Avaliação de Processos ETL para Análise de Dados usando SGBD Orientado a Grafos

Jones Dhyemison Quito de Oliveira, Leonardo Andrade Ribeiro

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Goiânia – GO – Brasil

{jonesoliveira,laribeiro}@inf.ufg.br

Abstract. *The presence of duplicates is a perennial problem in databases. This type of inconsistency violates integrity constraints and may compromise the results of data analysis activities. A graph-oriented DBMS can be used to perform similarity graph queries and, thus, identify potential duplicates. This approach requires the execution of an ETL process for extracting data from relational sources, transforming them into a similarity graph, and loading this graph into a graph-oriented DBMS. This paper presents a performance comparison between two ETL processes for this purpose. The first process performs the calculation of similarities using the relational DBMS itself. The second process performs the calculation of similarities using a specialized algorithm. The results show that the use of the specialized algorithm outperforms the approach based on technology purely relational by orders of magnitude.*

Resumo. *A presença de duplicatas é um problema perene em bancos de dados. Esse tipo de inconsistência viola restrições de integridade e pode comprometer o resultado de atividades de análise de dados. Um SGBD orientado a grafos pode ser usado para execução de consultas sobre um grafo de similaridade para identificação de possíveis duplicatas. Essa abordagem requer a execução de um processo ETL para extração de dados de fontes relacionais, transformação deles em um grafo de similaridade, e carga deste grafo em um SGBD orientado a grafos. Este trabalho apresenta uma comparação de desempenho entre dois processos ETL para este fim. O primeiro processo realiza o cálculo de similaridades usando o próprio SBGD relacional. O segundo processo realiza o cálculo de similaridades usando um algoritmo especializado. Os resultados obtidos mostram que uso do algoritmo especializado supera a abordagem baseada em tecnologia puramente relacional em ordens de magnitude.*

1. Introdução

Bancos de dados relacionais são popularmente usados para persistir dados produzidos e usados por aplicações. Ao longo dos anos, a quantidade e o volume desses bancos de dados têm aumentado consideravelmente. Neste contexto, é comum a ausência de critérios e padrões para ingestão de dados, causando o surgimento de várias inconsistências [Hernández and Stolfo 1998]. Uma inconsistência muito comum é a presença das chamadas duplicatas, isto é, múltiplas representações para um mesmo objeto do mundo real [Elmagarmid et al. 2007]. Duplicatas podem comprometer operações rotineiras de recuperação e análise de dados, fazendo com que o resultado dessas operações seja também inconsistente.

Um dos primeiros passos para enfrentar o problema exposto é a identificação dos objetos de dados que representam possíveis duplicatas. Para essa finalidade, o uso de algoritmos de similaridade têm se mostrado muito útil [Chaudhuri et al. 2006]. Pode-se assumir que objetos que possuem entre si um valor de similaridade superior a um limiar pré-determinado sejam considerados como possíveis duplicatas e selecionados para uma análise posterior mais criteriosa.

Algumas propostas defendem que os dados duplicados sejam unificados em uma única representação em uma operação chamada de fusão de dados [Dong and Naumann 2009]. No entanto, essa abordagem pode conduzir à perda de informação, pois unificar dados que sejam considerados como possíveis duplicatas implica em eliminar cópias e preservar apenas uma representação. Com isso, caso um conjunto de dados seja equivocadamente classificado como duplicatas, será difícil reverter a operação de fusão.

Uma alternativa para a abordagem acima é manter as possíveis duplicatas no banco de dados e modelar a relação de similaridade entre as mesmas através de um grafo de similaridade [Vaz et al. 2019]. Neste contexto, vértices representam registros e arestas representam a similaridade entre pares de registros. Novas informações podem ser incorporadas através da inserção de novos vértices e atualização do valor associado a arestas. Sistemas gerenciadores de banco de dados orientados a grafos (SGBDGs) podem ser usados para gerenciamento e execução de consultas sobre o grafo de similaridade. Esse gerenciamento permite que o analista de dados decida se é conveniente tratar os similares como duplicatas ou se é mais interessante explorar a similaridade existente entre as possíveis duplicatas para produzir análises mais flexíveis.

O processo ETL é atividade crucial neste contexto para extração dos dados, transformação dos mesmos em um grafo de similaridade e carga em um SGBD orientado a grafos — ETL é um acrônimo do inglês *Extract, Transform, Load*. Este trabalho apresenta uma comparação entre duas propostas para implementação de um processo ETL para esse fim. A primeira proposta, chamada (*SimDataMapper*)[Ribeiro et al. 2016], utiliza a própria infraestrutura do SGBD relacional onde os dados residem para realizar os cálculos de similaridade; a segunda proposta, chamada *MPJoin* [Ribeiro and Härder 2011], é baseada em um algoritmo especializado, que é executado de maneira *standalone*, em área de memória separada do SGBD. Ambas propostas realizam uma junção por similaridade entre os objetos textuais, baseando-se no conceito de similaridade de conjuntos. Após o processo de extração de similaridade, os dados e suas respectivas similaridades são carregados em um SGBD orientado a grafos para exploração de consultas posteriores. Nesse trabalho, foi escolhido o SGBDG Neo4j [1] devido à sua popularidade.

O restante deste artigo está organizado como segue. Na Seção 2 são apresentados os fundamentos teóricos que permeiam o trabalho. Na Seção 3 são discutidos os trabalhos relacionados. Os processos ETL comparados neste trabalho são apresentados na Seção 4. Os experimentos realizados são descritos e os resultados analisados na Seção 5. Finalmente, as conclusões são apresentadas na Seção 6.

