilizados por repositórios adotando o padrão GTFS. No entanto, no que se resume aos campos de cada arquivo, é possível que não seja encontrada uma propriedade correspondente, como no caso descrito mais à frente. A Figura 21 demonstra, em detalhe, a organização da ontologia utilizada, já contendo as alterações necessárias descritas nesta seção.

A escolha pela GTFS *Ontology* foi motivada pela adequação da ontologia ao conjunto de dados escolhido, tornado desnecessário mapear os conceitos para uma outra ontologia ou a criação de uma. O objetivo é manter todos os bancos envolvidos ligados ao mesmo modelo conceitual, permitindo que sejam feitas comparações mais diretas sem a necessidade de considerar o mapeamento entre modelos diferentes.

Contudo, não é possível dizer que certas adaptações não foram necessárias para adequar a ontologia ao contexto deste trabalho. Essas adaptações ocorrem em razão de um de dois casos: (i) a demanda por propriedades não descritas no vocabulário original e (ii) a necessidade de atribuir certas classes a termos da ontologia para cumprir requerimentos de domínio. No primeiro dos casos, estão as adições do atributo `fareClass`, descrito na documentação[19], mas não implementado na ontologia, à classe *`FareRule`*, do atributo *`timepoint`* à classe `StopTime` já que não existe termo equivalente no vocabulário e a criação da classe `NoWheelchairAccessInformationAvailable` para cobrir situações em que a informação de acessibilidade de uma parada não é especificada e nem pode ser obtida a partir de sua parada pai. Já no segundo, foi adicionada a classe `Schedule` da ontologia *Schema.org*[20]. A GTFS *Ontology* faz uso de alguns elementos da *Schema.org*. Entre eles, os atributos `endDate` e `startDate` têm a classe `Schedule` como domínio definido[21,22]. Dessa forma, `Schedule` foi adicionada às classes consideradas.

Para o único banco de dados em grafo criado nesse trabalho, o *triplestore* escolhido foi o GraphDB[23]. O SGBD foi escolhido em razão do mesmo ser de código aberto e pelo

[18] http://vocab.GTFS.org/GTFS.ttl.

[19] https://github.com/OpenTransport/linked-GTFS/blob/master/spec.md.

[20] http://schema.org/.

[21] https://schema.org/endDate.

[22] https://schema.org/startDate.

[23] https://graphdb.ontotext.com/.

grande suporte fornecido tendo em vista a existência de uma comunidade ativa de usuários e desenvolvedores.

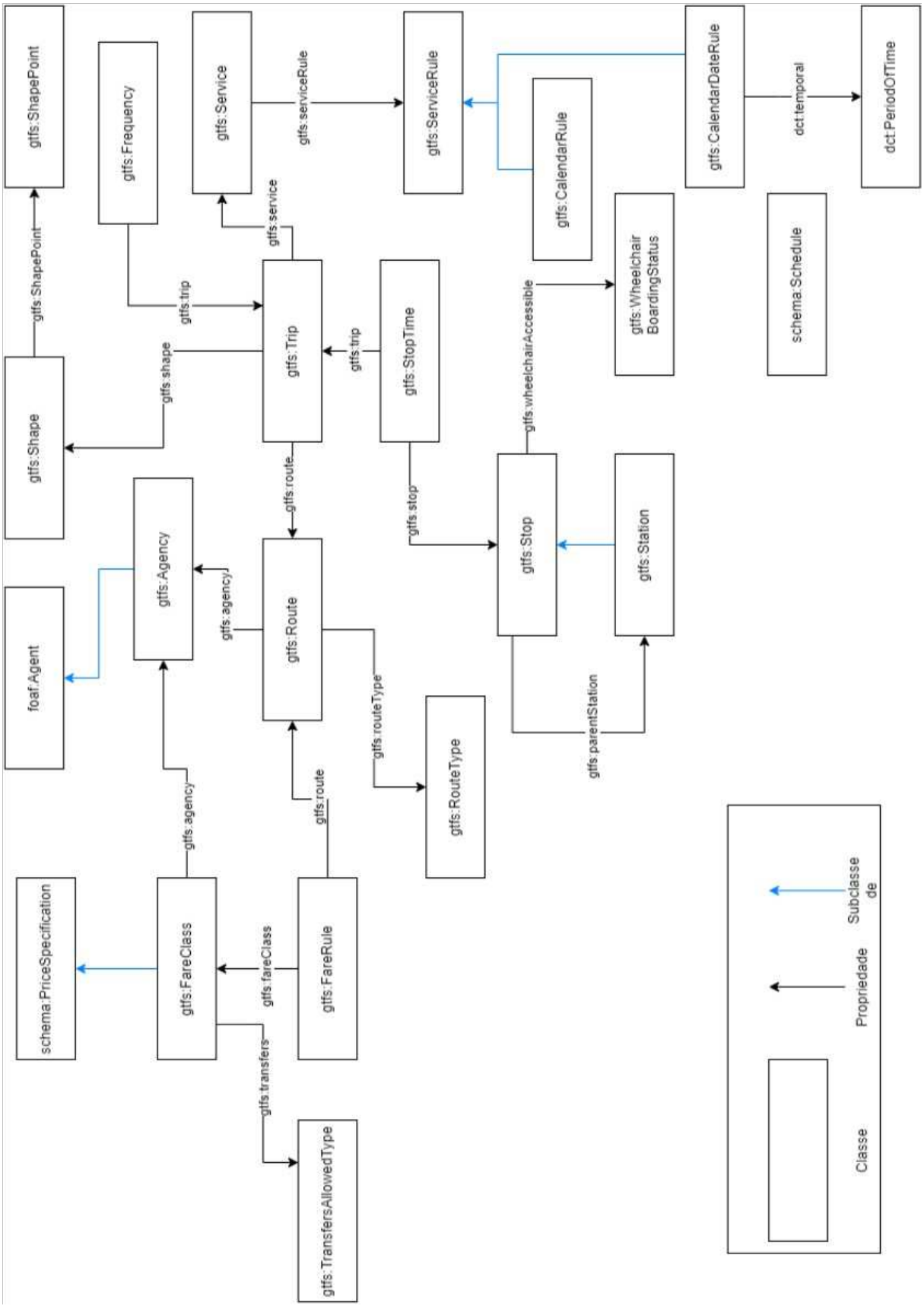


Figura 21 – Recorte de classes da *Linked* GTFS *Ontology* utilizado

3.3 PROCESSO DE ETL

A partir da definição, escolha e estruturação dos bancos de dados (relacionais e de grafos) e da escolha das fontes de dados a serem usadas, deve-se pensar em processos para carga de dados. Dessa forma, o objetivo é adequar os dados obtidos a partir do portal DATA.RIO para popular os campos das tabelas, no caso relacional, e para gerar as triplas RDF, no caso do banco de grafos.

3.3.1 Bancos Relacionais

Embora existam dois bancos relacionais diferentes, apenas um fluxo de ETL foi criado para popular ambos. Isso porque os bancos foram construídos para armazenarem o mesmo conjunto de dados e são alimentados a partir das mesmas fontes. Assim, a existência de um fluxo único visa evitar acessos repetidos desnecessários aos arquivos-fontes.

3.3.1.1 Arquivos de Entrada

O fluxo de ETL que contempla os bancos relacionais é alimentado por 12 arquivos. Desses arquivos, 11 foram os arquivos disponibilizados pela fonte (Quadro 1) com exceção de `feed_info.txt` e o 12º foi criado para suprir uma necessidade dos bancos projetados. Essa necessidade surge a partir de diferenças entre a implementação do modelo GTFS nos bancos e o conjunto de dados disponibilizados pelo DATA.RIO. Na referência do modelo, é recomendado que os identificadores presentes no arquivo `calendar_dates.txt` no campo “id” estejam referenciando a coluna “id” do arquivo `calendar.txt`. Isso motivou a decisão de implementar as colunas que representam esse campo “id” do arquivo `calendar_dates.txt`, `fk_calendar`, no banco GTFS, e `fk_dia_servico`, no banco modificado, como chaves estrangeiras referenciando as tabelas `tb_calendar` e `tb_dia_servico`, respectivamente. Nos dados usados como base, essa recomendação se mantém na maioria dos casos. No entanto, durante o processo de carga nos bancos, foram descobertos casos, em `calendar_dates.txt`, que contêm referências a identificadores não presentes em `calendar.txt`. Assim, o arquivo `calendar_addon.txt` foi criado para conter os elementos referenciados em `calendar_dates.txt` não presentes em `calendar.txt`, mantendo assim a consistência da restrição de chave estrangeira.

Quadro 3 – Dias de operação

DIA DE SERVIÇO	SEGUNDA-FEIRA	TERÇA-FEIRA	QUARTA-FEIRA	QUINTA-FEIRA	SEXTA-FEIRA	SÁBADO	DOMINGO
D_OBRA_007	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera
D_OBRA_009	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera
D_OBRA_011	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera
D_OBRA_014	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera
D_OBRA_015	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera
D_OBRA_016	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera
D_REG	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera
S_OBRA_007	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera	Não Opera
S_OBRA_009	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera	Não Opera
S_OBRA_011	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera	Não Opera
S_OBRA_014	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera	Não Opera
S_OBRA_015	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera	Não Opera
S_OBRA_016	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera	Não Opera
S_REG	Não Opera	Não Opera	Não Opera	Não Opera	Não Opera	Opera	Não Opera
U_OBRA_007	Opera	Opera	Opera	Opera	Opera	Não Opera	Não Opera
U_OBRA_009	Opera	Opera	Opera	Opera	Opera	Não Opera	Não Opera
U_OBRA_011	Opera	Opera	Opera	Opera	Opera	Não Opera	Não Opera
U_OBRA_014	Opera	Opera	Opera	Opera	Opera	Não Opera	Não Opera
U_OBRA_015	Opera	Opera	Opera	Opera	Opera	Não Opera	Não Opera
U_OBRA_016	Opera	Opera	Opera	Opera	Opera	Não Opera	Não Opera
U_REG	Opera	Opera	Opera	Opera	Opera	Não Opera	Não Opera

Como já mencionado, `calendar_addon.txt` é uma extensão do `calendar.txt` (segue a mesma estrutura). Ambos os arquivos tratam de dias de serviço. Um dia de serviço representa um tipo de dia de operação (dia útil, final de semana, feriado, etc.). Esse é caracterizado por um identificador, os dias da semana em que é ativo, e as datas de início e fim de sua operação[24]. Entretanto, para os dias de serviço no arquivo `calendar_addon.txt`, a única informação concreta possível de se obter, por intermédio dos arquivos já existentes, é a do identificador. Algumas tentativas foram feitas para contornar esse empecilho. A primeira delas foi buscar definições para os dias de serviço, representados pelos identificadores, em algum tipo de dicionário de dados, porém nenhum documento do gênero foi encontrado. A solução adotada consiste em obter as informações faltantes por meio de inferências e presunções a partir do que já existe nos arquivos originais.

Um padrão interessante que pode ser percebido nas ocorrências dos dias de serviço representados no arquivo `calendar_dates.txt`, mas não no `calendar.txt`, é a existência de um dia de serviço dual cujos campos estão descritos no `calendar.txt`. É considerado que dois dias de serviço são duais se eles sempre aparecem com campos `exception_type` de valor oposto na mesma data. Por exemplo, o dia de serviço identificado como `U_DESAT_OBRA_007` sempre aparece no arquivo `calendar_dates.txt` com mesma data do dia de serviço `U_OBRA_007`, porém com o campo `exception_type` de valor oposto, indicando que `U_DESAT_OBRA_007` ocorre apenas em dias que `U_OBRA_007` deveria acontecer mas não acontece. A partir disso, foi assumido que, para esse caso e outros semelhantes, um dia de serviço e seu dual compartilham os mesmos dias de funcionamento e período de existência.

Todavia, há casos onde não se pode constatar a existência de um dual. Para esses, foi observada a relação entre o identificador de um dia de serviço e os dias que esse dia de serviço opera (Quadro 3). Por meio dela, é possível perceber que o identificador descreve em quais dias da semana o dia de serviço opera. De forma mais específica, pode-se inferir que as letras “`D`”, “`S`” e “`U`” representam se o funcionamento é aos domingos, sábados ou dias úteis respectivamente. Com isso, para os casos em que não há um dual, os dias de funcionamento são definidos a partir da primeira letra do identificador. Contudo, ainda é necessário definir uma data de início e de fim para o dia de serviço. Com a ausência de uma referência oficial ou um dia de serviço dual para se basear, foi decidido que a data mais antiga registrada junto do dia de serviço no arquivo `calendar_dates.txt` seria considerada como a `data`

[24] https://gtfs.org/pt-BR/schedule/reference/#calendartxt.

`inicio` no registro do arquivo `calendar_addon.txt`. De forma similar, a data mais recente seria considerada a `data fim`.

3.3.1.2 Fluxo de Carga

Utilizando o Talend, foi criado o *job* `carga_bancos_relacionais`[25] com o objetivo de carregar os arquivos de entrada em ambos os bancos (GTFS e Modificado). Esse *job* é composto por 15 *subjobs* (Quadro 4) divididos de acordo com o arquivo responsável por alimentar seu fluxo e ordenados de forma a respeitar as restrições e relacionamentos do esquema de cada banco, ou seja, tabelas referenciadas são carregadas antes das tabelas que as referenciam. O *subjobs* são executados sequencialmente na mesma ordem em que são apresentados no Quadro 4, ou seja, o *subjob* seguinte começa apenas com o término bem-sucedido do anterior.

Os componentes utilizados para a composição desse fluxo foram aqueles apresentados na seção 2.4.1. Além das configurações necessárias descritas anteriormente na introdução dos componentes, também existem opções avançadas para maior personalização. Para a criação dos *subjobs* desse fluxo, 2 componentes precisaram ter suas opções avançadas configuradas: `tFileInput_Delimited` e `tDBOutput`. No caso do primeiro, definiu-se o UTF-8 como formato padrão para a leitura dos arquivos. Já para o segundo, teve que se determinar quais campos da tabela correspondente serão escritos e/ou atualizados e definir as chaves tanto para atualização quanto para deleção, quando preciso.

Pode-se notar que, no Quadro 4, alguns dos arquivos têm mais de um *subjob* dedicado. Para explicar esse fenômeno e prover um entendimento de como são os processos realizados por esses *subjobs*, dado que eles são similares em todos os casos, serão descritos em detalhes os fluxos `Calendar Dates 1` (Figura 22) e `Calendar Dates 2` (Figura 23).

Primeiramente, a necessidade da carga do arquivo `calendar_dates.txt` estar dividida entre esses dois *subjobs* surge a partir da interseção da estrutura do banco de dados "banco modificado" e a forma de funcionamento do Talend. Ao executar um *subjob,* a ferramenta executa primeiro, uma única vez, os componentes que acessam informação de alguma fonte (`tFileInput_Delimited` e `tDBInput`). Dessa forma, os dados obtidos por uma consulta no banco refletem o estado antes do início da execução. Isso posto, no caso abordado, o arquivo `calendar_dates.txt` alimenta duas tabelas no "banco modificado": `tb_data` e `tb_data_dia_servico`. No caso das inserções em `tb_data_dia_ser-`

[25]O job completo é apresentado no Apêndice C.

vico, é necessário acessar as tabelas tb_data e tb_ dia_servico (Figura 23) para a obtenção das chaves estrangeiras, ou seja, a carga dessas tabelas deve ser feita em *subjobs* anteriores. Outros casos em que a carga de um arquivo não esteve contida em um único fluxo seguem o mesmo princípio.

Quadro 4 – Lista de *subjobs* do "carga_bancos_relacionais"

ORDEM	*SUBJOB*	DESCRIÇÃO
1º	Agency	Realiza a carga dos dados do arquivo agency.txt da tb_agencia no banco modificado e da tb_agency no banco GTFS.
2º	Calendar	Realiza a carga da tb_calendar no banco GTFS. No banco modificado, carrega os dados da tb_dia_servico com exceção das chaves estrangeiras e insere as datas presentes nos arquivos calendar.txt e calendar_addon.txt na tb_data.
3º	Calendar 2	Preenche os campos de chave estrangeira da tb_dia_servico.
4º	Calendar Dates 1	Faz a carga do arquivo calendar_dates.txt na tabela tb_calendar_dates. No banco modificado, insere as datas do arquivo na tb_data.
5º	Calendar Dates 2	Preenche a tabela tb_data_dia_servico no banco modificado.
6º	Fare Attributes	Insere os dados do arquivo fare_attribute.txt na tb_fare_attribures do banco GTFS e, no banco modificado, carrega a tb_tarifa.
7º	Routes	Faz a carga dos dados do arquivo routes.txt na tb_routes no banco GTFS e na tb_linha.
8º	Fare Rules	Realiza a carga dos dados do arquivo fare_rules.txt na tb_fare_rules do banco GTFS e atualiza a chave estrangeira entre tb_tarifa e tb_linha.
9º	Shapes 1	No banco GTFS, insere os dados de shapes.txt na tb_shapes. No banco modificado, os dados são inseridos na tb_ponto_espaco e tb_trajeto.
10º	Shapes 2	Faz a ligação das tabelas tb_ponto_espaco e tb_trajeto por meio da tb_ponto_espaco_trajeto
11º	Trips	Carrega os dados do arquivo trips.txt na tb_trips do banco GTFS e na tb_viagem do banco modificado.
12º	Frequencies	Faz a carga dos dados do arquivo frequencies.txt na tb_frequencies do banco GTFS e na tb_viagem do banco modificado.
13º	Stops 1	Realiza a carga na tb_stops do banco GTFS com exceção do auto-relacionamento de parada pai. No banco modificado, insere as coordenadas de cada parada na tb_ponto_espaco e os campos sem auto-relacionamento da tb_parada.
14º	Stops 2	Preenche o auto-relaciomento tanto da tb_stops no banco GTFS e da tb_parada no banco modificado.
15º	Stop Times	Insere os dados do arquivo stop_times.txt na tabela tb_stop_times do banco GTFS. No banco modificado, preenche a tabela tb_parada_viagem.

Figura 22 – *Subjob* Calendar Dates 1

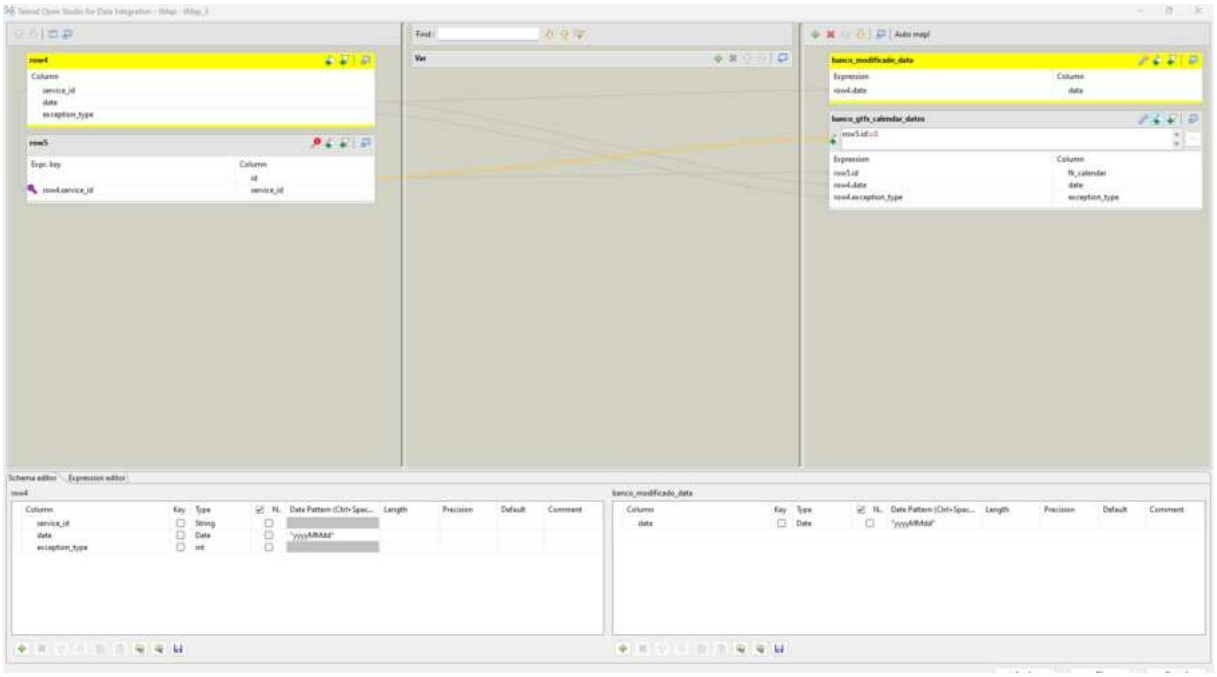
Figura 23 – *Subjob* Calendar Dates 2

Isso posto, como já dito anteriormente, o funcionamento de ambos os fluxos mencionados, assim como os demais no *job*, ocorre de forma similar. Assim, o fluxo médio de um *subjob* segue 3 etapas: (i) acesso as fontes; (ii) obtenção de chaves estrangeiras; e (iii) carga no banco.