[1] https://neo4j.com/

2. Fundamentação teórica

O termo similaridade é definido como ”Característica, estado ou natureza do que é similar; semelhança”[Aurélio 2019], ou seja são objetos ou coisas que apresentam muitas características em comum. Neste trabalho, quando houver referência ao termo similaridade de string ou similaridade entre cadeias de caracteres, trata-se do quão semelhante são duas strings distintas, isto é, a proporção de atributos comuns entre essas strings.

2.1. Funções de similaridade

Atualmente existem várias abordagens para calcular a similaridade textual. A distância de edição e a similaridade de *Jaccard* são exemplos dessas abordagens. A distância de edição é definida como o menor número de operações de edição (inserção, exclusão e substituição) necessária para transformar uma string A em uma string B [Navarro 2001]. Outra abordagem muito utilizada para calcular a similaridade entre duas strings, é a similaridade com base na sobreposição de conjuntos derivados das strings sobre as quais deseja-se calcular a similaridade. Os itens desses conjuntos são chamados de *tokens*. Este trabalho terá foco em similaridade baseada em sobreposição de conjuntos. O cálculo da sobreposição de conjuntos é computacionalmente menos oneroso que a distância de edição e foi demonstrado em avaliações empíricas que as duas técnicas possuem eficácia comparável [Cohen et al. 2003].

Funções de similaridade baseadas em conjuntos de tokens necessitam de uma etapa de transformação das cadeias de caracteres. Uma técnica popular para mapear uma string para um conjuntoa de tokens é baseada no conceito de *q-grams*. Dada uma string, os *q-grams* correspondentes são substrings de comprimento q obtidos através do deslizamento de uma janela de comprimento q sobre essa string [Ukkonen 1992]. A ideia é que dada uma string **S**, ao deslizar uma janela de tamanho q sobre essa string **S**, obtemos um conjunto de substrings de tamanho q. No entanto, o deslizamento dessa janela nos primeiros e nos últimos caracteres vai gerar tokens de tamanho inferior a q, uma solução para esse problema é adicionar como prefixo a ocorrência de q - 1 caracteres especiais não pertencentes à string **S**, e adicionar a mesma quantidade de caracteres especiais como sufixo de **S**. Essa adição de prefixos e sufixos assegura que todo *q-gram* contenha q caracteres e que todos caracteres estejam presentes em exatamente q *q-grams*.

Exemplo 1 *Considere a string “Gabriel”. Para gerar um conjunto de q-gram de tamanho 3 (3-grams) a partir dessa string, a mesma é acrescida com 2 ocorrências de um caractere especial (**#**) ,que não ocorra em qualquer string do banco de dados, no inicio e no fim da mesma. Dessa forma tem-se a seguinte cadeia derivada (**##Gabriel##**). Agora, ao deslizar-se uma janela de tamanho 3 sobre a string S tem-se como primeiro 3-gram os caracteres **##G**. Ao deslizar a janela sobre o resto da string, obtem-se o seguinte conjunto de 3-grams: {**#Ga**, **Gab**, **abr**, **bri**, **rie**, **iel**, **el#**, **l##**}. Dessa forma obtem-se um conjunto de tokens de tamanho q=3.*

2.2. Esquema de Ponderação

Uma característica importante ao se trabalhar com algoritmos de similaridade baseados em sobreposição de conjuntos é a noção de importância dos tokens ou *q-grams*. A cada token pode-se associar um peso, resultando em conjuntos ponderados [Ribeiro et al. 2016]. No contexto deste trabalho, trata-se de atribuir pesos para os *q-grams* de forma que os tokens mais relevantes para determinar a similaridade entre strings recebam pesos maiores.

Dessa forma, quando os tokens mais relevantes pertencerem à intersecção dos conjuntos, a similaridade resultante é mais significativa.

Um esquema de ponderação bastante popular é o *Inverse Document Frequency* (IDF), usado originalmente no contexto de Recuperação de Informação [Baeza-Yates and Ribeiro-Neto 2011]. A ideia é que os tokens que aparecem com menor frequência em um corpus de palavras (ou documentos) são mais significativos e, portanto têm mais importância na avaliação da similaridade. Nessa abordagem, quando se deseja trabalhar com conjuntos não ponderados, basta atribuir o peso 1 para todos os tokens.

2.3. Similaridade Jaccard

A similaridade de Jaccard, por sua vez, faz uso da ideia de conjuntos. Dadas duas strings A e B, os conjuntos de substring derivados de A e B são comparados em termos da intersecção e união dos mesmos. Resumidamente, pode-se afirmar que a similaridade de Jaccard é o quociente da intersecção pela união dos conjuntos derivados de A e B.

Definição 1 *Seja A e B duas strings e denote-se por* $Q_q(A)$ *e* $Q_q(B)$ *os conjuntos de q-grams derivados das mesmas. A similaridade Jaccard entre A e B é definida por*

$$Jaccard(A,B) = \frac{|Q_q(A) \cup Q_q(B)|}{|Q_q(A) \cap Q_q(B)|}$$

Exemplo 2 *Considere as strings* $A = "Gabriel"$ *e* $B = "Gabiel"$ *(a segunda contém um erro de digitação). Os conjuntos de 3-grams derivados dessas strings são:*

$Q_3(A)$=$\{$***#Ga***, ***Gab***, *abr, bri, rie,* ***iel***, ***el#***, ***l##***$\}$ *e*
$Q_3(B)$=$\{$***#Ga***, ***Gab***, *abi, bie,* ***iel***, ***el#***, ***l##***$\}$

Tem-se $|Q_3(A)| = 8$, $|Q_3(B)| = 7$ *e* $|Q_q(A) \cap Q_q(B)| = 5$. *Portanto,* $Jaccard(A,B) = \frac{5}{8+7-5} = \frac{5}{10} = 0.5$.

2.4. Junção por Similaridade

Os algoritmos de similaridade de Jaccard sobre objetos do tipo texto são amplamente utilizados em seleção por similaridade, onde uma string de busca e um limite são passados como parâmetros para o algoritmo, e toda string que apresenta similaridade igual ou superior ao limite informado é retornada como resultado da busca. A junção por similaridade é outra aplicação muito comum para algoritmos de similaridade de objetos de texto. Essa operação retorna pares de tuplas similares de duas tabelas de entrada.

Definição 2 *Dadas duas tabelas* ***T1*** *e* ***T2*** *com atributos do tipo string* ***T1.Ai*** *e* ***T2.Aj****, respectivamente, uma função de similaridade* ***Sim*** *e um limiar de corte* ***lim****. Uma junção por similaride entre* ***T1*** *e* ***T2*** *retorna todos os pares de tuplas (t1, t2) pertencentes a* ***T1*** *x* ***T2*** *tal que Sim(****t1.Ai, t2.Aj****) é maior ou igual* ***lim****.*

2.5. Gerenciamento de Duplicatas usando SGBDGs

As técnicas de junção por similaridade possibilitam formar conjuntos de objetos que apresentam alta similaridade entre si. Outro uso de junções por similaridade é encontrar, em tabelas distintas, valores que possivelmente representam um mesmo objeto de dados. Os pares no resultado de uma junção por similaridade podem ser interpretados como

possíveis duplicatas. Com base nessa interpretação pode ser desejável gerenciar estas possíveis duplicatas de modo que haja apenas uma representação desses dados, evitando assim, ambiguidades e resultados inconsistentes em operações de análise de dados por exemplo. No entanto não há garantia de que dados que apresentam alta similaridade entre si correspondam de fato a um único objeto do mundo real. É perfeitamente possível que em uma base de dados exista mais de uma ocorrência do nome *"João da Silva"* e que de fato elas representam entidades distintas no mundo real. Dessa forma, a abordagem de eliminar possíveis duplicatas baseado apenas na similaridade apresentada entre os atributos textuais desses objetos não é prudente. Uma alternativa é gerenciar as duplicatas usando um SGBDG.

O Neo4j é atualmente o mais popular SGBDG. Diferente outros SGBDs NoSQL, o Neo4j implementa as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade). Os dados são organizados em forma de grafo de propriedades. A composição básica de um banco de dados no Neo4j são os nós (vértices) que representam as entidades de dados, e as arestas que representam os relacionamentos entre os nós. Tanto os nós como seus relacionamentos podem possuir propriedades que são compostas de nome e valor. Além das propriedades, os nós e seus relacionamentos podem possuir rótulos que determinam um tipo ou domínio. A cardinalidade de rótulos para um nó é de zero para vários, ou seja, um nó pode possuir nenhum ou vários rótulos, ao passo que um relacionamento pode possuir no máximo um rótulo.

Usuários e programas de aplicação interagem com os dados através da linguagem de consulta do Neo4j, o Cypher. Trata-se de uma linguagem fortemente baseada em padrões, que são usados para corresponder às estruturas desejadas do grafo. Uma vez que uma estrutura correspondente tenha sido encontrada ou criada, o Neo4j pode usá-la para processamento adicional.

3. Trabalhos relacionados

Vários trabalhos relacionados à similaridade de strings têm sido publicados nos últimos anos. Alguns desses trabalhos tem focado em algoritmos ad-hoc desenvolvidos em SQL como forma de estender as funcionalidades de SGBDs relacionais, outros têm implementado os algoritmos de similaridade em linguagens de programação imperativa.

O *SimDataMapper* [Ribeiro et al. 2016] descreve um arcabouço que disponibiliza um conjunto de funcionalidades relacionadas à junção e seleção por similaridade para aplicações desenvolvidas no paradigma de orientação a objetos. O referido arcabouço abstrai a complexidade necessária para gerenciar as estruturas criadas no lado do SGBD que são usadas para suportar as operações de similaridade na infraestrutura de um banco de dados relacional.

[Ribeiro and Härder 2011] propõem um algoritmo baseado em índice invertido para junção de conjuntos por similaridade. Com foco na redução do custo computacional durante geração de candidatos, eles apresentam o conceito de *min-prefix* o qual tem como objetivo contribuir com redução do tempo de geração de candidatos. *Min-prefix* é uma generalização do conceito de filtragem de prefixo proposto em [Chaudhuri et al. 2006].

[Gruenheid et al. 2014] apresentam um arcabouço para identificação de duplicatas de maneira incremental. A solução proposta mantém um grafo de similaridade, que é atu-

alizado quando o banco de dados é modificado. O gerenciamento do grafo de similaridade usando um SGBDG não foi considerado.

[Vaz et al. 2019] faz uso de técnicas de junção por similaridade para identificação de indícios de fraudes em processos licitatórios. Nessa abordagem a gerencia dos dados duplicados não implica na unificação das duplicatas, mas no gerenciamento das mesmas em um SGBDG. Ele faz uso da ideia de gerenciamento de duplicatas com auxílio de um SGBD orientado a grafos. O uso de um SGBDG para identificação de fraudes em processos licitatórios havia sido apresentado por Erven et al [van Erven 2015] em trabalho anterior; entretanto operações de similaridade não foram consideradas.