No exemplo, a primeira etapa engloba os componentes nomeados `calendar_dates.txt` e `tb_calendar`. O `calendar_dates.txt` é do tipo `tFileInput_Delimited` e faz a leitura do arquivo `calendar_dates.txt` do modelo GTFS. Já `tb_calendar` é do tipo `tDBInput` e recebe o resultado da consulta que executa no banco. Os dados levantados convergem no `tMap_11` dando início à segunda etapa.

Essa segunda etapa, ocorre inteiramente dentro do `tMap` do *subjob* (Figura 24). Os fluxos são unidos no componente e as informações relevantes a cada destino separadas. Para a *row* `banco_gtfs_calendar_dates` são transferidos todos os campos do arquivo original com os identificadores trocados por aqueles usados pelo banco, de forma a agirem como chaves estrangeiras, fornecidos pela *row* `row5` alimentada pelo componente `tb_calendar_date`. Já a *row* `banco_modificado_data` contém apenas o campo da data em que a exceção marcada no arquivo `calendar_dates.txt`.

Figura 24 – Alterações do fluxo de dados feitas no “tMap_3”



Por último, são realizadas as operações de escritas nos bancos por meio dos componentes `tDBOutput`, `tb_calendar` e `tb_data`. No caso do `tb_calendar` cada linha passada pela *row* é garantidamente única, logo é simplesmente realizada a inserção. Em relação ao `tb_data`, é esperado que a repetição de valores ocorra com certa frequência, então a opção escolhida foi a de “atualizar ou inserir” e como chave escolhida o valor da data.

3.3.2 Bancos de Grafos

Se tratando do banco em RDF, o processo de ETL tem como principal foco a conversão dos dados do formato tabular que são disponibilizados para uma estrutura compatível com a Web Semântica, seguindo o padrão RDF e não na inserção deles no banco como aconteceu nos processos anteriores, dado que em um banco de triplas, como o GraphDB, o grafo de conhecimento pode ser gerado a partir da importação de um arquivo RDF (em diferentes formatos de serialização). Essa conversão ocorre em duas etapas: (i) escolha de uma ontologia para a representação dos dados e (ii) mapeamento dos atributos das tabelas para os termos da ontologia. O primeiro passo foi discutido na seção 3.2.2, enquanto o segundo é tópico desta. Os arquivos considerados foram os mesmos que no processo para os bancos relacionais, incluindo o arquivo criado `calendar_addon.txt`.

3.3.2.1 Criação das IRIs

A primeira escolha a ser feita foi definir o formato de criação de IRIs. Dado que os arquivos originais não seguem os princípios do RDF, é natural que os elementos não tenham IRIs já definidas nestes. Ao iniciar esse processo, surgiu a necessidade de decidir o que um elemento representado por uma IRI engloba. Assim, ficou definido que um elemento seria o representante de uma linha contida nos arquivos originais. Embora, houvesse outras alternativas para o escopo do elemento, as entidades descritas no Quadro 5, por exemplo, é possível perceber que a GTFS *Ontology* incentiva a abordagem escolhida. Visto que, observando a ontologia, é possível observar que os atributos correspondem a colunas do arquivo e, em sua maioria, as classes aos arquivos do padrão GTFS.

Quadro 5 – Relação entre as classes da GTFS Ontology e os arquivos GTFS

CLASSE	ARQUIVO
Agency	agency.txt
Stop	stops.txt
Route	routes.txt
Trip	trips.txt
StopTime	stop_times.txt
CalendarRule	calendar.txt/calendar_addon.txt
Service	calendar.txt/calendar_addon.txt
CalendarDateRule	calendar_dates.txt
FareClass	fare_attributes.txt
FareRule	fare_rules.txt
Shape	shapes.txt
ShapePoint	shapes.txt
Frequency	frequencies.txt

Em seguida, pode-se voltar à busca de um formato de criação de IRIs. Nela, havia três objetivos que o formato a ser encontrado deveria cumprir. Primeiro, ele deveria ser capaz de gerar IRIs únicas. Isso é obviamente derivado da definição de IRI e de seu papel como identificador de um elemento. Em segundo lugar, o formato deve possibilitar a geração automática. Essa problemática decorre do volume de registros e, no contexto da ferramenta utilizada, em conjunto com o objetivo anterior, implica que a IRI deve ser gerada, pelo menos em parte, a partir dos valores de alguns campos da linha que representa. Por último, o formato deve ser estável, isto é, para um elemento a IRI gerada deve ser sempre a mesma.

Após, a consideração dos objetivos, o formato geral definido foi o seguinte: "http://127.0.0.1:3333/NOME_DA_ENTIDADE/IDENTIFICADOR". Nele, "http://127.0.0.1:3333" é a IRI base escolhida a partir do domínio local onde o OpenRefine é hospedado[26]; "NOME_DA_ENTIDADE" se refere ao tipo de entidade que a IRI representa, geralmente, é o mesmo nome do arquivo de origem; "IDENTIFICADOR" se trata de um campo capaz de identificar unicamente aquele elemento. Dois fatores devem ser destacados sobre esse formato final. Em primeiro lugar, o escopo inicial pensado para o elemento representado pela IRI gerada teve que ser alterado. Embora a intuição original se sustente na maioria dos casos, quando observada a maneira que a ontologia representa os arquivos calendar.txt e calendar_dates.txt, a ideia de que cada IRI deva representar uma linha de cada arquivo não é verdade para todos os casos. Dessa forma, a partir do Quadro 5, é possível inferir quantos elementos são gerados a partir de uma linha de um arquivo de maneira fiel ao resultado final. Por último, há alguns arquivos cujo formato não pode ser aplicado em razão dos registros nesses arquivos não conterem um campo que possa assumir a função de identificador. Assim, foi necessário construir as IRIs a partir de combinação de campos que garantiriam unicidade. Esses casos são apontados no Quadro 6.

Quadro 6 – Formatos de IRIs excepcionais

FORMATO	ENTIDADE	ARQUIVO
http://127.0.0.1:3333/ServiceRule/service_id+date	CalendarDateRule	calendar_dates.txt
http://127.0.0.1:3333/ServiceRule/ service_id+start_date+end_date	CalendarRule	calendar.txt
http://127.0.0.1:3333/Period/start_date+end_data	Period	calendar.txt
http://127.0.0.1:3333/FareRule/fare_id/route_id	FareRule	fare_rules.txt
http://127.0.0.1:3333/Shape/shape_id/shape_pt_sequence	ShapePoint	shapes.txt
http://127.0.0.1:3333/StopTime/trip_id/stop_id	StopTime	stop_times.txt

3.3.2.2 Processo de Aplicação da Ontologia

O último passo no processo de mapeamento é atribuir aos campos dos arquivos elementos da ontologia, gerando o mapeamento. Como a ontologia foi pensada para a representação do modelo GTFS, a correlação entre os itens de ambos os lados do mapeamento é simples. Dessa forma, será trazido como exemplo o caso do mapeamento do arquivo

[26]Definida apenas para a realização dos testes deste trabalho, para a publicação desses dados na Web, deve-se haver a atribuição de IRIs válidas globalmente

`calendar.txt` e, por consequência do `calendar_addon.txt`, pois esses dois arquivos apresentam decisões e soluções presentes em todos os outros.

A abordagem de mapeamento começa pela criação e escolha da IRI que gerará um nó sem arestas de entrada no subgrafo gerado por esses arquivos como raiz. Geralmente, uma linha do arquivo contém informação sobre apenas um nó, logo a IRI desse nó é a escolhida, porém, no contexto do `calendar.txt` e `calendar_addon.txt`, existem 3 candidatos (Quadro 5). Contudo, observando os relacionamentos apresentados pela documentação, é possível ter uma noção da forma desse subgrafo (Figura 25). Com isso, é possível perceber que o primeiro passo é gerar as IRIs dos nós da classe `gtfs:Service`. Para isso, seleciona-se o campo base para gerar a IRI, o tipo de nó como IRI, o formato de geração da IRI (Figura 26 e Figura 27) de acordo com o Quadro 6 e atribui-se à classe correspondente.

Figura 25 – Subgrafo gerado pelos arquivos calendar.txt e calendar_addon.txt

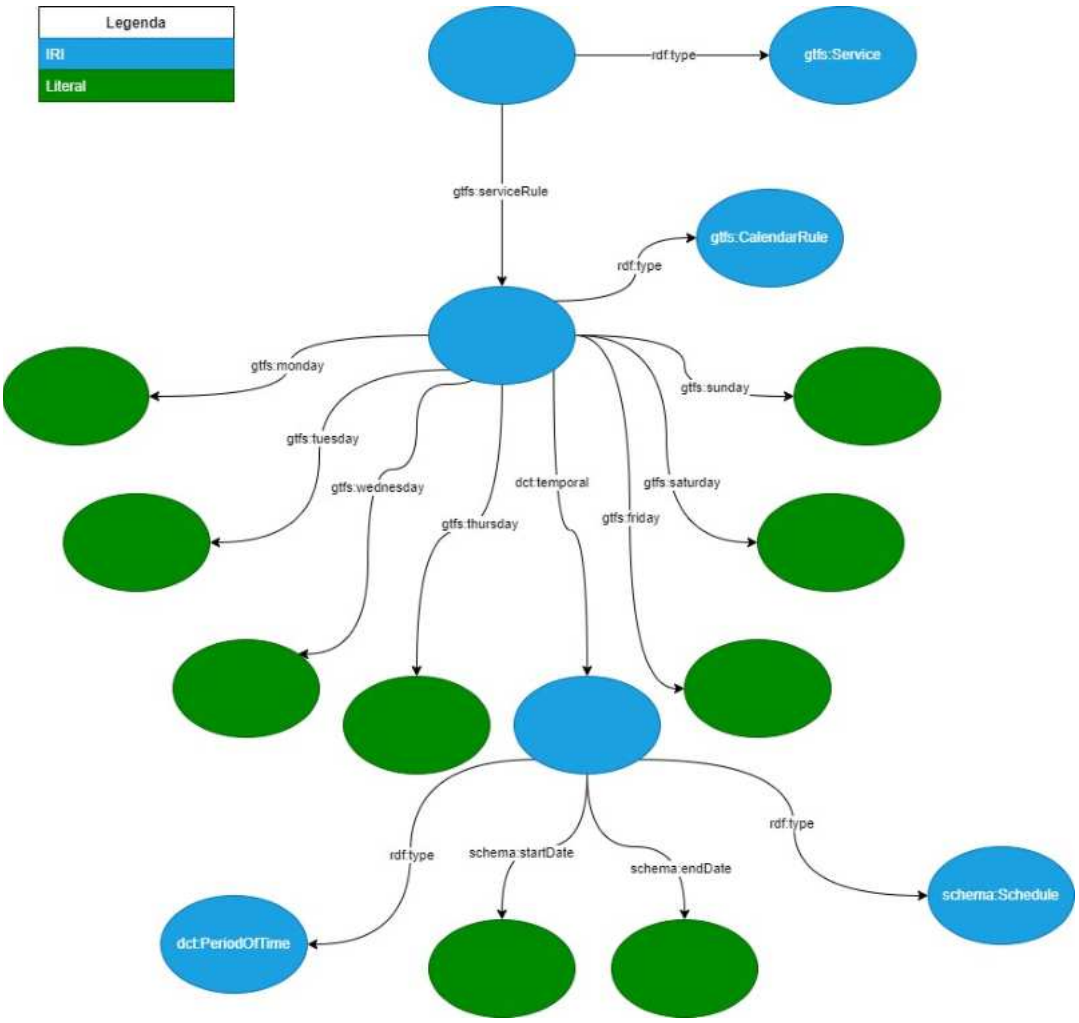


Figura 26 – Criando uma IRI pela interface RDF Node

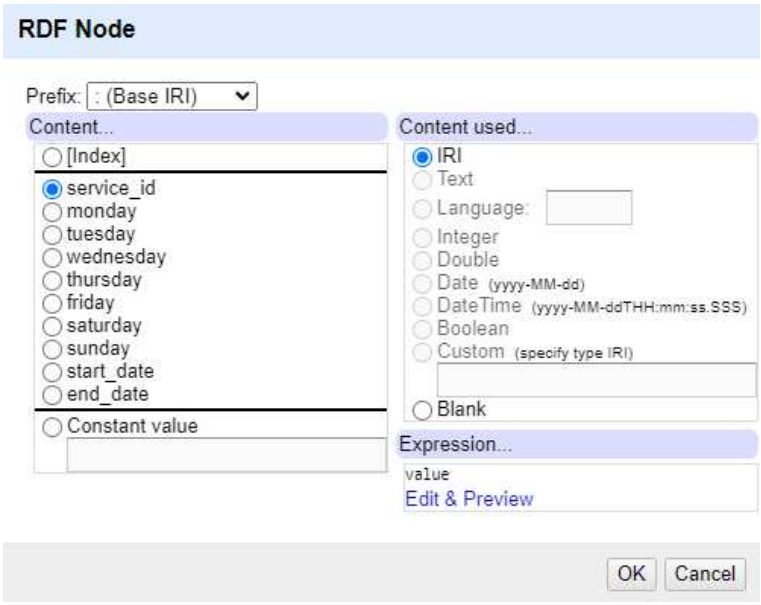
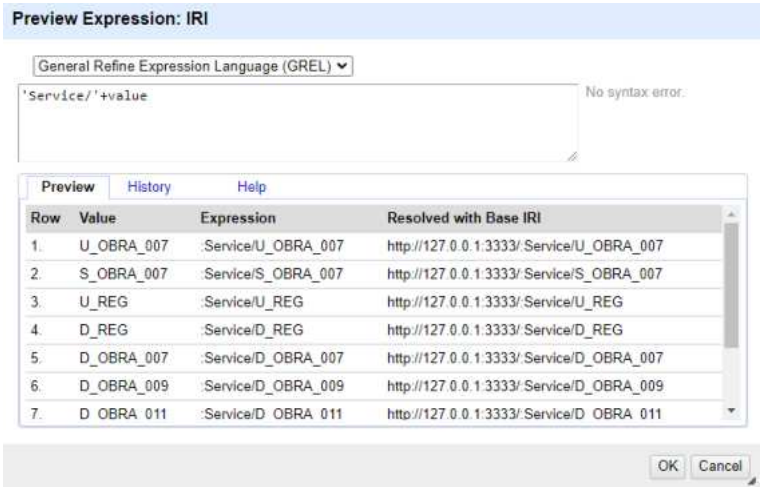


Figura 27 – Interface GREL

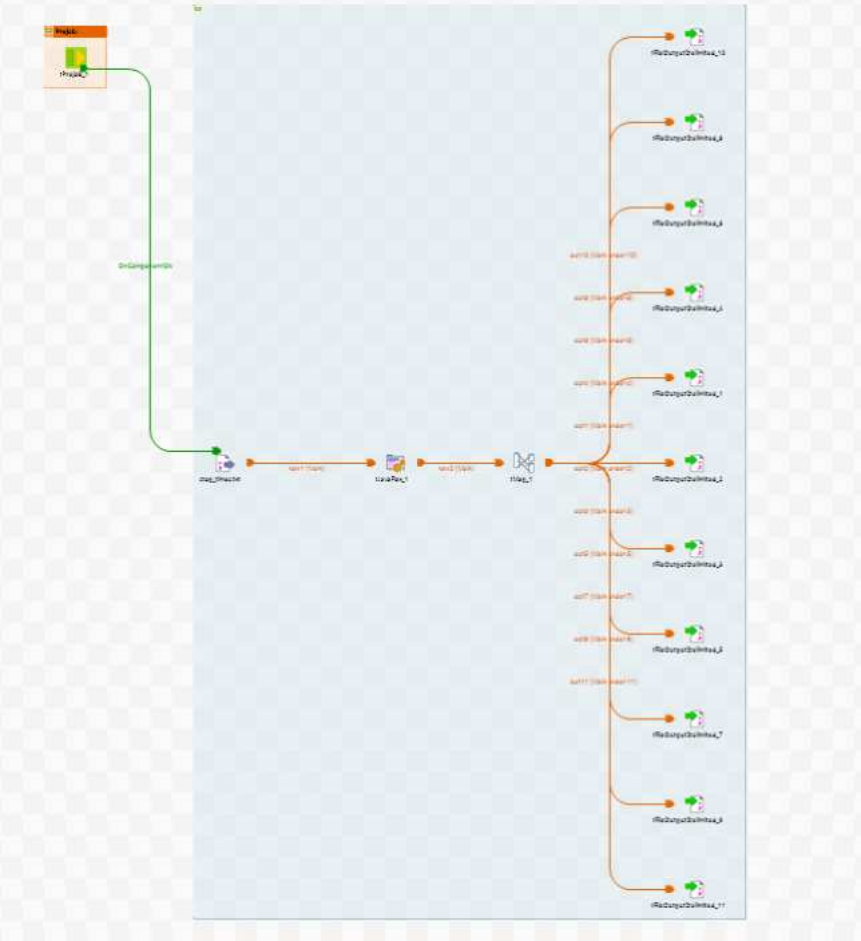


As próximas etapas do mapeamento ocorrem, de forma exploratória, seguindo os relacionamentos da ontologia, ou seja, são mapeadas as propriedades de um nó IRI, caso o objeto ligado pelo predicado também seja um nó IRI são mapeadas as suas propriedades. Ao fim, é realizada a exportação do arquivo gerado a partir do mapeamento aplicado.

Os mapeamentos para os outros arquivos seguiram processos semelhantes e, portanto, não serão descritos em detalhes. No entanto, uma ocorrência durante o mapeamento do arquivo `stop_times.txt` é digna de nota. Esse acontecimento não está relacionado ao processo de mapeamento em si, e sim durante a geração dos arquivos de saída. Em razão do tamanho do arquivo original, o OpenRefine utilizava toda memória alocada enquanto exportava o arquivo RDF, o que levava a uma falha crítica da aplicação. Com isso, foi necessário buscar

uma solução. A alternativa encontrada foi dividir o conteúdo do arquivo em 11 arquivos menores, cada um contendo 10.000 instâncias. A escolha do número 11 foi baseada no tamanho dos arquivos que conseguiam ser exportados pela ferramenta, objetivando um valor que gerasse arquivos com uma quantidade de bytes semelhante a deles. O método para a divisão do conteúdo do arquivo foi a criação do *job* `divide_stop_times` utilizando o Talend (Figura 28).