4. Processos ETL

No contexto deste trabalho, junção por similaridade é a principal operação do processo ETL, corresponde à etapa de transformação. Esta seção apresentará duas propostas de processo ETL baseadas em diferentes implementações dessa operação. O resultado da junção por similaridade em cada proposta é um conjunto de tuplas compostas de três (3) campos, a saber: dois identificadores únicos de dois objetos e um valor que expressa a similaridade entre eles. Para facilitar o processo de carga dos dados no Neo4j, os dados de saída são persistidos em arquivo CSV.

4.1. ETL usando o *SimDataMapper*

O *SimDataMapper* é apresentado como um padrão arquitetural que faz uma ponte entre aplicativos desenvolvidos no paradigma de orientação a objetos e sistemas de bancos de dados relacionais [Ribeiro et al. 2016]. Basicamente ele faz um mapeamento de objetos de memória para tabelas em bases de dados relacionais. O referido padrão funciona como uma camada de software que isola em memória os objetos de domínio e o banco de dados e lida com a transferência de dados entre eles. No entanto, diferentemente de outras propostas que fornecem esse mapeamento objeto relacional, o *SimDataMapper* oferece o suporte a operações de busca e junção por similaridade.

O *SimDataMapper* implementa operações de similaridade baseada em distância de edição e em conjuntos de tokens, sendo que no último caso é possível usar conjuntos ponderados ou não ponderados. Em um processo de junção por similaridade, por exemplo, as etapas básicas para recuperação dos valores similares é converter as strings alvo do calculo de similaridade em conjuntos de tokens, criar tabelas auxiliares para armazenar os tokens resultantes e armazenar a frequência com que os tokens ocorrem no conjunto de strings considerados como alvo da operação de similaridade. O *SimDataMapper* dispõe de um método chamado *EnableSim* que recebe a especificação de um atributo do tipo string T.A de uma tabela T e a partir desse atributo ele cria uma tabela de tokens e vários índices, de acordo com o estágio selecionado usando o método *SetStage*. A Figura 1 ilustra a interação entre o SGBDR e o *SimDataMapper*, juntamente com o respectivo fluxo de dados resultante dessa interação.

Após a criação das estruturas necessárias para habilitar os cálculos de similaridade, a operação de junção por similaridade pode ser invocada. Nesta etapa, um gerenciador de expressões SQL é usado para gerar as expressões SQL correspondentes e enviá-las ao SGBDR. O resultado da junção por similaridade é escrito em um arquivo de texto. O *SimDataMapper* realiza todo o processamento da operação de similaridade usando o motor de execução de consultas do SGBDR.

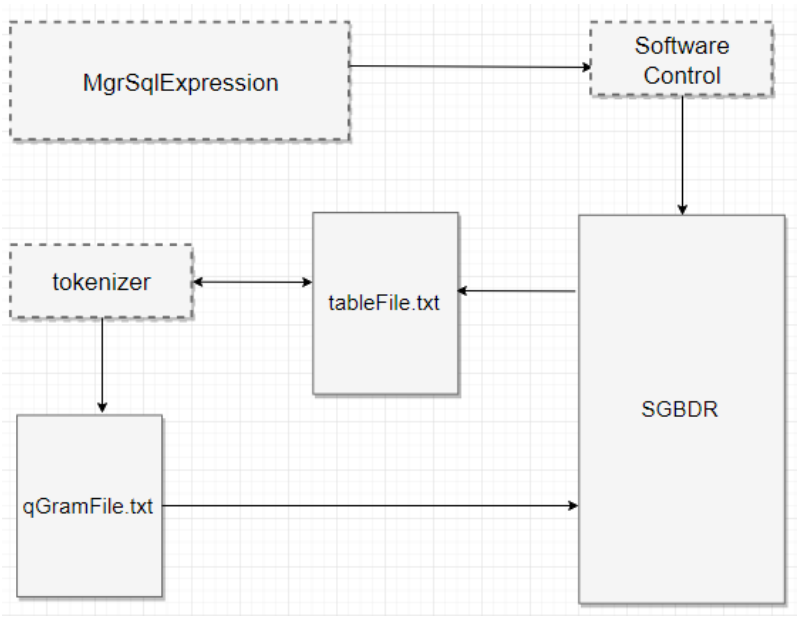

Figura 1. Fluxo de dados do *SimDataMapper*.

4.2. ETL usando o *MPJoin*

O *MPJoin* é um algoritmo especializado para junção por similaridade. Ele se distingue do *SimDataMapper* principalmente por não fazer uso da infraestrutura do SGBDR relacional para realizar as operações de similaridade. No *MPJoin*, todo o processo de junção por similaridade é realizado na sua própria área de memória.

O *MPJoin* distribui o processo de junção por similaridade em três etapas. A primeira etapa, chamada *XMLToText*, consiste da extração dos dados textuais de uma base indicada no arquivo de configuração usado pelo *MPJoin*. Na segunda etapa, chamada *CreateSSFile*, as strings são mapeadas para conjuntos, opcionalmente ponderados, e a coleção de conjuntos resultante é escrita em um arquivo binário. Por fim, na terceira etapa, chamada `SimJoin`, a junção de similaridade é realizada; a saída é composta pelos identificadores das strings e os respectivos valores de similaridade entre elas.

Na sua forma original, o *MPJoin* busca os dados em uma base XML, extraindo o atributo que foi definido como alvo para o calculo de similaridade e armazenando esses dados em um arquivo que será usado como entrada na etapa seguinte do processo. No entanto, para adaptá-lo à necessidades de junção por similaridade com dados armazenados de uma base de dados relacional, foi desenvolvida uma alternativa ao `xmlToText` denominada `relationalExtractor`. O `relationalExtractor`, como o próprio nome sugere, extrai os dados de uma base relacional e os disponibiliza para a Etapa 2 do *MPJoin*. A Figura 2 ilustra o processo de junção por similaridade do *MPJoin*, mostrando o fluxo de dados no decorrer do processo.

4.3. Carga dos Dados de Similaridade no Neo4j

Neste trabalho, o processo de carga de dados no Neo4j consiste em duas etapas. A primeira é garantir que a conjunto de dados, sobre a qual se deseja gerenciar as possíveis duplicatas, seja carregado no banco. Para essa tarefa, pode-se usar o `LOAD CSV`, um utilitário do Neo4j para carga massiva de dados. Na segunda etapa fazemos a carga dos

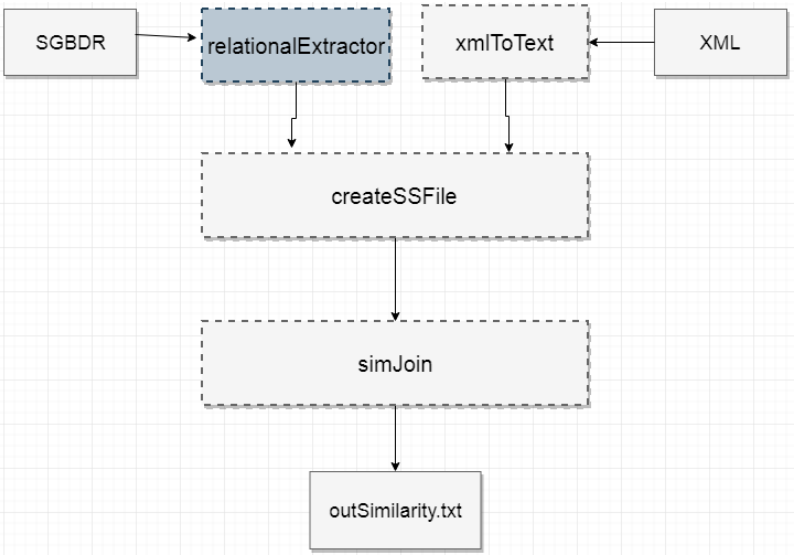

Figura 2. Fluxo de dados do *MPJoin*.

valores de similaridade associando cada nó a seus respectivos nós similares. Nesta etapa, são usados os dados de similaridade obtidos com o uso do *SimDataMapper* ou *MPJoin*. A Figura 3 ilustra o processo de carga de dados no Neo4j.

Na Etapa 2, para cada registro de entrada no `LOAD CSV`, o Neo4j realiza uma busca para encontrar os objetos que correspondem aos identificadores lidos no arquivo de entrada e então cria um relacionamento entre os objetos atribuindo um nome e um atributo a este relacionamento. Os relacionamentos criados têm como propriedade o valor da similaridade entre os objetos relacionados. Dessa forma, é possível o desenvolvimento de consultas que explorem o valor de similaridade entre objetos. Os scripts usados para realizar a carga dos dados no Neo4j são apresentados abaixo.

Carga da base de dados de publicações:

```
USING PERIODIC COMMIT 100
load csv with headers from
”file:///F:/INICIACAO_CIENTIFICA/importNeo4j/publicacoes.txt”as row
create(:Autortid:toInteger(row.tid), nome: row.nome, obra: row.obra)
```

Criando índice:

```
CREATE INDEX ON :Autor(tid);
```

Relacionando nós similares:

```
load csv with headers from
”file:///F:/INICIACAO_CIENTIFICA/importNeo4j/saida.txt”as row
match(a1:Autortid:toInteger(row.Id_01))
match(a2:Autortid:toInteger(row.Id_02))
where a1.tid <> a2.tid
merge(a1)-[sim:SIMILARIDADE]-(a2)
ON CREATE SET sim.similaridade = tofloat(row.sim);
```

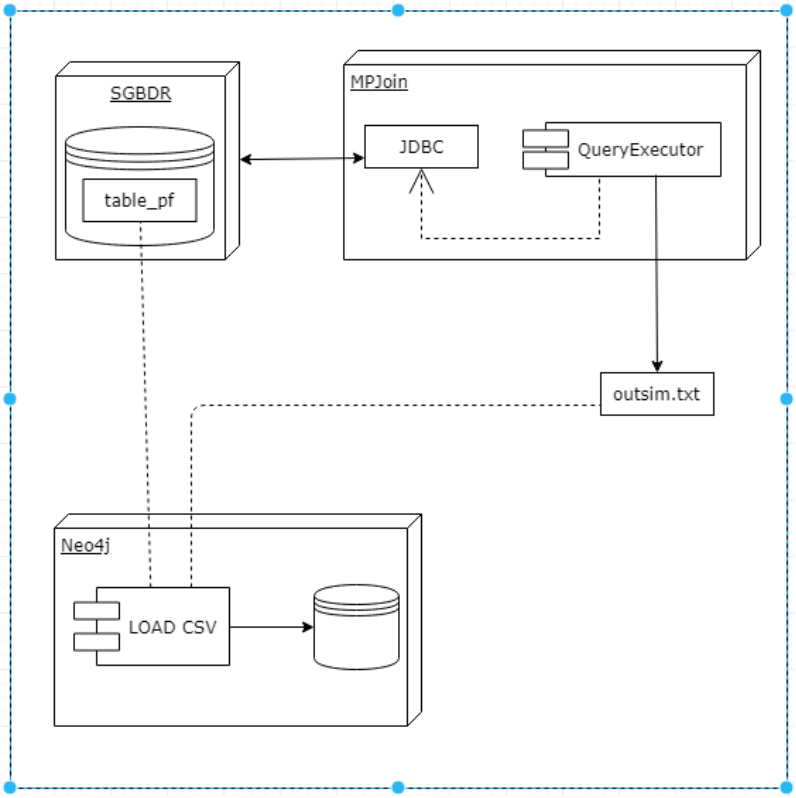

Figura 3. Visão geral do processo de ETL.