Figura 28 – Job "divide_stop_times"



O fluxo do *job* é simples: (i) extrai-se a informação do arquivo original utilizando o componente `tFileInput_Delimited`; (ii) o código presente no `tJavaFlex` atribui um índice para cada linha extraída do arquivo; (iii) no `tMap`, o índice atribuído na etapa anterior é usado para determinar qual `tFileOutput_Delimited` a linha será enviada; (iv) o `tFileOutput_Delimited` escreve ou sobrescreve (caso o arquivo já existe) o arquivo correspondente.

4 COMPARAÇÃO ENTRE OS BANCOS DE DADOS

Este capítulo foca na comparação de desempenho entre os bancos de dados discutidos anteriormente. Primeiro, são apresentados características do ambiente em que os teste foram realizados como o volume de dados em cada banco e as especificações da máquina em que os bancos foram implementados e as consultas de comparação realizadas. Também se descreve a metodologia por trás dos testes, incluindo as consultas escolhidas para comparação, a forma de se medir os tempos de execução, etc. Por fim, são discutidos os resultados encontrados, salientando o banco que obteve o maior desempenho e algumas consultas que se destacaram.

4.1 AMBIENTE DE TESTES

As comparações foram realizadas nos três bancos de dados introduzidos no capítulo anterior: os dois bancos relacionais (GTFS e Modificado) e o banco em RDF. As tabelas abaixo (Tabela 1, Tabela 2 e Tabela 3) indicam o volume de dados nos bancos GTFS, Modificado e RDF, respectivamente. Os bancos foram implementados e as consultas executadas em uma máquina com armazenamento em um SSD de 1TB, memória de 24 GB, processador Intel Core i5-10400F e de sistema operacional Windows 11. Como dito no capítulo anterior, para os bancos relacionais o SGBD escolhido foi o Postgresql e para o banco RDF foi usado o GraphDB. As respectivas versões utilizadas foram a 16.2 para o Postgresql e 10.6.1 para o GraphDB.

Tabela 1 – Volume de dados do banco “Banco GTFS” demonstrado em instâncias por tabela.

TABELA	QUANTIDADE DE INSTÂNCIAS
tb_agency	5
tb_calendar	28
tb_calendar_dates	476
tb_fare_attributes	60
tb_fare_rules	449
tb_frequencies	16.561
tb_routes	449
tb_shapes	354.750
tb_stops	7.406
tb_stop_times	1.063.667
tb_trips	18.556

Tabela 2 – Volume de dados do banco “Banco Modificado” demonstrado em instâncias por tabela.

TABELA	QUANTIDADE DE INSTÂNCIAS
tb_agencia	5
tb_data	121
tb_data_dia_servico	476
tb_dia_servico	28
tb_linha	449
tb_parada	7.406
tb_parada_viagem	1.063.667
tb_ponto_espaco	64.330
tb_ponto_espaco_trajeto	354.750
tb_tarifa	60
tb_trajeto	956
tb_viagem	18.556

Tabela 3 – Volume de dados do banco em RDF demonstrado em instâncias por classe.

CLASSE	QUANTIDADE DE INSTÂNCIAS
gtfs:Agency	5
gtfs:CalendarDateRule	476
gtfs:CalendarRule	28
gtfs:FareClass	12
gtfs:FareRule	449
dct:PeriodOfTime	6
gtfs:Frequency	16.561
gtfs:Route	449
schema:Schedule	6
gtfs:Service	28
gtfs:ServiceRule	504
gtfs:Shape	956
gtfs:ShapePoint	354.750
gtfs:Station	153
gtfs:Stop	7.406
gtfs:StopTime	1.062.865
gtfs:Trip	188.556

4.2 MÉTODO DE EXPERIMENTAÇÃO

A forma de avaliação para os três bancos se baseou no método apresentado em (Chaves-Fraga et al., 2024). O artigo descreve a aplicação de dezoito consultas em SPARQL pensadas com o objetivo de forçar o uso de certos elementos da linguagem (Agrupamentos, Ordenação, Padrões Opcionais etc.) para avaliar o desempenho de consultas no grafo de conhecimento formado pela GTFS *Ontology*. Essas consultas foram reescritas para a estrutura do banco RDF. As versões inalteradas das consultas podem ser vistas no Anexo A. Além disso, uma nova consulta foi pensada. O objetivo da consulta adicional é simular uma requisição, pertinente a aplicativos ou análises do âmbito de transporte público, para encontrar todas as paradas acessíveis a partir de um ponto. Assim, um total de 19 consultas, presentes no Quadro 7, foram replicadas de forma a se adaptarem aos bancos GTFS, Modificado e RDF. O código de cada consulta como executado nos bancos GTFS, Modificado e RDF estão presentes na íntegra, em SQL ou SPARQL, a depender do banco, nos Apêndices D, E e F, respectivamente. A maneira escolhida para verificar se as consultas replicadas em cada banco são equivalentes foi comparar a quantidade de instâncias retornadas e a estrutura do dado retornado.

Para a medição do tempo de execução de cada consulta, a forma variou de acordo com o SGBD. No caso dos bancos implementados no PostgreSQL, que é um SGBD relacional, é possível usar a combinação das cláusulas `EXPLAIN` e `ANALYZE` do SQL. Essas cláusulas, quando incluídas antes de uma consulta, levam o SGBD a exibir informações como o plano de execução (sequência de passos que o SGBD toma para alcançar o resultado da consulta), total de linhas retornadas e, mais importante para este trabalho, o tempo de execução da consulta. Já para o banco RDF, foram tentadas duas abordagens sendo apenas uma delas bem-sucedida. A primeira delas fez uso de uma funcionalidade exclusiva do GraphDB, a cláusula `onto:measure`. Similar ao caso do SQL, usando a `onto:measure`, é possível obter o tempo de execução da consulta. No entanto, a precisão da medição é insuficiente, sendo impossível medir consultas com duração de poucos milissegundos. Assim, se fez necessário encontrar uma forma de maior precisão para a mensurar o tempo. Como solução, utilizou-se a ferramenta de *log* do GraphDB. Ela registra cada ação tomada no banco, incluindo o início e término de consultas, como mostrado no destaque feito na Figura 29. Com isso, subtraindo os carimbos de data/hora, é possível calcular a duração da consulta. Dada a melhor precisão, essa foi a alternativa escolhida para mensurar o tempo de execução das consultas nesse banco.

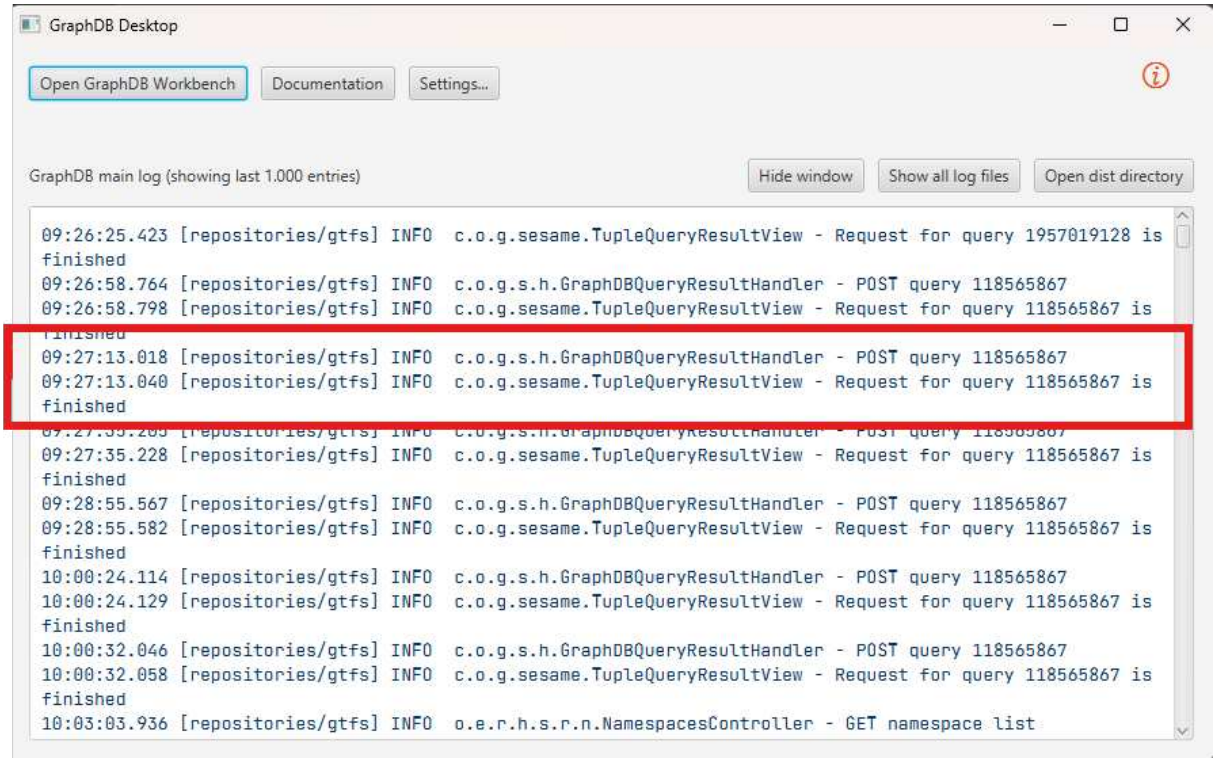
Quadro 7 – Lista de consultas

CONSULTA	DESCRIÇÃO
c1	Lista de todas as coordenadas e seus trajetos associados.
c2	Todas as paradas, e algumas de suas propriedades, que estejam localizadas a uma latitude maior que a média.
c3	Informação de acessibilidade de todas as estações, se disponível.
c4	Lista de todas as agências e suas rotas, com algumas de suas propriedades.
c5	Dias de serviço adicionados em uma data específica.
c6	Quantidade de rotas em uma agência específica.
c7	Lista de todas as paradas sem informação sobre acessibilidade para cadeira de rodas ao longo de uma linha específica.
c8	Lista de linhas com suas viagens, dias de serviço, paradas e tempo de paradas associados mais algumas propriedades adicionais.
c9	Viagens e os trajetos associados.
c10	Número de Viagens com duração maior que 30 minutos.
c11	Viagens que estão disponíveis em uma certa data.
c12	Número de paradas sem informação sobre acessibilidade para cadeira de rodas agrupado por linha.
c13	Todos os acessos de estações.
c14	Todos os horários de parada e suas relativas linhas e ordem de parada, ordenados pela ordem de parada.
c15	Qualquer propriedade contendo a sequência de caracteres especificada.
c16	Para todas as linhas, todas as adições no calendário durante um mês específico.
c17	Viagens com seus tempos de início e fim e linhas associadas.
c18	Todas as linhas com viagens no domingo.
c19	Todas as paradas acessíveis a partir de uma parada determinada.

Contudo, há problemas com o uso do *log* para estimar o tempo de execução de consultas que devem ser considerados ao se observar os resultados obtidos. Em primeiro lugar, a precisão da medição é inferior a obtida para os bancos relacionais, na ordem de 10^0 ms para o banco em grafos e 10^{-3} ms para os relacionais. Além disso, não há como medir o impacto do *cache* do GraphDB nas medições. Ao contrário das cláusulas `EXPLAIN` e `ANALYZE`, a ferramenta não tem como função primária a coleta de estatísticas sobre uma consulta. Dessa forma, não se garante que o SGBD vá executar a consulta por completo sem recorrer a dados já armazenados no *cache*.

Utilizando os métodos de coleta descritos anteriormente, cada consulta foi executada 5 vezes, seguindo o apresentado em (Chaves-Fraga et al., 2024). Os resultados de cada execução foram registrados e foi calculada a média aritmética destes para cada consulta.

Figura 29 – Interface de *log* do GraphDB



4.3 RESULTADOS

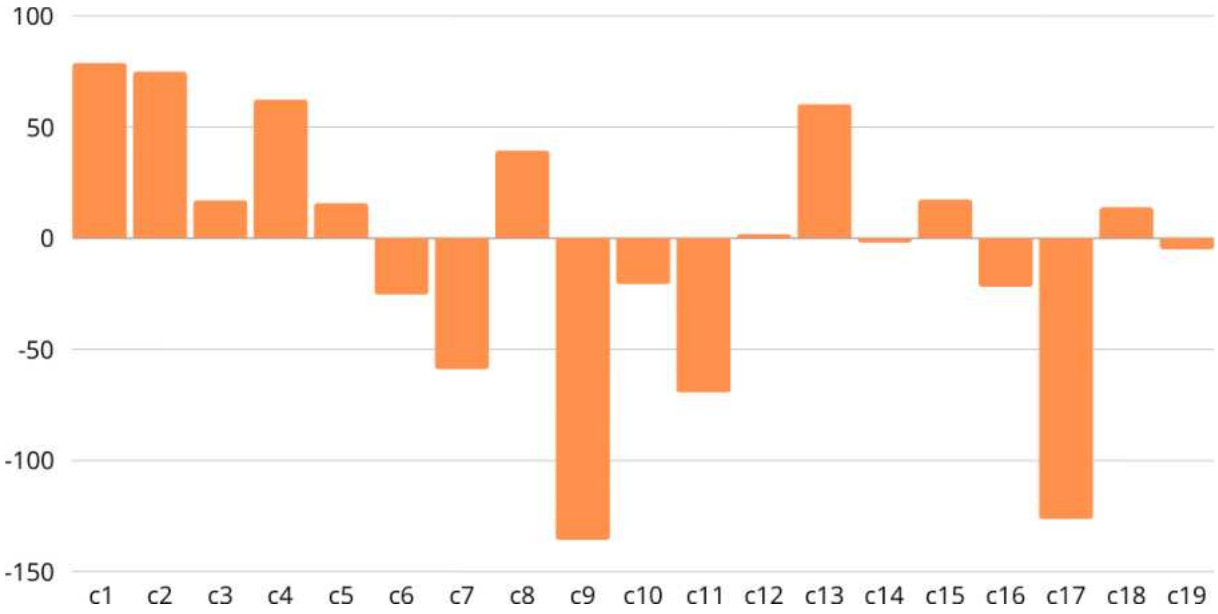
Os tempos médios calculados foram compilados, junto com o número de resultados retornados para cada consulta, na Tabela 4. O registro de cada teste individual está contido nos apêndices G, H e I para os bancos GTFS, Modificado e RDF, respectivamente.

Tabela 4 – Comparação da execução de consultas em cada banco apresentando tempo médio.

CONSULTA	TEMPO MÉDIO (ms)			NÚMERO DE RESULTADOS
	BANCO GTFS	BANCO MODIFICADO	BANCO RDF	
c1	36,7248	173,3006	1.260,6	354.750
c2	9,6734	38,808	522,8	3.931
c3	0,7270	0,875	9,6	153
c4	0,1584	0,4218	14,8	449
c5	0,1738	0,2066	18,6	7
c6	0,2350	0,1874	3,2	1
c7	152,6904	96,068	38.551,8	32
c8	2.278,0058	3.765,0102	10.986,2	1.062.865
c9	28,1504	11,9376	42,8	18.556
c10	570,6500	472,7082	6.960,4	1
c11	11,2948	6,6644	20,6	9.499
c12	5.260,4842	5.361,2104	7.481,6	338
c13	0,9800	2,4686	16,8	365
c14	661,3622	648,0526	19.072,4	1.063.667
c15	73,9232	89,588	355,4	2
c16	25,6576	21,0508	92,6	17.518
c17	11,0392	4,8758	127	16.561
c18	3,4584	4,0196	27,8	4.278
c19	150,1056	142,9924	259,8	58

4.3.1 Resultados da Comparação entre Bancos Relacionais

Figura 30 – Comparação entre os resultados dos bancos relacionais



Comparando os resultados obtidos pelos dois bancos relacionais (GTFS e Modificado), foi elaborado o gráfico da Figura 30, representando a porcentagem de melhora na execução de uma consulta no banco GTFS em relação ao Modificado. No gráfico, se observa que não há consenso em qual dos bancos tem os menores tempos médios de execução. Dessa forma, para se entender melhor o impacto da normalização nos bancos, serão discutidos os melhores resultados segundo a Figura 30 para o banco GTFS e para o banco Modificado. Essas consultas foram `c1` (Figura 31) e `c9` (Figura 32) respectivamente.

A consulta `c1` foi o caso em que a versão executada no banco GTFS (Figura 31a) teve a melhor desempenho quando comparada a versão executada no banco Modificado (Figura 31b). Observando as duas versões, a diferença evidente é o número de tabelas envolvidas. Fica claro que essa é a principal causa do melhor desempenho por parte do banco GTFS. Na linguagem SQL, sabe-se que a cláusula `JOIN` possui custo inerente a seu uso. Logo é natural que, no caso de duas consultas que diferem apenas no número de operações `JOIN`, sem variações em índices nas tabelas, a consulta com menor número seja a mais rápida. Sobre o motivo da consulta `c1`, em específico, apresentar a maior discrepância, em tempo médio de execução, favorável ao banco GTFS, os fatores mais prováveis são: (i) a consulta tem o terceiro maior retorno entre as executadas (Tabela 4); (ii) a consulta retorna a

maioria dos campos que originalmente pertenciam ao arquivo `shapes.txt`. No banco GTFS, o conteúdo desse arquivo está todo contido na tabela `tb_shapes`, como dito na seção 3.2.1.1, porém, no banco Modificado, esse é o arquivo mais dividido entre diferentes tabelas (Quadro 2). A soma do grande número de resultados com a maior discrepância em número de cláusulas `JOIN` acarretam no desempenho do banco GTFS superar o do banco Modificado, em uma margem maior do que em qualquer outra consulta.

Figura 31 – Consulta c1

```
SELECT
     shape_id
     ,shape_pt_lat
     ,shape_pt_lon
     ,shape_dist_traveled
FROM tb_shapes;
```

(a)

```
SELECT
     id_trajeto
     ,longitude
     ,latitude
     ,ordem_sequencia
FROM tb_ponto_espaco pt
JOIN tb_ponto_espaco_trajeto pet
   on pt.id = pet.fk_ponto_espaco
JOIN tb_trajeto t
   on t.id = pet.fk_trajeto
```

(b)

Já no caso da consulta `c9`, o banco Modificado apresenta a melhor desempenho quando comparado com o banco GTFS. Nesse caso, diferentemente do anterior, não há diferença no número de cláusulas `JOIN` entre a consulta executada no banco GTFS, `c9_gtfs` (Figura 32a), e a consulta realizada no banco Modificado, `c9_modificado` (Figura 32b). No entanto, o custo da operação difere entre as versões. Isso ocorre pois a quantidade de registros envolvidos na operação de `JOIN` é diferente em ambos os casos. Na consulta `c9_gtfs`, a cláusula `JOIN` está entre as tabelas `tb_trips`, que contém 18.556 registros (Tabela 1), e `tb_shapes`, composta de 354.750 registros (Tabela 1). Já em `c9_modificado`, a cláusula está entre as tabelas `tb_viagem` e `tb_trajeto`, com 18.556 e 956 registros respectivamente (Tabela 2). Assim, a versão `c9_modificado` tem a operação `JOIN` de menor custo. A diminuição do custo é consequência dos esforços de normalização aplicados na criação do banco Modificado. Na tabela `tb_shapes`, assim como no arquivo `shapes.txt`, cada registro representa as coordenadas de um ponto[27], trazendo seu trajeto associado. Dessa forma, como cada trajeto é composto de múltiplos pontos, um mesmo trajeto aparece em diferentes registros dessa tabela. Com isso, ao separar os trajetos das coordenadas como é feito no banco Modificado, a normalização favorece o desempenho desse banco ao contrário do caso da consulta `c1`.

[27]Observado no código de criação da tabela no Apêndice A e na referência do padrão.

Figura 32 – Consulta c9

```
SELECT DISTINCT
     trip_id
     ,shape_id
FROM tb_trips t
JOIN tb_shapes s
   on t.fk_shapes = s.id;
```

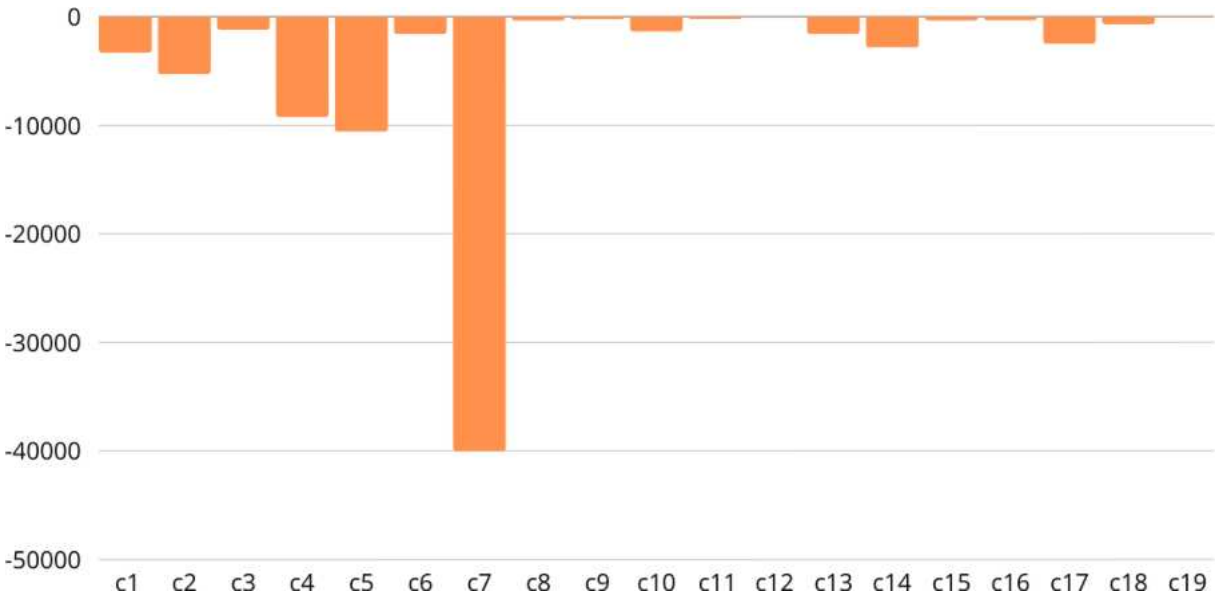
(a)

```
SELECT DISTINCT
     id_viagem
     ,id_trajeto
FROM tb_viagem v
JOIN tb_trajeto t
   on v.fk_trajeto = t.id
```

(b)

4.3.2 Comparação entre os Bancos Relacionais e o Banco de Grafos

Figura 33 – Comparação entre os resultados dos bancos relacionais e o RDF



Ao observar os resultados obtidos em cada banco de dados (Tabela 4), construiu-se um gráfico similar ao da Figura 30 (apresentado na Figura 33). O gráfico foi elaborado comparando o tempo médio de execução das consultas no banco RDF com o menor tempo médio entre as execuções nos bancos GTFS e Modificado. É visível que os bancos relacionais obtiveram melhor desempenho na execução de todas as consultas. Assim, buscando explicar o motivo da soberania do modelo relacional nas comparações realizadas, serão discutidas, de forma análoga à seção anterior, a consulta de melhor desempenho do banco RDF em comparação aos relacionais e a de pior desempenho: `c12` (Figura 32) e `c7` (Figura 34) respectivamente. A avaliação foi feita entre o tempo médio da consulta no banco RDF e o menor tempo médio de um banco relacional (GTFS ou Modificado).

A consulta `c12` se destaca como a consulta em que a variante no banco RDF, `c12_RDF` (Figura 34b), mais se aproxima aos resultados nos bancos relacionais,

`c12_GTFS`[28] (Figura 34a), no tempo médio de execução. O motivo mais provável para isso é a quantidade de cláusulas `JOIN` em `c12_GTFS`. A capacidade de recuperar informações do banco sem a realização de operações `JOIN` explícitas é uma qualidade conhecida dos bancos de dados em grafos (NEO4J, 2024). Dessa forma, mesmo que o custo de recuperação de informação seja maior no banco RDF, a medida que o número de operações `JOIN` aumenta, como é o caso de `c12_GTFS`, o modelo de grafos e o relacional parecem se equiparar. Isso sugere que, para consultas que realizam os mesmos tipos de operações, se a consulta relacional tem uma grande quantidade de operações `JOIN`, a consulta para o banco de grafos terá um melhor desempenho.

Já no caso da consulta `c7`, tem-se o caso oposto ao anterior. A versão da consulta no banco RDF, `c7_RDF` (Figura 34b), tem complexidade equiparável a consulta `c12_RDF`. Contudo, sua versão relacional, `c7_modificado`[29] (Figura 34a), é bem mais simples em comparação à `c12_GTFS`. Assim, o custo maior de recuperação da informação em um banco de grafos, mencionado anteriormente, tem um maior impacto na diferença do tempo médio de execução.

[28]Nesse caso a variante comparada é a do banco GTFS, pois é com melhor tempo médio de execução.

[29]Nesse caso, a variante comparada é a do banco Modificado, pois apresenta melhor tempo médio de execução.

Figura 34 – Consulta c12

```
SELECT
  route_long_name
  ,count(stop_name)
FROM(
     SELECT
       r.route_long_name
       ,s.stop_name
       FROM tb_routes  r
       LEFT JOIN tb_trips t
          on r.id =t.fk_routes
      LEFT JOIN tb_stop_times st
         on st.fk_trips = t.id
      LEFT JOIN tb_stops s
         on s.id= st.fk_stops
      WHERE
        (wheelchair_boarding is null
        OR wheelchair_boarding = 0)
      AND s.fk_stops is null
      UNION
      SELECT
        r.route_long_name
        ,s.stop_name
      FROM tb_routes  r
      LEFT JOIN tb_trips t
         on r.id =t.fk_routes
      LEFT JOIN tb_stop_times st
         on st.fk_trips = t.id
      LEFT JOIN tb_stops s
         on s.id= st.fk_stops
      JOIN tb_stops s2
        on s.fk_stops  = s2.id
      WHERE
       (s.wheelchair_boarding =0
       OR s.wheelchair_boarding is null)
       AND (s2.wheelchair_boarding = 0
           OR s2.wheelchair_boarding is
null)
      )
GROUP BY route_long_name
```

(a)

```
SELECT
  ?longName
  (count(distinct ?name) as ?
count)
WHERE {
  {
   ?route a gtfs:Route .
   ?route gtfs:longName ?longName
.
   ?trip a gtfs:Trip .
   ?trip gtfs:route ?route .
   ?stopTime a gtfs:StopTime .
   ?stopTime gtfs:trip ?trip .
   ?stopTime gtfs:stop ?stop .
   ?stop a gtfs:Stop.
   ?stop foaf:name ?name.
   ?stop
gtfs:wheelchairAccessible
gtfs:NoWheelchairAccessInformatio
nAvaiable . }
UNION
  {
   ?route a gtfs:Route .
   ?route gtfs:longName ?longName
.
   ?trip a gtfs:Trip .
   ?trip gtfs:route ?route .
   ?stopTime a gtfs:StopTime .
   ?stopTime gtfs:trip ?trip .
   ?stopTime gtfs:stop ?stop .
   ?stop a gtfs:Stop .
   ?stop foaf:name ?name.
   ?stop ^gtfs:parentStation ?
parentStation.
   ?parentStation foaf:name ?
parentStationName.
   ?stop
gtfs:wheelchairAccessible
gtfs:CheckParentStation.
   ?parentStation
gtfs:wheelchairAccessible
gtfs:NoWheelchairAccessInformatio
nAvaiable  }
} GROUP BY ?longName
```

(b)

Figura 34 – Consulta c7

```
SELECT DISTINCT
     l.nome_curto
     ,l.descricao
     ,v.nome_curto
     ,p.descricao
    ,latitude
    ,longitude
FROM tb_linha l
JOIN tb_viagem v
   on v.fk_linha = l.id
JOIN tb_parada_viagem pv
   on pv.fk_viagem = v.id
JOIN tb_parada p
   on pv.fk_parada = p.id
JOIN tb_ponto_espaco pt
   on p.fk_ponto_espaco = pt.id
WHERE
    l.nome_curto  = '913'
```

(a)

```
SELECT DISTINCT
     ?routeShortName
     ?routeDescription
     ?tripShortName
     ?stopDescription
     ?stopLat
     ?stopLong
WHERE
  {
   ?route a gtfs:Route .
   OPTIONAL { ?route gtfs:shortName ?
routeShortName . }
   OPTIONAL { ?route dct:description ?
routeDescription . }
   ?trip a gtfs:Trip .
   OPTIONAL { ?trip gtfs:shortName ?
tripShortName . }
   ?trip gtfs:service ?service .
   ?trip gtfs:route ?route .
   ?stopTime a gtfs:StopTime .
   ?stopTime gtfs:trip ?trip .
   ?stopTime gtfs:stop ?stop .
   ?stop a gtfs:Stop .
   OPTIONAL { ?stop dct:description ?
stopDescription . }
   OPTIONAL { ?stop geo:lat ?stopLat ;
geo:long ?stopLong . }
  {
   {?stop gtfs:wheelchairAccessible
gtfs:NoWheelchairAccessInformationAvaiable
.}
  UNION
   {?stop gtfs:wheelchairAccessible
gtfs:CheckParentStation.
   ?stop ^gtfs:parentStation ?
parentStation.
   ?parentStation
gtfs:wheelchairAccessible
gtfs:NoWheelchairAccessInformationAvaiable
. }}
   FILTER (?routeShortName='913')
}
```

(b)

4.4 CONSIDERAÇÕES FINAIS

Dadas as análises da seção anterior e os resultados apresentados na Tabela 4, os testes executados revelaram três principais características no armazenamento de dados do padrão GTFS: (i) o custo para recuperação de informação em consultas simples é maior no banco em RDF; (ii) no modelo relacional, não há grandes diferenças em desempenho entre uma modelagem mais ou menos normalizada, variando a depender da consulta; (iii) é possível que

consultas, cuja versão em um banco relacional contenha um grande número de operações `JOIN`, tenham um melhor desempenho quando executadas em um banco de grafos.

5 CONCLUSÃO

Este trabalho buscou entender o impacto no desempenho causado pelo uso de diferentes tipos de bancos de dados no armazenamento de dados do transporte público. Para isso, o desempenho foi medido sob a ótica do tempo de execução de consultas, sem considerar os objetivos de uma possível aplicação em que essas consultas estariam inseridas. A comparação envolveu uma implementação de banco de dados em grafos RDF no armazenamento de dados de transporte público e implementações mais tradicionais em bancos relacionais.

Além da comparação, o texto apresentou todas as etapas anteriores necessárias para a execução dos testes. Inicialmente, passou-se pela escolha do conjunto de dados a alimentar todos os bancos, trazendo a opções consideradas e o motivo para a escolha final. Em seguida, foram descritas as estruturas de cada um dos bancos de dados, suas tabelas e como se relacionam com os dados da fonte. Então, é detalhado o processo de ETL responsável pela transformação dos dados de forma a se adequar aos diferentes bancos e sua inserção neles, apresentando o passo a passo feito em cada ferramenta que foi apresentada.

A última parte do trabalho focou na condução da comparação. Foi trazido o volume de dados de cada banco no momento em que o teste foi realizado, bem como as características da máquina onde foram estabelecidos. Por fim, foram apresentados os resultados, destacando e comentando as consultas com melhor e pior desempenho segundo a métrica considerada.

Pode-se citar três principais contribuições deste trabalho. Em primeiro lugar, foi apresentada uma versão do padrão GTFS mais normalizada, utilizada no banco modificado. Além disso, também se contribuiu com versões triplificadas dos arquivos que compões o GTFS da cidade do Rio de Janeiro. Não só isso, também foram gerados os esquemas de mapeamento, isto é, por meio deles é possível gerar, para todo conjunto de dados seguindo o padrão GTFS, uma versão triplificada do arquivo correspondente usando o OpenRefine. Isto pode facilitar o esforço a ser despendido por aqueles que desejarem desenvolver futuramente pesquisas no campo da Web Semântica envolvendo o padrão GTFS. Tanto os arquivos triplificados quanto os esquemas de mapeamento podem ser encontrados em um repositório no Google Drive[30]. Por fim, outra contribuição a ser destacada são os resultados coletados a partir das comparações entre bancos, que podem ser utilizados para guiar trabalhos futuros.

[30] https://drive.google.com/drive/folders/1VUWKqcp5EjwWqdYQt1e3oeM35M0rkOof?usp=sharing

Os resultados obtidos por esse trabalho tratam sobre um conjunto específico de condições em que os testes foram realizados. Assim, desenvolvimentos futuros, em busca de uma visão mais geral, podem ser focados em expandir qualquer uma delas. É de interesse, por exemplo, expandir as comparações para abranger outros SGBDs tanto relacionais quanto como *triplestores*, com o intuito entender se o impacto dessa escolha é considerável no resultado final. Outra direção possível nas alterações das condições iniciais dos testes é a experimentação com diferentes ontologias. Embora o uso da L*inked* GTFS O*ntology* traga a vantagem de compartilhar o modelo conceitual por trás da fonte de dados, facilitando o processo de mapeamento e triplificação, é possível que outras ontologias sejam mais compatíveis com o tipo de consulta a ser executada.

Além de expansões nas etapas preparatórias, trabalhos futuros podem explorar métricas diferentes de comparação além do tempo de execução de consultas. O modelo de grafos RDF tem como características mais marcantes as interconexões com diferentes conjuntos de dados e a adição de valor semântico. Assim, explorar experimentos que avaliem o efeito dessas qualidades em comparação ao modelo relacional, que não tem elas como foco, seria muito relevante.

Também há espaço para iniciavas futuras que, utilizando como base as conclusões atingidas durante as análises de consultas presentes neste trabalho, explorem o armazenamento de dados usando mais de um tipo de banco de dados. Seguindo o exemplo apresentado por Chaves et al. (2020), pode-se avaliar a compatibilidade tanto da execução distribuída de consultas em bancos dos dois tipos (relacionais e de grafos) quanto aplicações relacionais que simulem ou sejam integradas a grafos RDF.

Em suma, os resultados obtidos favorecem o modelo relacional, mostrando que esse possui, em geral, um tempo médio de execução menor que o de grafos em RDF. No entanto, ainda há necessidade de maiores testes para entender se não há fatores capazes de permitir que o contrário aconteça.

REFERÊNCIAS BIBLIOGRÁFICAS

ANGLES, Renzo. A Comparison of Current Graph Database Models. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING WORKSHOPS, 28., 2012, Arlington, VA, USA. **A Comparison of Current Graph Database Models** [...]. [s.l.]: IEEE, 2012. p. 171-177, Disponível em: https://ieeexplore.ieee.org/abstract/document/6313676. Acesso em: 7 ago. 2024.

ANGLES, Renzo; THAKKAR, Harsh; TOMASZUK, Dominic. Mapping RDF Databases to Property Graph Databases. IEEE Access, v. 8, p. 86091–86110, 2020. Diponível em: https:// ieeexplore.ieee.org/abstract/document/9088985. Acesso em 31 ago. 2024.

BENERS-LEE, Tim; HENDLER, James; LASSILA, Ora. *The Semantic Web*. ***Scientific American*****,** v. 284, n. 2, p. 34-43, maio de 2001. Disponível em: https://www.jstor.org/stable/26059207. Acesso em: 06 dez. 2023.

BRAGA, M., SANTOS, M. Y., MOREIRA, A., Integrating Public Transportation Data: Creation and Editing of GTFS Data. **Advances in Intelligent Syste and Computing**, Cham, Springer International Publishing, 2014. p. 53–62. Disponível em: http://dx.doi.org/10.1007/978-3-319-05948-8_6. Acesso em: 19 abr. 2024.

BRASIL. Controladoria-Geral da União (CGU). **Portal Brasileiro de Dados Abertos**. 2019. Disponível em: https://www.gov.br/governodigital/pt-br/dados-abertos/portalbrasileirodadosabertos.pdf. Acesso em: 24 jan. 2024

CHAVES-FRAGA, David; PRIYATNA Freddy; CIMMINO Andrea; TOLEDO Jhon; RUCKHAUS Edna; CORCHO Oscar. GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport domain. **Journal of Web Semantics**, v. 65, art. 100596, dez. 2020. Disponível em: https://doi.org/10.1016/j.websem.2020.100596. Acesso em: 02 maio 2024.

CUNHA, Maria Alexandra; PRZEYBILOVICZ, Erico; MACAYA, Javiera Fernanda Medina, BURGOS, Fernando. . **Smart cities**: transformação digital de cidades. São Paulo. 2016. color. ISBN: 978-85-87426-29-1. Disponível em: https://ceapg.fgv.br/sites/ceapg.fgv.br/files/u60/ebook_smart_cities.pdf. Acesso em: 7 ago. 2024.

DINIZ, Gabriel Ribeiro; DA CUNHA, Felipe Domingos; LOUREIRO, Antonio Alfredo Ferreira. Análise da Mobilidade em Redes Veiculares Usando Diferentes Tipos de Grafos. In: WORKSHOP DE COMPUTAÇÃO URBANA (COURB), 4., 2020, Rio de Janeiro. **Anais** […]. Porto Alegre: Sociedade Brasileira de Computação, 2020. p. 248-261. ISSN 2595-2706. DOI: https://doi.org/10.5753/courb.2020.12367.

FORTIN, Philippe; MORENCY Catherine; TRÉPANIER Martin. Innovative GTFS Data Application for Transit Network Analysis Using a Graph-Oriented Method. **Journal of Public Transportation**, v. 19, n. 4, p. 18-37, dez. 2016. Disponível em: https://doi.org/10.5038/2375-0901.19.4.2. Acesso em: 02 maio 2024.

GRIMM, Stephan; ABECKER Andreas; VÖLKER, Johanna; STUDER, Rudi. *Ontologies and the Semantic Web*. *In:* DOMINGUE, John; FENSEL Dieter; HENDLER, James A. (org). **Handbook of Semantic Web Technologies**. 1. ed. Berlin: Springer Berlin, Heidelberg, 2011. p. 509–564. Acesso em: 6 dez. 2023.

GUARINO, Nicola; ORBELE, Daniel; STAAB Steffen. What Is an Ontology?. *In*: STAAB Steffen; STUDER Rudi. **Handbook on Ontologies**. 2. ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Cap. 1, p 1-17. Disponível em: https://iaoa.org/isc2012/docs/Guarino2009_What_is_an_Ontology.pdf. Acesso em 05 ago. 2024.

GUBERT, Fernanda Regina; SANTIN, Priscila Louise Leyser; MUNARETTO, Anelise; FONSECA, Mauro; SILVA, Thiago Henrique. Análise Espaço-temporal da Mobilidade Urbana de Diferentes Grupos Socioeconômicos: Uma Abordagem Multicamadas. In: WORKSHOP DE COMPUTAÇÃO URBANA (COURB), 4., 2020, Rio de Janeiro. **Anais** […]. Porto Alegre: Sociedade Brasileira de Computação, 2020. p. 152-165. ISSN 2595-2706. DOI: https://doi.org/10.5753/courb.2020.12360.

HEUSER, Carlos Alberto. Projeto de banco de dados. [s.l.] Porto Alegre: Sagra Luzzatto, 1998. Disponível em: https://www.cin.ufpe.br/~jrsl/Books/Projeto%20de%20Banco%20de%20Dados%20-%20C.%20A.%20Heuser.pdf. Aceso em: 14 jun. 2024.

KLYNE, Graham; CARROLL, Jeremy J.; MCBRIDE, Brian. **RDF 1.1 *Concepts and Abstract Syntax***. 2014. Disponível em: https://www.w3.org/TR/rdf-concepts/. Acesso em: 06 dez 2023.