4.4. Consultas por Similaridade no Neo4J

Para explorar a aresta de similaridade, foram especificadas algumas consultas simples, mas que ilustram como o valor de similaridade entre objetos pode ser usado para gerenciar duplicatas.

Consulta 1:
match(a1:Autor) -[sim:SIMILARIDADE]- (a2:Autor)
where sim.similaridade $>= 0.8$ return *

Consulta 2:
match(a1:Autor) -[sim:SIMILARIDADE]- (a2:Autor)
where sim.similaridade $>= 0.8$
and sim.similaridade < 1.0 return *

Consulta 3:
match(a1:Autor) -[sim:SIMILARIDADE]- (a2:Autor)
where sim.similaridade $>= 0.7$
and sim.similaridade < 1.0
and a1.obra $=$ a2.obra
return * limit 30

A Consulta 1 retorna todos os nós que possuem relacionamento com a propriedade [SIMILARIDADE] maior ou igual a 0.8.

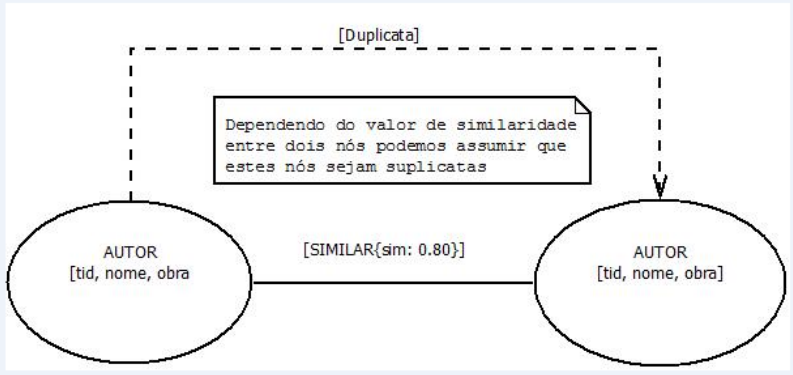

Figura 4. Grafo de similaridade para identificação de duplicatas.

Na Consulta 2 também é feita uma restrição da consulta com base no valor da similaridade entre os nós, a diferença é que agora é usada uma condição a mais na similaridade, restringindo também a similaridade máxima, fazendo com que apenas os nós que satisfaçam ambas condições sejam retornados.

A Consulta 3 busca todos os pares de autores cujos nomes tem similaridade maior ou igual a 0.7. Mas diferente das Consultas 1 e 2, ela usa um segundo atributo para reforçar a ideia de duplicidade. É possível observar que a segunda verificação, realizada no atributo `Autor.obra`, não faz uso de correspondência por similaridade. No entanto, esse atributo também é do tipo texto e, portanto é passível de ser registrado na base com alguma inconsistência que pode gerar falsos negativos deixando de retornar possíveis duplicatas que estão representadas de formas distintas devido a um erro de digitação, por exemplo.

A Consulta 3 é muito útil para ilustrar a importância de gerenciar as possíveis duplicatas em vez de simplesmente unificá-las. Isso foi percebido quando alimentado o limiar de similaridade para um valor maior que 0.8. Essa configuração não retornou resultados, ou seja, a consulta considerou que não há autores com nome muito semelhante e que tenham publicado obras com o mesmo nome. No entanto, ao considerar limiares de corte iguais ou inferiores a 0.8, observa-se que existem autores com semelhança de nome de até 0.8 que publicaram obras com o mesmo nome. Essa observação aumenta a confiança de que os registros desses autores são, de fato, duplicatas.

Usando o grafo de propriedades do Neo4j poderíamos representar a semelhança entre objetos com base na similaridade entre múltiplos atributos de texto dos objetos. Para tal, seria suficiente adicionar mais propriedades na aresta de relacionamento, de modo a representar a similaridade entre os demais atributos dos objetos relacionados, conforme mostra a Figura 4.

5. Experimentos

Esta seção apresenta uma avaliação comparativa de desempenho entre os processos ETL descritos na seção anterior.

5.1. Configuração dos Experimentos

Para a realização dos experimentos foram utilizadas as seguintes configurações de hardware e software:

MÉDIA VALORES REAIS EM SEGUNDOS		
Configurações	SDM	MPJoin
	Tempo Médio	Tempo Médio
auth 1000 reg.	1,567333333	0,062
auth 10000 reg.	155,1746667	1,609
auth 100000 reg.	Valor indefinido	880,093

Figura 5. Tempos de execução sobre a base de publicações.

- Sistema Operacional: Linux Ubuntu 18.04.3 LTS.
- Processador: Intel(R) Xeon(R) CPU E31240 @ 3.30GHz.
- Arquitetura: x86_64.
- Socket: 1.
- Core per socket: 4.
- Thread per core: 2.
- SGBDR: Postgres 10 - configurações padrão.
- Neo4j Community-3.5.4.