LOPES, Dario Rais; MARTORELLI, Martha; VIEIRA, Aguiar Gonzaga. Conceito de Mobilidade Urbana Sustentável. In _____. **Mobilidade Urbana: Conceito e Planejamento no Ambiente Brasileiro**. 1. ed. Curitiba, PR: Appris, 2020.

MONTEIRO, Jéssica; SÁ, Filipe; BERNARDINO, Jorge. Experimental Evaluation of Graph Databases: JanusGraph, Nebula Graph, Neo4j, and TigerGraph. **Applied Sciences**, v. 13, n. 9, p. 5770, 1 jan. 2023. Disponível em: https://www.researchgate.net/publication/370613960_Experimental_Evaluation_of_Graph_Databases_JanusGraph_Nebula_Graph_Neo4j_and_TigerGraph. Acesso em: 10 jun. 2024.

NEO4J. **What is a graph database? - Getting Started**. Disponível em: https://neo4j.com/docs/getting-started/get-started-with-neo4j/graph-database/. Acesso em 06 ago. 2024.

ROSE, Karen; ELDRIDGE, Scott; CHAPIN, Lyman. What is the Internet of Things?. In: _. **The Internet of Things**: An Overview: Understanding the Issues and Challenges of a More Connected World. [s.l.]: Internet Society, 2015. p. 7-19. Disponível em: https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf. Acesso em 07 ago. 2024.

VICKNAIR, Chad; MACIAS, Michael; ZHAO, Zhendong; NAN, Xiaofei; CHEN, Yixin; WILKINS Dawn. A comparison of a graph database and a relational database. 15 abr. 2010. **Anais** […] New York, NY, USA, ACM, 15 abr. 2010. Disponível em: http://dx.doi.org/10.1145/1900008.1900067. Acesso em: 11 dez. 2023.

GLOSSÁRIO

Dia de serviço	Tipo de dia de operação.
Linha	Conjunto de percursos.
Trajeto	Uma série de pontos que formam o polígono que descreve uma viagem.
Viagem	Um percurso pertencente a uma linha

APÊNDICES

APÊNDICE A – CÓDIGO PARA CRIAÇÃO DO BANCO GTFS

```
-- Criação da tabela tb_agency
CREATE TABLE tb_agency(
    id INT GENERATED ALWAYS AS IDENTITY,
    agency_id VARCHAR NOT NULL,
    agency_name VARCHAR NOT NULL,
    agency_url VARCHAR NOT NULL,
    agency_timezone VARCHAR,
    agency_lang VARCHAR,
    PRIMARY KEY(id)
);

-- Criação da tabela tb_stops
CREATE TABLE tb_stops(
    id INT GENERATED ALWAYS AS IDENTITY,
    stop_id VARCHAR NOT NULL,
    stop_code VARCHAR,
    stop_name VARCHAR NOT NULL,
    stop_desc VARCHAR,
    stop_lat VARCHAR NOT NULL,
    stop_lon VARCHAR NOT NULL,
    zone_id VARCHAR,
    stop_url VARCHAR,
    location_type INT2,
    fk_stops INT,
    stop_timezone VARCHAR,
    wheelchair_boarding INT2,
    platform_code VARCHAR,
    PRIMARY KEY(id),
    CONSTRAINT fk_stops_stops FOREIGN KEY(fk_stops) REFERENCES tb_stops(id)
);

-- Criação da tabela tb_calendar
CREATE TABLE tb_calendar(
    id INT GENERATED ALWAYS AS IDENTITY,
    service_id VARCHAR NOT NULL,
    monday BOOLEAN NOT NULL,
    tuesday BOOLEAN NOT NULL,
    wednesday BOOLEAN NOT NULL,
    thursday BOOLEAN NOT NULL,
    friday BOOLEAN NOT NULL,
    saturday BOOLEAN NOT NULL,
    sunday BOOLEAN NOT NULL,
    start_date DATE,
    end_date DATE,
    PRIMARY KEY(id)
);

-- Criação da tabela tb_calendar_dates
CREATE TABLE tb_calendar_dates(
    id INT GENERATED ALWAYS AS IDENTITY,
    fk_calendar INT NOT NULL,
    date Date NOT NULL,
    exception_type INT2,
    PRIMARY KEY(id),
    CONSTRAINT fk_calendar_dates_calendar FOREIGN KEY(fk_calendar)
REFERENCES tb_calendar(id)
```

```
);

-- Criação da tabela tb_routes
CREATE TABLE tb_routes(
    id INT GENERATED ALWAYS AS IDENTITY,
    route_id VARCHAR NOT NULL,
    fk_agency INT NOT NULL,
    route_short_name VARCHAR NOT NULL,
    route_long_name VARCHAR NOT NULL,
    route_desc VARCHAR,
    route_type INT2 NOT NULL,
    route_color VARCHAR,
    route_text_color VARCHAR,
    PRIMARY KEY(id)
);

-- Criação da tabela tb_shapes
CREATE TABLE tb_shapes(
    id INT GENERATED ALWAYS AS IDENTITY,
    shape_id VARCHAR NOT NULL,
    shape_pt_sequence INT NOT NULL,
    shape_pt_lat VARCHAR NOT NULL,
    shape_pt_lon VARCHAR NOT NULL,
    shape_dist_traveled FLOAT,
    PRIMARY KEY(id)
);

-- Criação da tabela tb_trips
CREATE TABLE tb_trips(
    id INT GENERATED ALWAYS AS IDENTITY,
    trip_id VARCHAR NOT NULL,
    fk_routes INT NOT NULL,
    fk_calendar INT NOT NULL,
    trip_headsign VARCHAR ,
    trip_short_name VARCHAR,
    direction_id BOOLEAN ,
    fk_shapes INT NOT NULL,
    PRIMARY KEY(id),
    CONSTRAINT fk_trips_routes FOREIGN KEY(fk_routes) REFERENCES
tb_routes(id),
    CONSTRAINT fk_trips_calendar FOREIGN KEY(fk_calendar) REFERENCES
tb_calendar(id),
    CONSTRAINT fk_trips_shapes FOREIGN KEY(fk_shapes) REFERENCES
tb_shapes(id)
);

-- Criação da tabela tb_fare_attributes
CREATE TABLE tb_fare_attributes(
    id INT GENERATED ALWAYS AS IDENTITY,
    fare_id VARCHAR NOT NULL,
    price FLOAT4 NOT NULL,
    currency_type VARCHAR NOT NULL,
    payment_method INT2 NOT NULL,
    transfers INT2,
    fk_agency INT NOT NULL,
    transfer_duration int4,
    PRIMARY KEY(id),
    CONSTRAINT fk_fare_attributes_agency FOREIGN KEY(fk_agency) REFERENCES
tb_agency(id)
);
```

```
-- Criação da tabela tb_fare_rules
CREATE TABLE tb_fare_rules(
    id INT GENERATED ALWAYS AS IDENTITY,
    fk_fare_attributes INT NOT NULL,
    fk_routes INT,
    PRIMARY KEY(id),
    CONSTRAINT fk_fare_rules_fare_attributes FOREIGN
KEY(fk_fare_attributes) REFERENCES tb_fare_attributes(id),
    CONSTRAINT fk_fare_rules_routes FOREIGN KEY(fk_routes) REFERENCES
tb_routes(id)
);

-- Criação da tabela tb_stop_times
CREATE TABLE tb_stop_times(
    id INT GENERATED ALWAYS AS IDENTITY,
    fk_trips INT NOT NULL,
    stop_sequence INT NOT NULL,
    fk_stops INT NOT NULL,
    arrival_time  VARCHAR,
    departure_time VARCHAR,
    stop_headsign VARCHAR,
    shape_dist_traveled FLOAT4,
    time_point BOOLEAN,
    PRIMARY KEY(id),
    CONSTRAINT fk_stop_times_trips FOREIGN KEY(fk_trips) REFERENCES
tb_trips(id),
    CONSTRAINT fk_stop_times_stops FOREIGN KEY(fk_stops) REFERENCES
tb_stops(id)
);

-- Criação da tabela tb_frequencies
CREATE TABLE tb_frequencies(
    id INT GENERATED ALWAYS AS IDENTITY,
    fk_trips INT NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,
    headway_secs INT NOT NULL,
    exact_times BOOLEAN,
    PRIMARY KEY(id)
);


-- Criação da função utilizada na consulta c15
create or replace  function select_tb_stops(campo varchar)
returns  table(stop_id VARCHAR,atributo  varchar,valor varchar) as $$
declare consulta varchar;
begin
      consulta :='SELECT stop_id,'||quote_literal(campo)||'::varchar,'||
campo||'::varchar FROM tb_stops';
      return QUERY execute consulta;
end$$LANGUAGE plpgsql;
```

APÊNDICE B – CÓDIGO PARA CRIAÇÃO DO BANCO MODIFICADO

```
--Criação da tabela tb_agencia
CREATE TABLE tb_agencia(
    id INT GENERATED ALWAYS AS IDENTITY,
    id_agencia VARCHAR NOT NULL,
    nome VARCHAR NOT NULL,
    url VARCHAR NOT NULL,
    fuso_horario VARCHAR NOT NULL,
    lingua VARCHAR,
    PRIMARY KEY(id)
);

--Criação da tabela tb_tarifa
CREATE TABLE tb_tarifa(
    id INT GENERATED ALWAYS AS IDENTITY,
    id_tarifa VARCHAR NOT NULL,
    preco FLOAT4 NOT NULL,
    moeda VARCHAR NOT NULL,
    metodo_pagamento INT2 NOT NULL,
    baldiacoes INT2 ,
    tempo_baldiacao INT ,
    fk_agencia INT NOT NULL,
    PRIMARY KEY(id),
    CONSTRAINT fk_tarifa_agencia FOREIGN KEY(fk_agencia)
          REFERENCES tb_agencia(id)
);

--Criação da tabela tb_linha
CREATE TABLE tb_linha(
    id INT GENERATED ALWAYS AS IDENTITY,
    id_linha VARCHAR NOT NULL,
    nome_curto VARCHAR ,
    nome_longo VARCHAR ,
    descricao VARCHAR,
    tipo INT4 NOT NULL,
    cor VARCHAR ,
    cor_texto VARCHAR,
    fk_tarifa INT4,
    fk_agencia INT4 NOT NULL,
    PRIMARY KEY(id),
    CONSTRAINT fK_linha_tarifa FOREIGN KEY(fk_tarifa) REFERENCES
tb_tarifa(id),
    CONSTRAINT fk_linha_agencia FOREIGN KEY(fk_agencia) REFERENCES
tb_agencia(id)
);

--Criação da tabela tb_ponto_espaço
CREATE TABLE tb_ponto_espaco(
    id INT GENERATED ALWAYS AS IDENTITY,
    latitude VARCHAR NOT NULL,
    longitude VARCHAR NOT NULL,
    PRIMARY KEY(id)
);

--Criação da tabela tb_parada
CREATE TABLE tb_parada(
    id INT GENERATED ALWAYS AS IDENTITY,
```

```
            id_parada VARCHAR NOT NULL,
            fk_ponto_espaco INT4 ,
            codigo VARCHAR,
            descricao VARCHAR,
            zona VARCHAR,
            tipo_localizacao INT2,
            fuso_horario VARCHAR,
            acesso_cadeira_de_rodas INT2,
            fk_parada INT2,
            url VARCHAR,
            codigo_plataforma VARCHAR,
            nome VARCHAR,
            PRIMARY KEY(id),
            CONSTRAINT fk_parada_ponto_espaco FOREIGN KEY(fk_ponto_espaco)
REFERENCES tb_ponto_espaco(id),
            CONSTRAINT fk_parada_parada FOREIGN KEY(fk_parada) REFERENCES
tb_parada(id)
      );

      --Criação da tabela tb_trajeto
      CREATE TABLE tb_trajeto(
            id INT GENERATED ALWAYS AS IDENTITY,
            id_trajeto VARCHAR NOT NULL,
            PRIMARY KEY(id)
      );

      --Criação da tabela tb_ponto_espaço_trajeto
      CREATE TABLE tb_ponto_espaco_trajeto(
            id INT GENERATED ALWAYS AS IDENTITY,
            fk_ponto_espaco INT4 NOT NULL,
            fk_trajeto INT4 NOT NULL,
            distancia_inicio FLOAT4 ,
            ordem_sequencia INT4 NOT NULL,
            PRIMARY KEY(id),
            CONSTRAINT fk_ponto_espaco FOREIGN KEY(fk_ponto_espaco)
REFERENCES tb_ponto_espaco(id),
            CONSTRAINT fk_trajeto FOREIGN KEY(fk_trajeto) REFERENCES
tb_trajeto(id)
      );

      --Criação da tabela tb_data
      CREATE TABLE tb_data(
            id INT GENERATED ALWAYS AS IDENTITY,
            data DATE NOT NULL,
            PRIMARY KEY(id)
      );

      --Criação da tabela tb_dia_servico
      CREATE TABLE tb_dia_servico(
            id INT GENERATED ALWAYS AS IDENTITY,
            id_dia_servico VARCHAR NOT NULL,
            segunda_feira BOOLEAN ,
            terca_feira BOOLEAN ,
            quarta_feira BOOLEAN,
            quinta_feira BOOLEAN,
            sexta_feira BOOLEAN ,
            sabado BOOLEAN ,
            domingo BOOLEAN ,
            fk_data_inicio int4 ,
            fk_data_fim int4 ,
            PRIMARY KEY(id),
```

```
          CONSTRAINT fk_dia_servico_data_inicio FOREIGN KEY(fK_data_inicio)
REFERENCES tb_data(id),
          CONSTRAINT fk_dia_servico_data_fim FOREIGN KEY(fK_data_fim)
REFERENCES tb_data(id)
      );

      --Criação da tabela tb_data_dia_servico
      CREATE TABLE tb_data_dia_servico(
          id INT GENERATED ALWAYS AS IDENTITY,
          fk_dia_servico INT4 NOT NULL,
          fk_data INT4 NOT NULL,
          tipo_excecao INT2 NOT NULL,
          PRIMARY KEY(id),
          CONSTRAINT fk_dia_servico FOREIGN KEY(fk_dia_servico) REFERENCES
tb_dia_servico(id),
          CONSTRAINT fk_data FOREIGN KEY(fk_data) REFERENCES tb_data(id)
      );

      --Criação da tabela tb_viagem
      CREATE TABLE tb_viagem(
          id INT GENERATED ALWAYS AS IDENTITY,
          fk_trajeto INT4,
          fk_linha INT4 NOT NULL,
          fk_dia_servico INT4 NOT NULL,
          id_viagem VARCHAR NOT NULL,
          letreiro VARCHAR,
          nome_curto VARCHAR,
          direcao BOOLEAN,
          hora_inicio TIME,
          hora_fim TIME,
          intervalo INT4,
          pontualidade BOOLEAN,
          PRIMARY KEY(id),
          CONSTRAINT fk_viagem_trajeto FOREIGN KEY(fk_trajeto) REFERENCES
tb_trajeto(id),
          CONSTRAINT fk_viagem_linha FOREIGN KEY(fk_linha) REFERENCES
tb_linha(id),
          CONSTRAINT fk_viagem_dia_servico FOREIGN KEY(fk_dia_servico)
REFERENCES tb_dia_servico(id)
      );

      --Criação da tabela tb_parada_viagem
      CREATE TABLE tb_parada_viagem(
          id INT GENERATED ALWAYS AS IDENTITY,
          fk_parada INT4 NOT NULL,
          fk_viagem INT4 NOT NULL,
          ordem INT4 NOT NULL,
          horario_chegada VARCHAR,
          horario_saida VARCHAR,
          letreiro VARCHAR,
          pontualidade BOOLEAN,
          distancia_percorrida FLOAT4,
          PRIMARY KEY(id),
          CONSTRAINT fk_parada FOREIGN KEY(fk_parada) REFERENCES
tb_parada(id),
          CONSTRAINT fk_viagem FOREIGN KEY(fk_viagem) REFERENCES
tb_viagem(id)
      );

      --Criação da função utilizada na consulta c15
      create or replace  function select_tb_parada(campo varchar)
```

```
      returns  table(id_parada VARCHAR,atributo  varchar,valor varchar) as
$$
      declare consulta varchar;
      begin
            consulta :='SELECT id_parada,'||
quote_literal(campo)||'::varchar,'||campo||'::varchar FROM tb_parada';
            return QUERY execute consulta;
      end$$LANGUAGE plpgsql;
```

APÊNDICE C – *JOB* DE INSERÇÃO DE DADOS NO BANCO RELACIONAL

Figura 35 – *Job* carga_bancos_relacionais parte 1

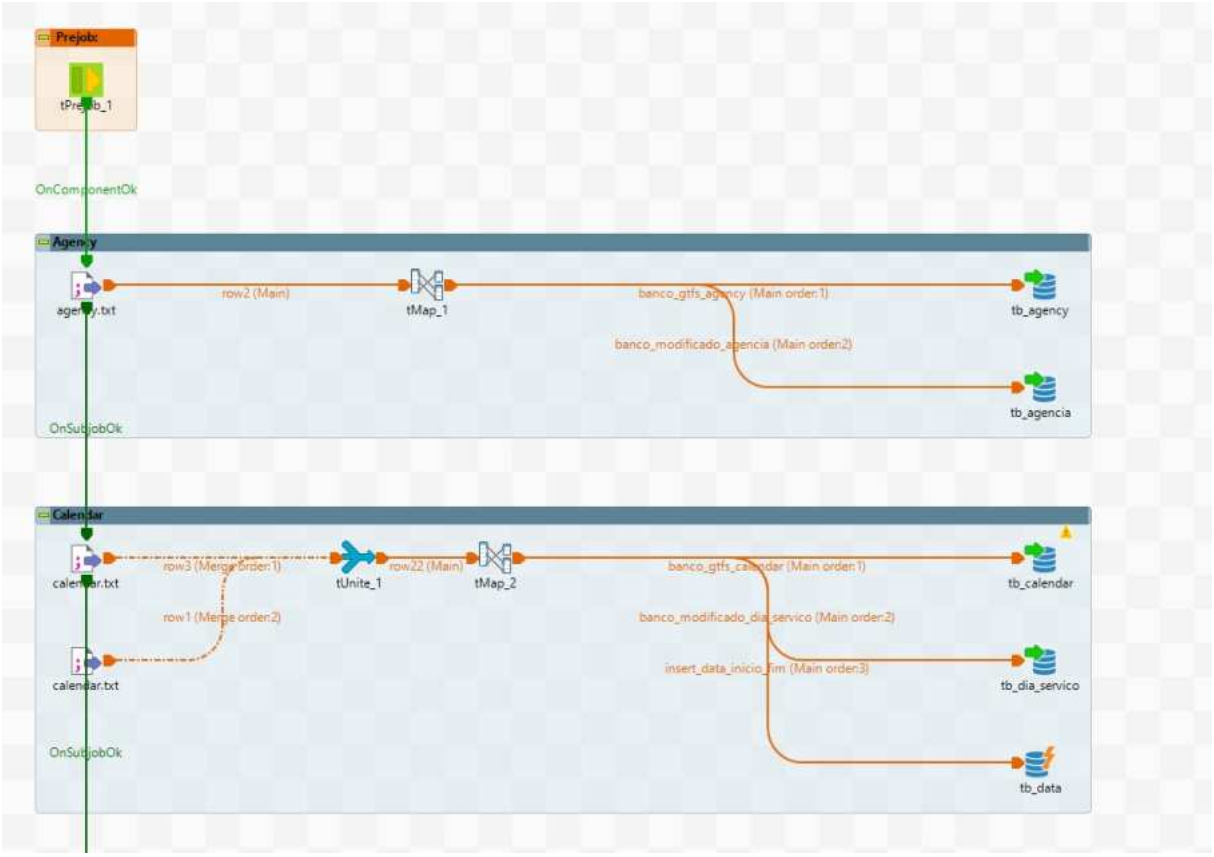


Figura 36 – *Job* carga_bancos_relacionais parte 2

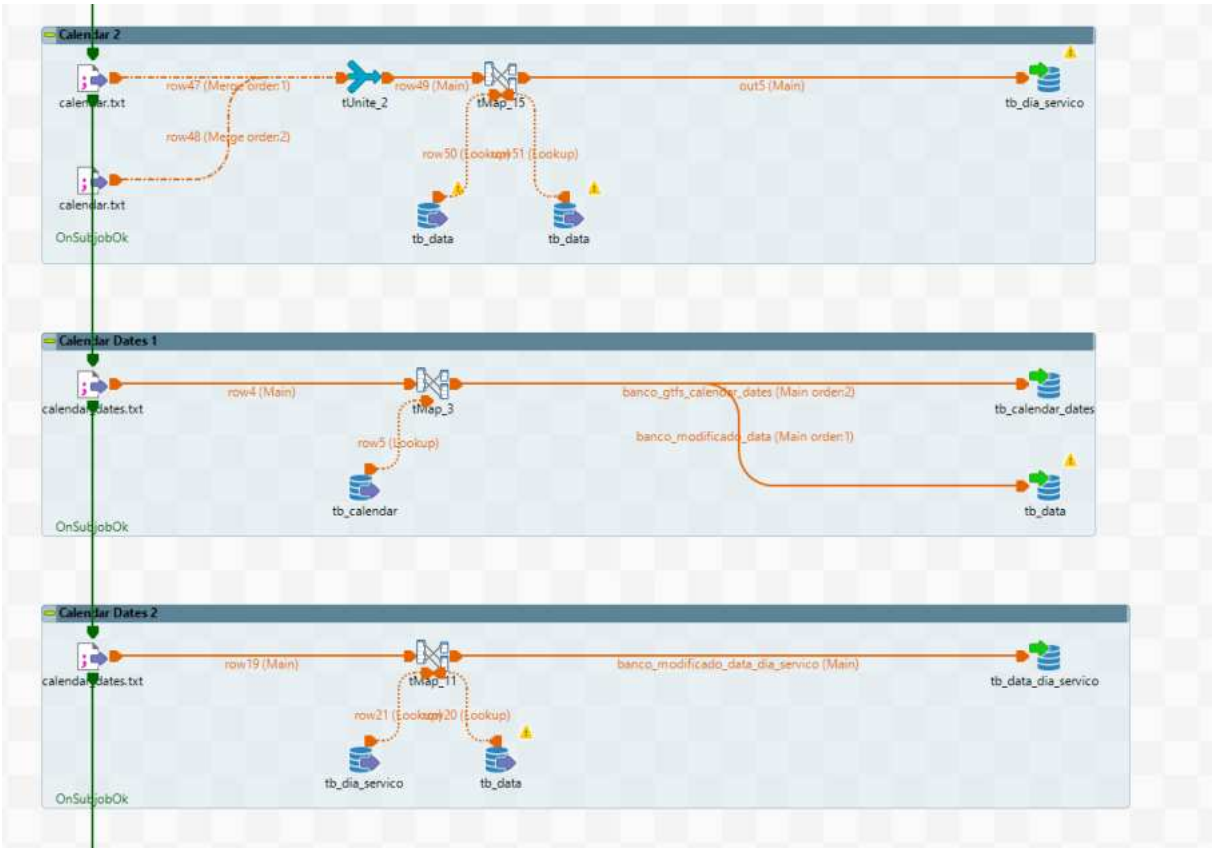


Figura 37 – *Job* carga_bancos_relacionais parte 3

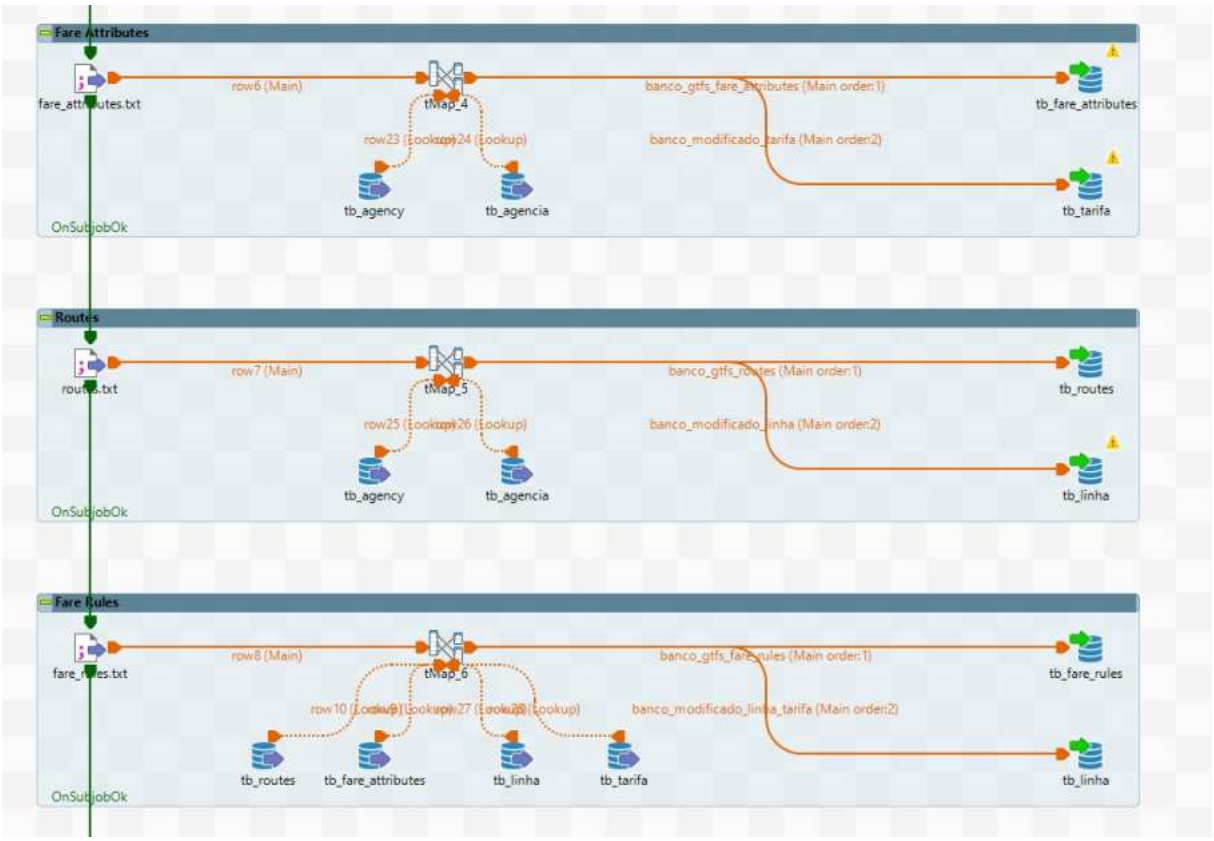


Figura 38 – *Job* carga_bancos_relacionais parte 4

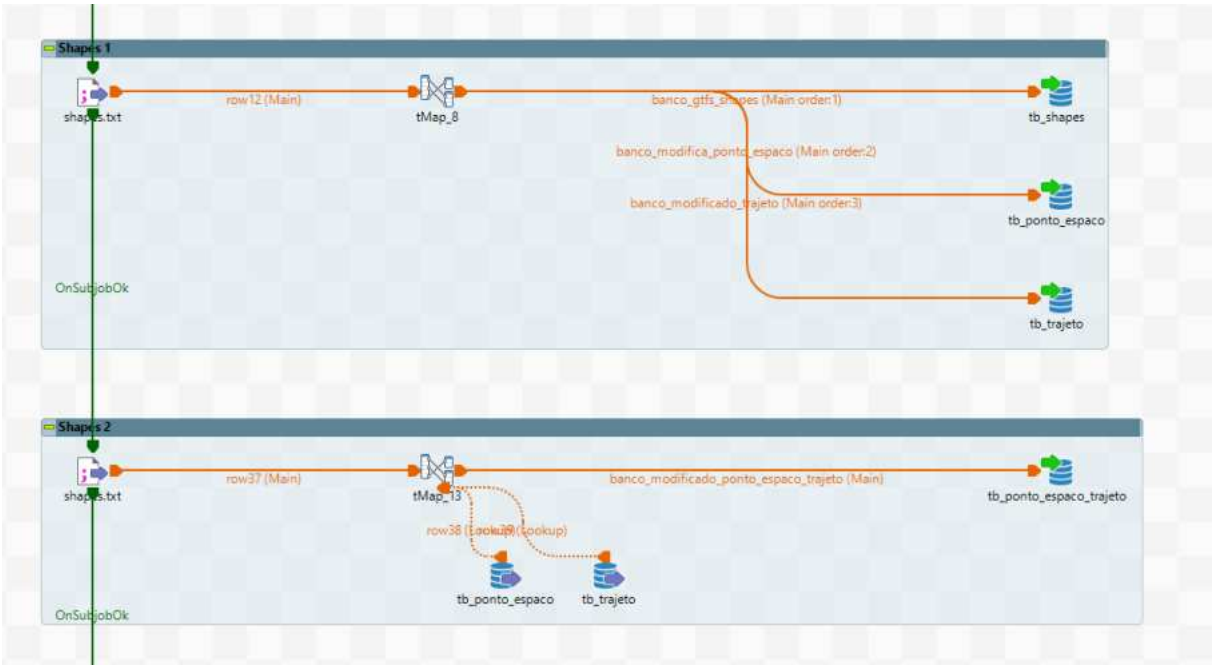


Figura 39 – *Job* carga_bancos_relacionais parte 5

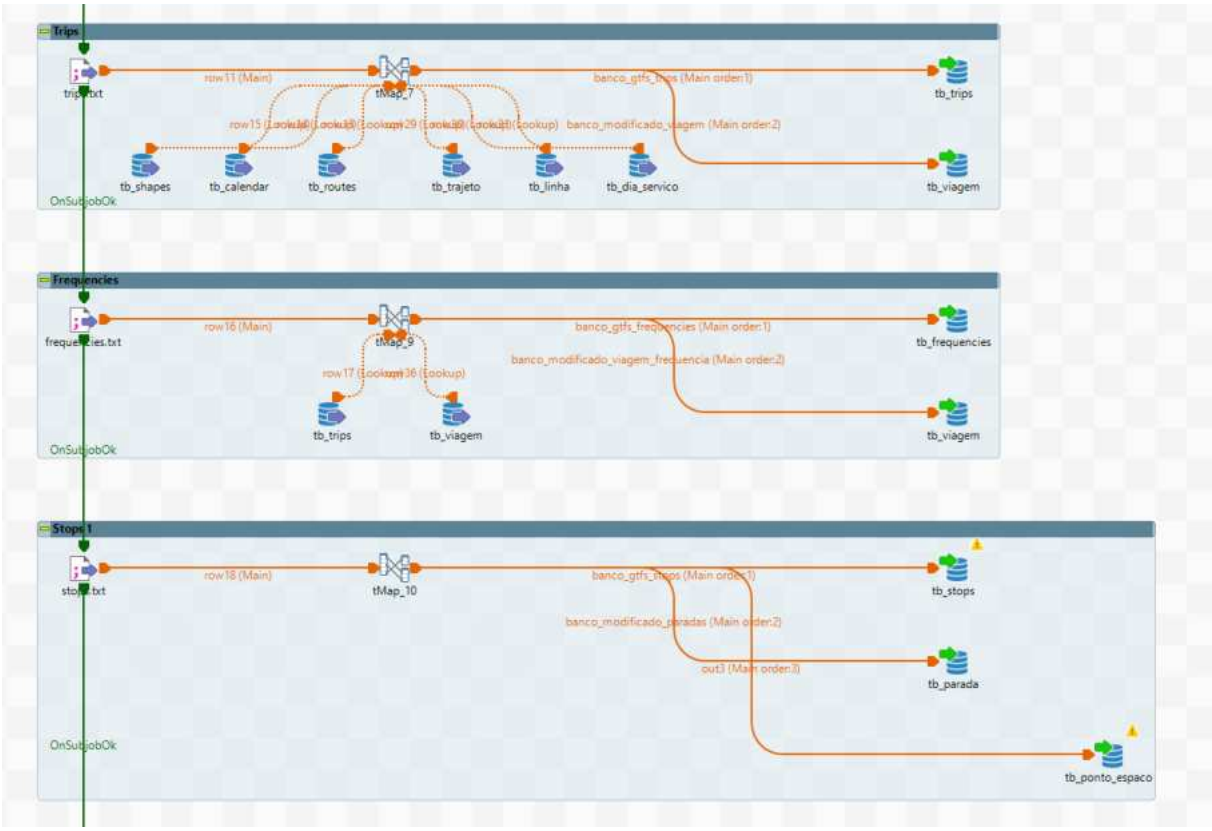
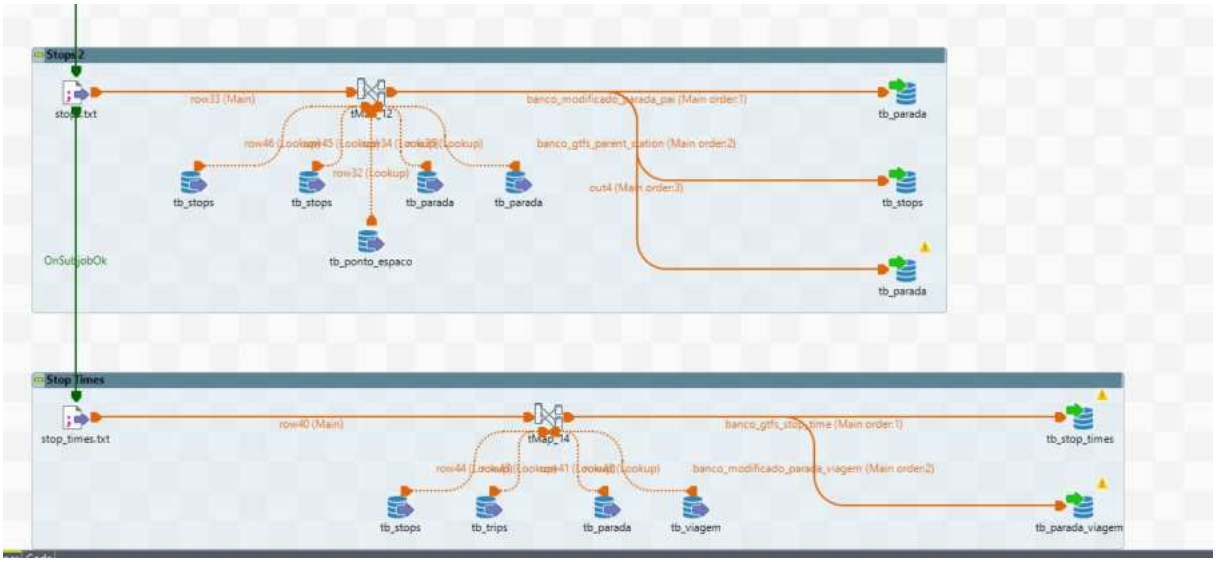


Figura 40 – *Job* carga_bancos_relacionais parte 6



APÊNDICE D – CONSULTAS EXECUTADAS NO BANCO GTFS

```
      -- c1 Lista de todas as coordenadas e seus trajetos associados.
      SELECT shape_id,shape_pt_lat,shape_pt_lon,shape_dist_traveled
      FROM tb_shapes;

      -- c2 Todas as paradas, e algumas de suas propriedades, que estejam
localizada a uma latitude maior que a média.
      SELECT stop_id,stop_desc,wheelchair_boarding ,stop_lat,stop_lon
      FROM tb_stops
      where  stop_lat::numeric > (select  avg(stop_lat::real) from tb_stops
ts )

      -- c3 Informação de acessibilidade de todas as estações se
disponível.
      SELECT
stop_id,location_type,stop_desc,stop_lat,stop_lon,wheelchair_boarding
      FROM tb_stops
      WHERE location_type = 1;

      -- c4 Lista de todas as agências e suas rotas, com algumas de suas
propriedades.
      SELECT route_id,
route_short_name,route_long_name,route_desc,agency_id,agency_url,agency_nam
e
      FROM tb_agency a LEFT JOIN tb_routes r on a.id = r.fk_agency

      -- c5 Dias de serviço adicionados numa data específica.
      SELECT service_id,date,exception_type
      FROM tb_calendar c
      join tb_calendar_dates cd on c.id = cd.fk_calendar
      WHERE cd.exception_type  = 1 and date = '20240101'

      -- c6 Quantidade de Rotas em uma agência especificada.
      SELECT count(*)
      FROM tb_agency a
      JOIN tb_routes r on a.id = r.fk_agency
      WHERE  a.agency_name ='Internorte'

      -- c7 Lista de todas as paradas sem informação sobre acessibilidade
para cadeira de rodas ao longo de uma linha específica.
      SELECT route_short_name,route_desc,trip_short_name,stop_desc,
stop_lat, stop_lon
      FROM tb_stops s
      JOIN tb_stop_times st on s.id = st.fk_stops
      JOIN tb_trips t on st.fk_trips = t.id
      JOIN  tb_routes r on t.fk_routes = r.id
      WHERE
          (s.wheelchair_boarding = 0 or s.wheelchair_boarding is null)
          AND r.route_short_name  = '913'
      union
      SELECT route_short_name,route_desc,trip_short_name,s1.stop_desc,
s1.stop_lat, s1.stop_lon
      FROM tb_stops s1
      JOIN tb_stop_times st on s1.id = st.fk_stops
      join tb_stops s2 on s1.fk_stops  =s2.id
      JOIN tb_trips t on st.fk_trips = t.id
      JOIN  tb_routes r on t.fk_routes = r.id
```

```
	where
		(s1.wheelchair_boarding = 0 or s1.fk_stops is null)
		and (s2.wheelchair_boarding = 0 or s2.fk_stops is null)
		and r.route_short_name  = '913'

	-- c8 Lista de linhas com suas viagens, dias de serviço, paradas e
tempo de paradas associados mais algumas propriedades adicionais.
	SELECT distinct
route_id,route_short_name,route_desc,trip_id,trip_short_name,service_id,sto
p_id,stop_desc
	FROM tb_routes r
	LEFT JOIN tb_trips t on t.fk_routes = r.id
	LEFT JOIN tb_calendar c on t.fk_calendar =c.id
	LEFT JOIN tb_stop_times st on st.fk_trips = t.id
	LEFT JOIN tb_stops s on st.fk_stops = s.id

	-- c9 Viagens e os trajetos associados.
	SELECT DISTINCT trip_id,shape_id
	FROM tb_trips t
	JOIN tb_shapes s on t.fk_shapes = s.id

	-- c10  Número de Viagens com duração maior que 30 minutos.
	SELECT COUNT(DISTINCT t.id)
	FROM tb_trips t
	LEFT JOIN tb_stop_times st on t.id = st.fk_trips
	WHERE departure_time>='00:30:00'

	-- c11 Viagens que estão disponíveis numa certa data.
	select
		distinct  trip_id
	from tb_trips t
	join tb_calendar c on t.fk_calendar = c.id
	left join tb_calendar_dates cd on cd.fk_calendar =c.id
	where
		c.start_date <'2024-01-01'
		and c.end_date >'2024-01-01'
		and c.id  not in(
			select fk_calendar
			from tb_calendar_dates tcd
			where date = '2024-01-01'
			and exception_type = 2
		)

	-- c12 Número de paradas sem informação sobre acessibilidade por
cadeira de rodas agrupadas por linha.
	SELECT route_long_name,count(stop_name) from(
		select r.route_long_name,s.stop_name
		FROM tb_routes  r
		LEFT JOIN tb_trips t on r.id =t.fk_routes
		LEFT JOIN tb_stop_times st on st.fk_trips = t.id
		LEFT JOIN tb_stops s on s.id= st.fk_stops
		WHERE
		    (wheelchair_boarding is null or wheelchair_boarding = 0)
		     and s.fk_stops is null
		union
		select r.route_long_name,s.stop_name
		FROM tb_routes  r
		LEFT JOIN tb_trips t on r.id =t.fk_routes
		LEFT JOIN tb_stop_times st on st.fk_trips = t.id
		LEFT JOIN tb_stops s on s.id= st.fk_stops
		 join tb_stops s2 on s.fk_stops  = s2.id
```

```
            where
                  (s.wheelchair_boarding =0 or s.wheelchair_boarding is
null)
                  and (s2.wheelchair_boarding = 0 or s2.wheelchair_boarding
is null)
      )
      GROUP BY route_long_name

      -- c13 Todos os acessos de estações.
      Select s1.stop_id as parada,s2.stop_id as "estação Pai",s1.stop_name
as "Nome da Parada", s2.stop_name as "Nome da Estação Pai"
      FROM tb_stops s1
      JOIN tb_stops s2 on s1.fk_stops  = s2.id
      where s1.fk_stops is not null

      -- c14 Todas os horários de parada e suas relativas linhas e ordem de
parada, ordenados pela ordem de parada.
      SELECT trip_id,stop_id,stop_sequence,route_id,stop_name
      FROM tb_trips t
      JOIN tb_stop_times st on t.id =st.fk_trips
      JOIN tb_stops s on s.id =st.fk_stops
      JOIN tb_routes r on t.fk_routes = r.id
      ORDER BY stop_sequence

      -- c15 Qualquer propriedade contendo a sequência de caracteres
especificada.
       SELECT stop_id,atributo,valor
       FROM(
          select  * from select_tb_stops('stop_id')
          UNION
          select  * from select_tb_stops('stop_code')
          UNION
          select  * from select_tb_stops('stop_name')
          UNION
          select  * from select_tb_stops('stop_desc')
          UNION
          select  * from select_tb_stops('stop_lat')
          UNION
          select  * from select_tb_stops('stop_lon')
          UNION
          select  * from select_tb_stops('zone_id')
          UNION
          select  * from select_tb_stops('location_type')
          UNION
          SELECT s1.stop_id,'parent_station',s2.stop_id
          FROM tb_stops s1
          JOIN tb_stops s2 on s1.fk_stops = s2.id
          UNION
          select  * from select_tb_stops('stop_timezone')
          UNION
          select stop_id,'wheelchair_boarding',   CASE WHEN
wheelchair_boarding = 1 THEN  'WheelchairAccessible'
              ELSE CASE WHEN wheelchair_boarding = 2  THEN
'NotWheelcharAccessible'
                  ELSE  NULL END  end
          from tb_stops s
          UNION
          select  * from select_tb_stops('platform_code')
       ) as t
       where valor like '%CCMN%'
```

```
      -- c16 Para todas as linhas, todas as adições no calendário durante
um mês especificado.
      SELECT trip_id,service_id,route_id,date,exception_type
      FROM tb_trips t
      JOIN tb_routes r on t.fk_routes = r.id
      JOIN tb_calendar c on t.fk_calendar = c.id
      JOIN tb_calendar_dates cd on cd.fk_calendar = c.id
      WHERE exception_type = 1
      AND EXTRACT(month from date) = 1  and EXTRACT(year from date) =2024

      -- c17 Viagens com seus tempos de início e fim e linhas associadas.
      SELECT route_short_name, route_type,trip_id,start_time,end_time
      FROM tb_trips t
      JOIN tb_routes r on t.fk_routes = r.id
      JOIN tb_frequencies f on f.fk_trips = t.id

      -- c18 Todas as linhas com viagens no domingo.
      SELECT service_id,trip_id,route_id
      FROM
          tb_calendar c
          JOIN tb_trips t on t.fk_calendar = c.id
          JOIN tb_routes r on t.fk_routes = r.id
      WHERE
          sunday = TRUE

      -- c19 Todas as paradas acessíveis a partir de uma parada
determinada.
      select distinct  s1.stop_id, s1.stop_name
      from tb_stops   s1
      join tb_stop_times st1 on s1.id = st1.fk_stops
      join tb_trips t on st1.fk_trips = t.id
      join tb_stop_times st2 on st2.fk_trips  = t.id
      join tb_stops   s2 on s2.id= st2.fk_stops
      join tb_calendar c on c.id = t.fk_calendar
      where
            st1.stop_sequence >st2.stop_sequence
            and s2.stop_name = 'CCMN - Geografia'
            and c.service_id  = 'U_REG'
```