Os experimentos com os algoritmos de similaridade foram realizados com dois conjuntos de dados distintos, um de pessoas físicas e outro de publicações, com 100 mil registros em cada. O primeiro é composto por um campo de identificação do tipo inteiro e um campo de nome do tipo texto. O segundo conjunto de dados contém informações sobre autores e suas respectivas publicações com a seguinte estrutura: (id: inteiro; nome: texto; obra: texto)

Para possibilitar a comparação de desempenho entre o *SimDataMapper* e *MPJoin* com diferentes volumes de dados, foram criadas tabelas com três quantidades diferentes de registros para cada dataset: 1000 (1k), 10000 (10k) e 100000 (100k) registros. Para cada uma das configurações de volume de dados, o *MPJoin* e o *SimDataMapper* foram executados 5 vezes cada. Dos cinco valores de tempo de execução coletados, foram descartados o maior e o menor valor. Os três valores intermediários restantes foram usados para calculo da média de tempo de execução. Em ambos os sistemas de calculo de similaridade foram usadas as implementações de junção por similaridade de Jaccard não ponderada. Finalmente, foi definido um limite de duas horas para cada execução; execuções que excederam este limite foram abortadas.

5.2. Resultados e Discussão

O *MPJoin* concluiu com sucesso para as três configurações de volume de dados nos dois datasets usados, apresentando melhor desempenho que o *SimDataMapper* em todos cenários. O *SimDataMapper* apresentou resultados apenas para as configurações de 1000 e 10000 registros em ambos os conjuntos de dados, na configuração de 100k ficou executando por mais de duas horas sem concluir o processamento. Os resultados obtidos nos bancos de dados de publicações e pessoas físicas são sumarizados nas Figuras 5 e 6, respectivamente — o *SimDataMapper* encontra-se abreviado para SDM.

Executando sobre a base de dados de pessoa física com 1k registros, o *MPJoin* demorou em média 0,041 segundos. Na configuração de 10k registros da tabela de pessoa física a duração média foi de 0,392. Por fim, na configuração de 100k registros da tabela de pessoa física o tempo médio de execução foi de 15,886. Quando executado

MÉDIA VALORES REAIS EM SEGUNDOS		
Configurações	SDM(segundos)	MPJoin (segundos)
	Tempo Médio	Tempo Médio
pf 1000 reg.	8,319666667	0,041
pf 10000 reg.	1765,795333	0,392
pf 100000 reg.	Valor indefinido	15,886

Figura 6. Tempos de execução sobre a base de pessoas físicas.

sobre a base de publicações o *MPJoin* apresentou os seguintes resultados: 0,062 segundos na configuração de 1k registros, 1,609 segundos executando sobre 10k registros de publicações e por fim 880,093 segundos operando junção por similaridade sobre 100k registros de publicações.

O *SimDataMapper*, usando o conjunto de dados de publicações, apresentou performance inferior nas três configurações de volume de dados. Os resultados do *SimDataMapper* sobre a base de publicações ficou como descrito abaixo. Na primeira configuração, com 1k registros, o tempo médio para concluir a operação foi de 1,567 segundos. Na segunda configuração, com 100k registros o tempo médio foi de 155,174 segundos. Na configuração de 100k, a operação de junção foi finalizada após duas horas de execução sem concluir o processamento.

Ainda sobre o *SimDataMapper*, porém agora executando junções por similaridade na base de dados de pessoas físicas, obtivemos os seguintes resultados: Na configuração de 1k registros o tempo médio para concluir a junção foi de 8,319 segundos. Na configuração de 100k mil registros o tempo médio de execução foi 1765,795 segundos e, por fim, com 100k registros da base de pessoa físicas, o *SimDataMapper* ficou executando por 2 horas sem concluir o processamento até ser intencionalmente finalizado.

Observa-se que o *SimDataMapper* não finalizou o processamento dentro do limite estipulado sobre 100k registros. Além disso, o *MPJoin* supera o *SimDataMapper* em duas ordens de magnitude em todos os demais cenários. O principal motivo para esta notável diferença de desempenho é que o emprego do índice invertido feito por algoritmos especializados como o *MPJoin* permite um conjunto maior otimizações para descartar pares de conjuntos sem a necessidade da realização do cálculo de similaridade. Em particular, a técnica conhecida como *min-prefix* usada pelo *MPJoin* permite remover dinamicamente conjuntos do índice invertido durante a execução do algoritmo. Como o índice invertido é usado para identificar pares de conjuntos para comparações de similaridade, a remoção desses conjuntos resulta em uma redução substancial na quantidade de comparações. Note que um conjunto é removido do índice apenas quando o mesmo, garantidamente, não pode ser similar a nenhum outro conjunto ainda não processado; portanto, a exatidão do *MPJoin* é preservada, isto é, nenhum resultado válido é perdido.

Também foi possível observar que o *SimDataMapper* apresentou performance melhor quando executado sobre a base de publicações, sugerindo que as características do conjunto de dados tem influência sobre a performance do mesmo. Esse ganho de performance pode está relacionado ao fato de a base de publicações possuir menor número de registros similares, gerando um número menor de candidatos para comparação. Por sua vez, o *MPJoin* teve degradação da performance quando executou sobre a base de publicações apresentando comportamento inverso ao *SimDataMapper*.

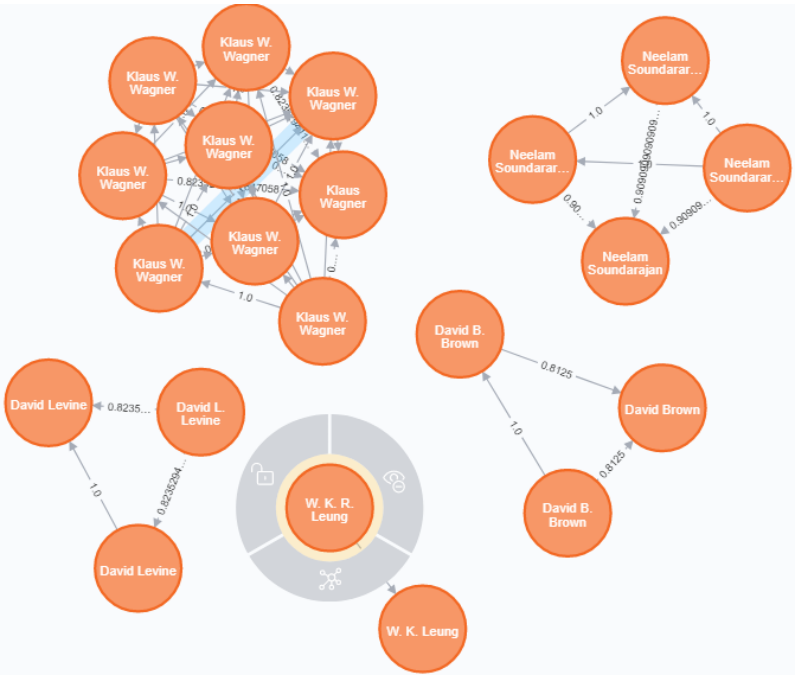

Figura 7. Resultado da Consulta 2.

Finalmente, a Figura 7 ilustra o resultado da Consulta 2 apresentada na Seção 4.4 deste documento. Ela mostra o agrupamento de objetos cuja similaridade entre si é maior ou igual a 0.8 e menor que 1.

6. Conclusão

Este artigo apresentou uma comparação entre processos ETL para gerenciamento de duplicatas baseado em SGBDGs. Foram comparadas duas abordagens que implementam junções por similaridade sobre objetos do tipo texto, o *SimDataMapper* e o *MPJoin*. A primeira abordagem realiza a junção por similaridade usando a infraestrutura do SGBD relacional onde os dados residem, ao passo que a segunda é baseada em um algoritmo especializado. Os resultados dos experimentos demonstraram que o emprego de um algoritmo especializado supera a abordagem mais genérica baseada em tecnologia puramente relacional em ordens de magnitude.

Quanto ao gerenciamento de duplicatas usando SGBDG Neo4j, ressalta-se que o uso do atributo de similaridade permite consultas mais flexíveis, conferindo autonomia para o analista de dados decidir se é relevante considerar determinados valores como duplicatas ou apenas como semelhantes. O atributo de similaridade também permite um refinamento nas consultas para determinar qual limiar de similaridade deve ser usado para tratar similares como duplicatas.

Também pode-se perceber que a abordagem de gerencia de duplicatas baseada em atributo de similaridade no relacionamento entre nós pode ser estendida para suportar similaridade com base em multi-atributos. Ou seja, é possível calcular a similaridade entre outros atributos textuais de um objeto de dados e criar novos relacionamentos no Neo4j para gerenciar essa semelhança. Essa abordagem no entanto não foi testada, ficando como sugestão para trabalhos futuros.

Referências

Aurélio (2019). Significado de Similaridade. Dicionário do Aurélio Online. Último acesso em 29.07.2019.

Baeza-Yates, R. A. and Ribeiro-Neto, B. A. (2011). *Modern Information Retrieval – the Concepts and Technology behind Search*. Pearson, 2sd edition.

Chaudhuri, S., Ganti, V., and Kaushik, R. (2006). A Primitive Operator for Similarity Joins in Data Cleaning. In *Proceedings of the IEEE International Conference on Data Engineering*, page 5.

Cohen, W. W., Ravikumar, P. D., and Fienberg, S. E. (2003). A comparison of string distance metrics for name-matching tasks. In *Proceedings of IJCAI-03 Workshop on Information Integration on the Web*, pages 73–78.

Dong, X. L. and Naumann, F. (2009). Data Fusion - Resolving Data Conflicts for Integration. *PVLDB*, 2(2):1654–1655.

Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. (2007). Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16.

Gruenheid, A., Dong, X. L., and Srivastava, D. (2014). Incremental Record Linkage. *Proceedings of the VLDB Endowment*, 7(9):697–708.

Hernández, M. A. and Stolfo, S. J. (1998). Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9–37.

Navarro, G. (2001). A Guided Tour to Approximate String Matching. *ACM Computing Surveys*, 33(1):31–88.

Ribeiro, L. A. and Härder, T. (2011). Generalizing Prefix Filtering to Improve Set Similarity Joins. *Information Systems*, 36(1):62–78.

Ribeiro, L. A., Schneider, N. C., de Souza Inácio, A., Wagner, H. M., and von Wangenheim, A. (2016). Bridging Database Applications and Declarative Similarity Matching. *Journal of Information and Data Management*, 7(3):217–232.

Ukkonen, E. (1992). Approximate String Matching with q-grams and Maximal Matches. *Theoretical Computer Science*, 92(1):191–211.

van Erven, G. C. G. (2015). MDG-NoSQL: Modelo de Dados para Bancos NoSQL Baseados em Grafos. Dissertação, Universidade de Brasília - UnB, Brasília.

Vaz, R. V., de Oliveira, J. D. Q., and Ribeiro, L. A. (2019). Duplicate Management Using Graph Database Systems: A Case Study. In *Proceedings of the XV Brazilian Symposium on Information Systems*, pages 50:1–50:8.