APÊNDICE E – CONSULTAS EXECUTADAS NO BANCO MODIFICADO

```
      -- c1 Lista de todas as coordenadas e seus trajetos associados.
      SELECT id_trajeto,longitude,latitude,ordem_sequencia
      FROM
          tb_ponto_espaco pt
          join tb_ponto_espaco_trajeto pet on pt.id = pet.fk_ponto_espaco
          join tb_trajeto t on t.id = pet.fk_trajeto

      -- c2 Todas as paradas, e algumas de suas propriedades, que estejam
localizada a uma latitude maior que a média.
      SELECT id_parada,descricao,acesso_cadeira_de_rodas,latitude,longitude
      FROM tb_parada p
      JOIN tb_ponto_espaco pt on p.fk_ponto_espaco = pt.id
      WHERE latitude::real > (select avg(latitude::real) from tb_parada p
JOIN tb_ponto_espaco pt on p.fk_ponto_espaco = pt.id)

      -- c3 Informação de acessibilidade de todas as estações se
disponível.
      SELECT
id_parada,tipo_localizacao,descricao,latitude,longitude,acesso_cadeira_de_r
odas
      FROM tb_parada p
      JOIN tb_ponto_espaco pt on p.fk_ponto_espaco = pt.id
      WHERE tipo_localizacao = 1

      -- c4 Lista de todas as agências e suas rotas, com algumas de suas
propriedades.
      SELECT id_linha, nome_curto,nome_longo,descricao,id_agencia,url,nome
      FROM tb_linha l
      JOIN tb_agencia a on l.fk_agencia = a.id

      -- c5 Dias de serviço adicionados numa data específica.
      SELECT d.id_dia_servico,data,tipo_excecao
      FROM tb_dia_servico d
      JOIN tb_data_dia_servico dds on dds.fk_dia_servico = d.id
      JOIN tb_data dt on dds.fk_data = dt.id
      WHERE   data ='2024-01-01'  and tipo_excecao =1

      -- c6 Quantidade de Rotas em uma agência especificada.
      SELECT count(*)
      FROM tb_linha l
      join tb_agencia a on l.fk_agencia  = a.id
      where a.nome = 'Internorte'

      -- c7 Lista de todas as paradas sem informação sobre acessibilidade
para cadeira de rodas ao longo de uma linha específica.
      SELECT DISTINCT
l.nome_curto,l.descricao,v.nome_curto,p.descricao,latitude,longitude
      FROM tb_linha l
      JOIN tb_viagem v on v.fk_linha = l.id
      JOIN tb_parada_viagem pv on pv.fk_viagem = v.id
      JOIN tb_parada p  on pv.fk_parada = p.id
      join tb_ponto_espaco pt on p.fk_ponto_espaco = pt.id
      WHERE l.nome_curto  = '913'

      -- c8 Lista de linhas com suas viagens, dias de serviço, paradas e
tempo de paradas associados mais algumas propriedades adicionais.
```

```
      SELECT DISTINCT
id_linha,l.nome_curto,l.descricao,id_viagem,v.nome_curto,
id_dia_servico,id_parada,p.descricao
      FROM tb_linha l
      JOIN tb_viagem v on l.id = v.fk_linha
      JOIN tb_parada_viagem pv on v.id = pv.fk_viagem
      JOIN tb_parada p on p.id =pv.fk_parada
      JOIN tb_dia_servico ds on v.fk_dia_servico = ds.id

      -- c9 Viagens e os trajetos associados.
      SELECT distinct  id_viagem,id_trajeto
      FROM tb_viagem v
      JOIN tb_trajeto t on v.fk_trajeto = t.id

      -- c10 Número de Viagens com duração maior que 30 minutos.
      Select count(distinct fk_viagem)
      FROM tb_parada_viagem
      WHERE horario_saida>='00:30:00'

      -- c11 Viagens que estão disponíveis numa certa data.
      select distinct id_viagem
      from tb_viagem v
      join tb_dia_servico ds on v.fk_dia_servico =ds.id
      join tb_data data_inicio on ds.fk_data_inicio = data_inicio.id
      join tb_data data_fim on ds.fk_data_fim = data_fim.id
      where
            data_inicio.data <'2024-01-01'
            and data_fim.data >'2024-01-01'
            and ds.id not in(
                  select fk_dia_servico
                  from tb_data_dia_servico dds
                  join tb_data d on dds.fk_data  =d.id
                  where
                        data = '2024-01-01'
                        and tipo_excecao = 2
            )

      -- c12 Número de paradas sem informação sobre acessibilidade por
cadeira de rodas agrupadas por linha.
      SELECT nome_longo, count(nome)from (
      select l.nome_longo, p.nome
      FROM tb_linha l
      JOIN tb_viagem v on v.fk_linha =l.id
      JOIN tb_parada_viagem pv on pv.fk_viagem =v.id
      JOIN tb_parada p on pv.fk_parada =p.id
      WHERE (p.acesso_cadeira_de_rodas is null or
p.acesso_cadeira_de_rodas=0)
      union
      select l.nome_longo, p.nome
      FROM tb_linha l
      JOIN tb_viagem v on v.fk_linha =l.id
      JOIN tb_parada_viagem pv on pv.fk_viagem =v.id
      JOIN tb_parada p on pv.fk_parada =p.id
      join tb_parada  p2 on p.fk_parada  = p2.id
      WHERE  (p.acesso_cadeira_de_rodas is null or
p.acesso_cadeira_de_rodas=0) and (p2.acesso_cadeira_de_rodas is null or
p2.acesso_cadeira_de_rodas=0)
      )GROUP BY nome_longo

      -- c13 Todos os acessos de estações.
```

```
      SELECT p1.id_parada as parada,p2.id_parada as "Parada Pai",p1.nome as
"Nome da Parada",p2.nome as "Nome Parada Pai"
      FROM tb_parada p1
      JOIN tb_parada p2 on p1.fk_parada = p2.id
      WHERE p1.tipo_localizacao = 0

      -- c14 Todas os horários de parada e suas relativas linhas e ordem de
parada, ordenados pela ordem de parada.
      SELECT id_parada,id_viagem,ordem,id_linha,p.nome
      FROM tb_parada_viagem pv
      JOIN tb_parada p on pv.fk_parada =p.id
      JOIN tb_viagem v on pv.fk_viagem = v.id
      JOIN tb_linha l on v.fk_linha = l.id
      order by ordem

      -- c15 Qualquer propriedade contendo a sequência de caracteres
especificada.
      SELECT id_parada,atributo,valor
      FROM(
          SELECT * FROM select_tb_parada('id_parada')
          UNION
          SELECT * FROM select_tb_parada('codigo')
          UNION
          SELECT * FROM select_tb_parada('nome')
          UNION
          SELECT * FROM select_tb_parada('descricao')
          UNION
          SELECT id_parada,'latitude', latitude FROM tb_parada p JOIN
tb_ponto_espaco pe on p.fk_ponto_espaco = pe.id
          UNION
          SELECT id_parada,'longitude', longitude FROM tb_parada p JOIN
tb_ponto_espaco pe on p.fk_ponto_espaco = pe.id
          UNION
          SELECT * FROM select_tb_parada('zona')
          UNION
          SELECT * FROM select_tb_parada('tipo_localizacao')
          UNION
          SELECT p1.id_parada,'estação pai',p2.id_parada FROM tb_parada p1
JOIN tb_parada p2 on p1.fk_parada = p2.id
          UNION
          SELECT * FROM select_tb_parada('fuso_horario')
          UNION
          SELECT p.id_parada,'acesso_cadeira_de_rodas', CASE WHEN
acesso_cadeira_de_rodas = 1 THEN  'WheelchairAccessible'
               ELSE CASE WHEN acesso_cadeira_de_rodas = 2  THEN
'NotWheelcharAccessible'
                    ELSE  NULL END  end
          FROM tb_parada p
          UNION
          SELECT * FROM select_tb_parada('codigo_plataforma')
      ) as t
      WHERE valor like '%CCMN%'

      -- c16 Para todas as linhas, todas as adições no calendário durante
um mês especificado.
      SELECT id_viagem,id_dia_servico,id_linha,d.data
      FROM tb_viagem v
      JOIN tb_dia_servico ds on v.fk_dia_servico = ds.id
      JOIN tb_linha l on v.fk_linha = l.id
      JOIN tb_data_dia_servico dds on dds.fk_dia_servico = ds.id
      JOIN tb_data d on dds.fk_data = d.id
```

```
      WHERE
          dds.tipo_excecao = 1
            AND EXTRACT(month from data) = 1  and EXTRACT(year from data)
=2024

      -- c17 Viagens com seus tempos de início e fim e linhas associadas.
      SELECT l.nome_curto,l.tipo,id_viagem,hora_inicio,hora_fim
      FROM tb_linha l
      JOIN tb_viagem v on v.fk_linha = l.id
      where hora_inicio is not null and hora_fim is not null

      -- c18 Todas as linhas com viagens no domingo.
      SELECT id_dia_servico,id_viagem,id_linha,l.nome_longo,l.nome_curto
      FROM tb_linha l
      JOIN tb_viagem v on v.fk_linha = l.id
      join tb_dia_servico ds on v.fk_dia_servico = ds.id
      WHERE domingo = TRUE

      --c19 Todas as paradas acessíveis a partir de uma parada determinada.
      select distinct  p1.id_parada, p1.nome
      from tb_parada p1
      join tb_parada_viagem pv1 on pv1.fk_parada = p1.id
      join tb_viagem v on pv1.fk_viagem = v.id
      join tb_parada_viagem pv2 on pv2.fk_viagem  = v.id
      join tb_parada p2 on pv2.fk_parada =p2.id
      join tb_dia_servico ds on v.fk_dia_servico = ds.id
      where
            pv1.ordem >pv2.ordem
            and p2.nome ='CCMN - Geografia'
            and id_dia_servico ='U_REG'
```

APÊNDICE F – CONSULTAS EXECUTADAS NO BANCO RDF

```
# PREFIXES
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX gtfs: <http://vocab.gtfs.org/ter#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX dct: <http://purl.org/dc/ter/>
PREFIX schema: <http://schema.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

# c1 Lista de todos os trajetos com algumas de suas propriedades.
SELECT * WHERE {
    ?shape a gtfs:Shape .
    ?shape gtfs:shapePoint ?shapePoint.
    ?shapePoint geo:lat ?shape_pt_lat .
    ?shapePoint geo:long ?shape_pt_lon .
    ?shapePoint gtfs:pointSequence ?shape_pt_sequence .
}

# c2 Todas as paradas, e algumas de suas propriedades, que estejam
localizada a uma latitude maior que a média.
SELECT * WHERE {
    ?stop a gtfs:Stop .
    OPTIONAL { ?stop dct:description ?stopDescription . }
    OPTIONAL { ?stop gtfs:wheelchairAccessible ?wheelchairAccesible }
    ?stop geo:lat ?stopLat .
    ?stop geo:long ?stopLong .
    {
        select (avg(xsd:float(?stopLat)) as ?x) where{
            ?stop a gtfs:Stop .
            ?stop geo:lat ?stopLat .
        }
    }
    FILTER (xsd:float(?stopLat) > ?x) .
}

# c3 Informação de acessibilidade de todas as estações se disponível.
SELECT * WHERE {
    ?stop a gtfs:Stop .
    ?stop a gtfs:Station.
    OPTIONAL { ?stop dct:description ?stopDescription . }
    OPTIONAL { ?stop geo:lat ?stopLat ; geo:long ?stopLong . }
    OPTIONAL { ?stop gtfs:wheelchairAccessible ?
wheelchairAccessible . }

}

# c4 Lista de todas as agências e suas rotas, com algumas de suas
propriedades.
SELECT * WHERE {
    ?route a gtfs:Route .
    OPTIONAL { ?route gtfs:shortName ?routeShortName . }
    OPTIONAL { ?route gtfs:longName ?routeLongName . }
    OPTIONAL { ?route dct:description ?routeDescription . }
    ?route gtfs:agency ?agency .
    ?agency a gtfs:Agency .
```

```
            ?agency foaf:page ?agencyPage .
            ?agency foaf:name ?agencyName .
            OPTIONAL { ?agency foaf:phone ?agencyPhone . }
        }

        #c5 Dias de serviço adicionados numa data específica.
        SELECT * WHERE {
            ?service a gtfs:Service .
            ?service gtfs:serviceRule ?serviceRule .
            ?serviceRule a gtfs:CalendarDateRule .
            ?serviceRule dct:date ?date .
            ?serviceRule gtfs:dateAddition "1"^^ xsd:boolean .
            FILTER(?date = "20240101"^^ xsd:date)
        }

        # c6 Quantidade de Rotas em uma agência especificada.
        SELECT (count(?route) as ?nRoutes) WHERE {
            ?route a gtfs:Route .
            ?route gtfs:agency ?agency .
            ?agency foaf:name ?nome
            FILTER (?nome="Internorte")
        }

        # c7 Lista de todas as paradas sem informação sobre acessibilidade
para cadeira de rodas ao longo de uma linha específica.
        SELECT DISTINCT ?routeShortName ?routeDescription ?tripShortName ?
stopDescription ?stopLat ?stopLong WHERE {
            ?route a gtfs:Route .
            OPTIONAL { ?route gtfs:shortName ?routeShortName . }
            OPTIONAL { ?route dct:description ?routeDescription . }
            ?trip a gtfs:Trip .
            OPTIONAL { ?trip gtfs:shortName ?tripShortName . }
            ?trip gtfs:service ?service .
            ?trip gtfs:route ?route .
            ?stopTime a gtfs:StopTime .
            ?stopTime gtfs:trip ?trip .
            ?stopTime gtfs:stop ?stop .
            ?stop a gtfs:Stop .
            OPTIONAL { ?stop dct:description ?stopDescription . }
            OPTIONAL { ?stop geo:lat ?stopLat ; geo:long ?stopLong . }
            {{?stop gtfs:wheelchairAccessible
gtfs:NoWheelchairAccessInformationAvaiable.}
            UNION
            {?stop gtfs:wheelchairAccessible gtfs:CheckParentStation.
             ?stop ^gtfs:parentStation ?parentStation.
             ?parentStation gtfs:wheelchairAccessible
gtfs:NoWheelchairAccessInformationAvaiable. }}
            FILTER (?routeShortName='913')
        }

        # c8 Lista de linhas com suas viagens, dias de serviço, paradas e
tempo de paradas associados mais algumas propriedades adicionais.
        SELECT * WHERE {
            ?route a gtfs:Route .
            OPTIONAL { ?route gtfs:shortName ?routeShortName . }
            OPTIONAL { ?route dct:description ?routeDescription . }
            ?trip a gtfs:Trip .
            OPTIONAL { ?trip gtfs:shortName ?tripShortName . }
            ?trip gtfs:service ?service .
            ?trip gtfs:route ?route .
            ?stopTime a gtfs:StopTime .
```

```
        ?stopTime gtfs:trip ?trip .
        ?stopTime gtfs:stop ?stop .
        ?stop a gtfs:Stop .
        OPTIONAL {?stop dct:description ?stopDescription . }
        ?service a gtfs:Service .
    }

    # c9 Viagens e os trajetos associados.
    SELECT * WHERE {
        ?trip a gtfs:Trip .
        ?trip gtfs:shape ?shape .
        ?shape a gtfs:Shape .
    }

    # c10 Número de Viagens com duração maior que 30 minutos.
    SELECT (count(distinct ?trip) as ?count) WHERE {
        ?trip a gtfs:Trip .
        ?stopTime a gtfs:StopTime .
        ?stopTime gtfs:trip ?trip .
        ?stopTime gtfs:departureTime ?departureTime .
        FILTER (?departureTime >= "00:30:00"^^ xsd:duration)
    }

    # c11 Viagens que estão disponíveis numa certa data.
    SELECT distinct ?trip WHERE {
        ?service a gtfs:Service .
        ?service gtfs:serviceRule ?calendarRule .
        ?trip gtfs:service ?service .
        ?calendarRule a gtfs:CalendarRule .
        ?calendarRule dct:temporal ?period.
        ?period schema:startDate ?startDate .
        ?period schema:endDate ?endDate .
        FILTER (?startDate <'20240101'^^xsd:date) .
        FILTER (?endDate > '20240101'^^xsd:date) .
        FILTER NOT EXISTS {
            ?service gtfs:serviceRule ?calendarDateRule .
            ?calendarDateRule a gtfs:CalendarDateRule .
            ?calendarDateRule dct:date '20240101'^^xsd:date .
            ?calendarDateRule gtfs:dateAddition "0"^^ xsd:boolean
        }
    }

    # c12 Número de paradas sem informação sobre acessibilidade por
cadeira de rodas agrupadas por linha.
    SELECT ?longName (count(distinct ?name) as ?count) WHERE {
        {
            ?route a gtfs:Route .
            ?route gtfs:longName ?longName .
            ?trip a gtfs:Trip .
            ?trip gtfs:route ?route .
            ?stopTime a gtfs:StopTime .
            ?stopTime gtfs:trip ?trip .
            ?stopTime gtfs:stop ?stop .
            ?stop a gtfs:Stop.
            ?stop foaf:name ?name.
            ?stop gtfs:wheelchairAccessible
gtfs:NoWheelchairAccessInformationAvaiable .
        }
        UNION  {
            ?route a gtfs:Route .
            ?route gtfs:longName ?longName .
```

```
            ?trip a gtfs:Trip .
            ?trip gtfs:route ?route .
            ?stopTime a gtfs:StopTime .
            ?stopTime gtfs:trip ?trip .
            ?stopTime gtfs:stop ?stop .
            ?stop a gtfs:Stop .
            ?stop foaf:name ?name.
            ?stop ^gtfs:parentStation ?parentStation.
            ?parentStation foaf:name ?parentStationName.
            ?stop gtfs:wheelchairAccessible gtfs:CheckParentStation.
            ?parentStation gtfs:wheelchairAccessible
gtfs:NoWheelchairAccessInformationAvaiable
        }
    } GROUP BY ?longName

    # c13 Todos os acessos de estações.
    SELECT * WHERE {
        ?stop a gtfs:Stop .
        ?parStation gtfs:parentStation ?stop .
        OPTIONAL {?stop foaf:name ?accName} .
        ?parStation a gtfs:Station .
        ?parStation foaf:name ?parentName
    }

    # c14 - Todas os horários de parada e suas relativas linhas e ordem
de parada, ordenados pela ordem de parada.
    SELECT * WHERE {
        ?stopTime a gtfs:StopTime .
        ?stopTime gtfs:trip ?trip .
        ?stopTime gtfs:stop ?stop .
        ?stopTime gtfs:stopSequence ?sequence .
        ?stop a gtfs:Stop .
        ?trip a gtfs:Trip .
        ?trip gtfs:route ?route .
        OPTIONAL {?stop foaf:name ?stopName}
    } ORDER BY ?sequence

    # c15 Qualquer propriedade contendo a sequência de caracteres
especificada.
    SELECT * WHERE {
        ?stop a gtfs:Stop .
        ?stop ?p ?str .
        FILTER regex (?str, "CCMN")
    }

    # c16 Para todas as linhas, todas as adições no calendário durante um
mês especificado.
    SELECT * WHERE {
        ?trip a gtfs:Trip .
        ?trip gtfs:service ?service .
        ?trip gtfs:route ?route .
        ?service a gtfs:Service .
        ?service gtfs:serviceRule ?serviceRule .
        ?serviceRule a gtfs:CalendarDateRule .
        ?serviceRule dct:date ?servDate .
        ?serviceRule gtfs:dateAddition "1"^^ xsd:boolean .
        FILTER (?servDate >= '20240101'^^xsd:date) .
        FILTER (?servDate <= '20241231'^^xsd:date) .
    }

    # c17 Viagens com seus tempos de início e fim e linhas associadas.
```

```
SELECT ?routeName ?routeType ?trip ?startTime ?endTime WHERE {
    ?trip a gtfs:Trip .
    ?trip gtfs:route ?route .
    ?frequency a gtfs:Frequency .
    ?frequency gtfs:startTime ?startTime .
    ?frequency gtfs:endTime ?endTime .
    ?frequency gtfs:trip ?trip .
    ?route a gtfs:Route .
    ?route gtfs:shortName ?routeName .
    ?route gtfs:routeType ?routeType .
}

# c18 Todas as linhas com viagens no domingo.
SELECT * WHERE {
    ?service a gtfs:Service .
    ?service gtfs:serviceRule ?serviceRule .
    ?serviceRule a gtfs:CalendarRule .
    ?serviceRule gtfs:sunday "1"^^ xsd:boolean .
    ?trip gtfs:service ?service .
    ?trip gtfs:route ?route .
    { ?route gtfs:longName ?longName } UNION
    { ?route gtfs:shortName ?shortName } .
}

# c19 Todas as paradas acessíveis a partir de uma parada determinada.
select distinct ?stopA where{
    ?stopA  a gtfs:Stop.
    ?stopA ^gtfs:stop ?stopTime2.
    ?stopA foaf:name ?nameA.
    ?trip1 a gtfs:Trip.
    ?trip1 gtfs:service ?service.
    ?trip1 ^gtfs:trip ?stopTime1.
    ?trip1 ^gtfs:trip ?stopTime2.
    ?stopTime1 gtfs:stopSequence ?ordem1.
    ?stopTime2 gtfs:stopSequence ?ordem2.
    ?stopB a gtfs:Stop.
    ?stopB foaf:name ?stopBName.
    ?stopB ^gtfs:stop ?stopTime1.
    FILTER(?stopBName = 'CCMN - Geografia').
    FILTER(?ordem2>?ordem1).
    FILTER(STR(?service) = 'http://127.0.0.1:3333/Service/U_REG').
}
```

APÊNDICE G – RESULTADO COMPLETO DOS TESTES DO BANCO GTFS

CONSULTA	TEMPO DE EXECUÇÃO (ms)				
	PRIMEIRA EXECUÇÃO	SEGUNDA EXECUÇÃO	TERCEIRA EXECUÇÃO	QUARTA EXECUÇÃO	QUINTA EXECUÇÃO
c1	42,427	33,947	38,692	35,362	33,196
c2	10,884	8,704	10,010	8,666	10,103
c3	0,726	0,664	0,721	0,684	0,840
c4	0,178	0,151	0,183	0,136	0,144
c5	0,147	0,151	0,170	0,257	0,144
c6	0.157	0,153	0,485	0,225	0,155
c7	155,764	148,687	149,615	150,569	158,817
c8	2.250,421	2.289,268	2.297,700	2.282,512	2.270,128
c9	27,815	29,060	27,491	27,881	28,505
c10	565,151	570,497	567,673	575,726	574,203
c11	14,145	9,954	11,644	10,828	9,903
c12	5.274,855	5.230,313	5.283,034	5.229,312	5.284,907
c13	0,971	0,966	0,963	0,992	1,008
c14	683,581	653,829	645,059	645,264	659,078
c15	70,933	82,955	72,203	69,358	74,167
c16	25,964	25,508	25,438	25,611	25,767
c17	10,611	11,680	10,943	10,696	11,266
c18	3,797	3,358	3,474	3,317	3,346
c19	142,784	141,222	150,090	174,366	142,066

APÊNDICE H – RESULTADO COMPLETO DOS TESTES DO BANCO MODIFICADO

CONSULTA	TEMPO DE EXECUÇÃO (ms)				
	PRIMEIRA EXECUÇÃO	SEGUNDA EXECUÇÃO	TERCEIRA EXECUÇÃO	QUARTA EXECUÇÃO	QUINTA EXECUÇÃO
c1	201,618	158,841	160,009	182,572	163,463
c2	39,053	39,330	38,218	38,169	39,270
c3	0,901	0,827	0,899	0,950	0,798
c4	1,335	0,199	0,189	0,188	0,198
c5	0,206	0,210	0,196	0,205	0,216
c6	0,168	0,166	0,177	0,260	0,166
c7	92,555	99,803	96,282	94,125	97,719
c8	3.916,473	3.648,933	3.759,526	3.680,159	3.819,960
c9	11,753	11,908	12,503	11,740	11,784
c10	468,105	467,699	479,480	479,213	469,044
c11	6,410	6,707	6,528	7,296	6,381
c12	5.434,066	5.362,094	5.369,903	5.306,939	5.333,050
c13	2,430	2,629	2,460	2,405	2,419
c14	640,322	638,584	649,641	653,880	657,836
c15	88,119	94,666	90,021	88,134	87
c16	22,186	20,581	20,743	20,792	20,952
c17	4,698	4,648	5,334	4,821	4,878
c18	4,989	3,544	3,491	3,523	4,551
c19	143,338	143,333	141,387	143,463	143,641

APÊNDICE I – RESULTADO COMPLETO DOS TESTES DO BANCO RDF

CONSULTA	TEMPO DE EXECUÇÃO (ms)				
	PRIMEIRA EXECUÇÃO	SEGUNDA EXECUÇÃO	TERCEIRA EXECUÇÃO	QUARTA EXECUÇÃO	QUINTA EXECUÇÃO
c1	1.247	1.250	1.245	1.197	1.364
c2	1.242	466	297	320	289
c3	6	12	11	10	9
c4	14	17	14	14	15
c5	54	13	9	9	8
c6	4	3	3	3	3
c7	44.702	38.162	38.522	37.796	38.577
c8	10.798	10.787	11.217	11.097	11.032
c9	62	40	36	39	37
c10	9.067	6.450	6.387	6.383	6.515
c11	31	22	20	15	15
c12	7.411	7.716	7.209	7.280	7.792
c13	41	11	9	13	10
c14	18.831	18.760	19.724	18.908	19.139
c15	839	236	239	240	223
c16	119	90	83	80	91
c17	158	128	125	107	117
c18	26	26	24	28	35
c19	258	283	54	410	294

ANEXOS

ANEXO A – CONSULTAS SPARQL INALTERADAS

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX gtfs: <http://vocab.gtfs.org/terms#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX schema: <http://schema.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

#Query 1 - List all shapes with some of their data.
SELECT * WHERE {
    ?shape a gtfs:Shape .
    ?shape geo:lat ?shape_pt_lat .
    ?shape geo:long ?shape_pt_lon .
    ?shape gtfs:pointSequence ?shape_pt_sequence .
}

#Query 2 - List all stops with some of their data including geographic
coordinates, where the latitude is bigger than its mean
SELECT * WHERE {
    ?stop a gtfs:Stop .
    OPTIONAL { ?stop dct:description ?stopDescription . }
    OPTIONAL { ?stop gtfs:wheelchairAccessible ?wheelchairAccesible }
    ?stop geo:lat ?stopLat .
    ?stop geo:long ?stopLong .
    FILTER (?stopLat > %LAT%) .
}

#Query 3 - Find the accessibility information for the stations, if
available
SELECT * WHERE {
    ?stop a gtfs:Stop .
    ?stop gtfs:locationType ?location .
    OPTIONAL { ?stop dct:description ?stopDescription . }
    OPTIONAL { ?stop geo:lat ?stopLat ; geo:long ?stopLong . }
    OPTIONAL { ?stop gtfs:wheelchairAccessible ?wheelchairAccessible . }
    FILTER
(?location=<http://transport.linkeddata.es/resource/LocationType/2>)
}

#Query 4 - List all agencies and their routes with some of their data
SELECT * WHERE {
    ?route a gtfs:Route .
    OPTIONAL { ?route gtfs:shortName ?routeShortName . }
    OPTIONAL { ?route gtfs:longName ?routeLongName . }
    OPTIONAL { ?route dct:description ?routeDescription . }
    ?route gtfs:agency ?agency .
    ?agency a gtfs:Agency .
    ?agency foaf:page ?agencyPage .
    ?agency foaf:name ?agencyName .
    OPTIONAL { ?agency foaf:phone ?agencyPhone . }
}

#Query 5 - Services that have been added on a specific day
SELECT * WHERE {
    ?service a gtfs:Service .
```

```
    ?service gtfs:serviceRule ?serviceRule .
    ?serviceRule a gtfs:CalendarDateRule .
    ?serviceRule dct:date ?date .
    ?serviceRule gtfs:dateAddition " true "^^ xsd:boolean .
    FILTER(?date > %DATE%)
}

#Query 6 - Check the number of routes of a particular agency
SELECT (count(?route) as ?nRoutes) WHERE {
    ?route a gtfs:Route .
    ?route gtfs:agency ?agency .
    FILTER (?agency=%AGENCY%)
}

#Query 7 - List all wheelchair accessible stops along a particular route,
with some of their additional data
SELECT DISTINCT ?routeShortName ?routeDescription ?tripShortName
    ?stopDescription ?stopLat ?stopLong WHERE {
    ?route a gtfs:Route .
    OPTIONAL { ?route gtfs:shortName ?routeShortName . }
    OPTIONAL { ?route dct:description ?routeDescription . }
    ?trip a gtfs:Trip .
    OPTIONAL { ?trip gtfs:shortName ?tripShortName . }
    ?trip gtfs:service ?service .
    ?trip gtfs:route ?route .
    ?stopTime a gtfs:StopTime .
    ?stopTime gtfs:trip ?trip .
    ?stopTime gtfs:stop ?stop .
    ?stop a gtfs:Stop .
    OPTIONAL { ?stop dct:description ?stopDescription . }
    OPTIONAL { ?stop geo:lat ?stopLat ; geo:long ?stopLong . }
    ?stop gtfs:wheelchairAccessible gtfsaccessible:1 .
    FILTER (?route=%ROUTE%)
}

#Query 8 - List the routes and their related trips, services, stops and
stop times with some of their additional data, if available.
SELECT * WHERE {
    ?route a gtfs:Route .
    OPTIONAL { ?route gtfs:shortName ?routeShortName . }
    OPTIONAL { ?route dct:description ?routeDescription . }
    ?trip a gtfs:Trip .
    OPTIONAL { ?trip gtfs:shortName ?tripShortName . }
    ?trip gtfs:service ?service .
    ?trip gtfs:route ?route .
    ?stopTime a gtfs:StopTime .
    ?stopTime gtfs:trip ?trip .
    ?stopTime gtfs:stop ?stop .
    ?stop a gtfs:Stop .
    OPTIONAL {?stop dct:description ?stopDescription . }
    ?service a gtfs:Service .
    ?service gtfs:serviceRule ?serviceRule .
}

#Query 9 - Trips and associated shapes where lat is bigger than its average
and some of their additional data
SELECT * WHERE {
    ?trip a gtfs:Trip .
    OPTIONAL { ?trip gtfs:shortName ?tripShortName . }
    ?trip gtfs:service ?service .
    ?trip gtfs:route ?route .
```

```
    ?trip gtfs:shape ?shape .
    ?shape a gtfs:Shape .
    ?shape geo:lat ?lat .
    FILTER (?lat > %LAT%)
}

#Query 10 - Number of trips that have a duration over 30 minutes
SELECT (count(distinct ?trip) as ?count) WHERE {
    ?trip a gtfs:Trip .
    ?stopTime a gtfs:StopTime .
    ?stopTime gtfs:trip ?trip .
    ?stopTime gtfs:departureTime ?departureTime .
    FILTER (?departureTime >= "00:30:00"^^ xsd:duration)
}

#Query 11 - Trips that are available on a certain date and some of their
additional data
SELECT * WHERE {
    ?service a gtfs:Service .
    ?service gtfs:serviceRule ?calendarRule .
    ?trip gtfs:service ?service .
    ?calendarRule a gtfs:CalendarRule .
    ?calendarRule schema:startDate ?startDate .
    ?calendarRule schema:endDate ?endDate .
    FILTER (?startDate <%DATE%) .
    FILTER (?endDate > %DATE%) .
    FILTER NOT EXISTS {
        ?service gtfs:serviceRule ?calendarDateRule .
        ?calendarDateRule a gtfs:CalendarDateRule .
        ?calendarDateRule dct:date %DATE% .
        ?calendarDateRule gtfs:dateAddition " false "^^ xsd:boolean
    }
}

#Query 12 - Number of stops that are wheelchair-accessible grouped by route
and some of their additional data
SELECT ?longName (count(?name) as ?count) WHERE {
    ?route a gtfs:Route .
    ?route gtfs:longName ?longName .
    ?trip a gtfs:Trip .
    ?trip gtfs:route ?route .
    ?stopTime a gtfs:StopTime .
    ?stopTime gtfs:trip ?trip .
    ?stopTime gtfs:stop ?stop .
    ?stop a gtfs:Stop .
    ?stop foaf:name ?name .
    ?stop gtfs:wheelchairAccessible gtfsaccessible:1 .
}
GROUP BY ?longName

#Query 13 - All the accesses of the stations
SELECT * WHERE {
    ?stop a gtfs:Stop .
    ?stop gtfs:parentStation ?parStation .
    OPTIONAL {?stop foaf:name ?accName} .
    ?stop gtfs:locationType gtfslocation:2 .
    ?parStation a gtfs:Stop .
    ?parStation foaf:name ?name
}
```

```
#Query 14 - All stops times and their related routes and stops order by
their sequence
SELECT * WHERE {
    ?stopTime a gtfs:StopTime .
    ?stopTime gtfs:trip ?trip .
    ?stopTime gtfs:stop ?stop .
    ?stopTime gtfs:stopSequence ?sequence .
    ?stop a gtfs:Stop .
    ?trip a gtfs:Trip .
    ?trip gtfs:route ?route .
    OPTIONAL {?stop foaf:name ?stopName}
} ORDER BY ?sequence

#Query 15 - Everything that contains a specific string in the object
placeholder (any property)
SELECT * WHERE {
    ?stop a gtfs:Stop .
    ?stop ?p ?str .
    FILTER regex (?str, %STRING%)
}

#Query 16 - For all the routes, all the calendar changes during a specific
month
SELECT * WHERE {
    ?trip a gtfs:Trip .
    ?trip gtfs:service ?service .
    ?trip gtfs:route ?route .
    ?service a gtfs:Service .
    ?service gtfs:serviceRule ?serviceRule .
    ?serviceRule a gtfs:CalendarDateRule .
    ?serviceRule dct:date ?servDate .
    ?serviceRule gtfs:dateAddition " true "^^ xsd:boolean .
    FILTER (?servDate >= %DATE1%) .
    FILTER (?servDate <= '%DATE2%) .
}

#Query 17 - Trips with their start and end time of the frequencies and
associated routes
SELECT ?routeName ?routeType ?trip ?startTime ?endTime WHERE {
    ?trip a gtfs:Trip .
    ?trip gtfs:route ?route .
    ?frequency a gtfs:Frequency .
    ?frequency gtfs:startTime ?startTime .
    ?frequency gtfs:endTime ?endTime .
    ?frequency gtfs:trip ?trip .
    ?route a gtfs:Route .
    ?route gtfs:shortName ?routeName .
    ?route gtfs:routeType ?routeType .
}

#Query 18 - All routes that have trips on Sunday
SELECT * WHERE {
    ?service a gtfs:Service .
    ?service gtfs:serviceRule ?serviceRule .
    ?serviceRule a gtfs:CalendarRule .
    ?serviceRule gtfs:sunday " true "^^ xsd:boolean .
    ?trip gtfs:service ?service .
    ?trip gtfs:route ?route .
    { ?route gtfs:longName ?longName } UNION
    { ?route gtfs:shortName ?shortName } .
}
```

Fonte: (Chaves-Fraga et al., 2020)