Detecção de Similaridade entre consultas SQL para fins educacionais

Gabriel Luiz Garbossa[1], **Sergio L. S. Mergen**[2]

[1]Centro de Tecnologia – Universidade Federal de Santa Maria (UFSM)
97105-900 -– Santa Maria -– RS — Brasil

[2]Departamento de Linguagens e Sistemas de Computação
Universidade Federal de Santa Maria (UFSM)

{glgarbossa,mergen}@inf.ufsm.br

Abstract. *This paper proposes a results comparison algorithm to facilitate the evaluation of SQL academic exercises, recognizing the complexity of such queries that challenge the capacity of teachers in dealing with subtle variations on answers. The algorithm aims to generate messages indicating how similar the results are between the reference query (template) and the student's attempt. In face of the diversity of possible answers, many parcially correct, like the use of a different number of columns or distinct filtering criteria, the algorithm plays the key role of noticing such subtleties. The results highlight the algorithm's efficiency in simplifying the correction process for educators, providing immediate and detailed feedback to students, thus promoting a more equitable and efficient assessment in the context of distance learning.*

Resumo. *Este artigo propõe um algoritmo de comparação de resultados para facilitar a avaliação de exercícios acadêmicos de SQL, reconhecendo a complexidade dessas consultas que desafiam a capacidade dos professores em lidar com variações sutis nas respostas. O algoritmo busca gerar mensagens indicando a semelhança entre os resultados da consulta de referência (gabarito) e as tentativas dos alunos. Diante da diversidade de respostas possíveis, muitas delas parcialmente corretas, como o uso de um número de colunas diferente ou critérios de filtragem distintos, o algoritmo desempenha o papel crucial de perceber essas sutilezas. Os resultados alcançados destacam a eficácia do algoritmo em simplificar o processo de correção para educadores, fornecendo feedback imediato e detalhado aos alunos, promovendo assim uma avaliação mais equitativa e eficiente no contexto do ensino à distância.*

1. Introdução

A Linguagem de Consulta Estruturada (SQL) desempenha um papel fundamental na manipulação e recuperação de dados em bancos de dados relacionais, sendo amplamente adotada tanto na indústria quanto no meio acadêmico. A facilidade de aprendizado, derivada de suas palavras-chave em inglês, e a integração fluida com diversas linguagens de programação contribuíram para sua popularidade entre profissionais da área.

Com o advento do ensino à distância, especialmente impulsionado pela pandemia de COVID-19, o uso generalizado de computadores pessoais para tarefas acadêmicas

trouxe consigo novos desafios e oportunidades. Enquanto a conveniência do aprendizado remoto aumenta, a avaliação eficiente de exercícios acadêmicos, especialmente aqueles envolvendo consultas SQL, torna-se crucial.

Diante da diversidade de consultas presentes em exercícios acadêmicos, a tarefa de avaliação muitas vezes desafia a capacidade dos professores, dada a sensibilidade às pequenas variações nas consultas que podem levar a resultados distintos. Nesse contexto, este artigo propõe o desenvolvimento de um algoritmo de comparação de resultados para consultas SQL, destinado a simplificar e aprimorar o processo de avaliação tanto para educadores quanto para estudantes.

O algoritmo proposto busca gerar uma mensagem de retorno que indique o quão semelhantes são os resultados entre uma consulta de referência (gabarito) e a tentativa do aluno. Além de apresentar uma solução prática para a correção, vislumbramos a incorporação desse algoritmo em uma ferramenta de execução de consultas SQL. Tal ferramenta, além de simplificar o processo de correção para professores, ofereceria aos alunos feedback imediato e detalhado sobre suas consultas, promovendo um ambiente acadêmico mais equitativo e adaptado às exigências do ensino remoto.

Este artigo está organizado da seguinte forma: na seção 2 são apresentados os trabalhos relacionados. A seção 3 apresenta situações que o algoritmo busca identificar. Na seção 4 o algoritmo proposto é detalhado. Os experimentos realizados são apresentados na seção 5, e a seção 6 traz as considerações finais.

2. Trabalhos Relacionados

O problema geral investigado neste trabalho envolve encontrar correspondências entre registros de duas tabelas de resultados. Na literatura, este problema já recebeu bastante atenção, em diversos cenários de aplicação, e tem seu nome ligado à diversas linhas de pesquisa, como *field matching*, *data integration*, *data scrubbing*, *data cleansing*, *duplicate detection* e *entity resolution*. Na área de banco de dados, as linhas de pesquisa que se aproximam bastante do tema deste trabalho são *Record Linkage* e *Record deduplication*. *Record Linkage* é usado para identificar registros equivalentes em diferentes fontes de dados, o que pode ser desafiador devido a erros nos dados, variações de representação e outros fatores. Por outro lado, *Record deduplication* visa manter uma única correspondência para registros idênticos dentro de uma base de dados, contribuindo para a qualidade e precisão dos dados.

A correspondência entre dois registros normalmente leva em consideração a similaridade combinada entre os seus atributos. Por exemplo, considerando que as fontes possuem os atributos 'nome', 'data-nascimento' e 'telefone', o desafio é encontrar uma forma de usar essas informações para inferir um valor de similaridade entre os registros. Uma das ideias propõe o uso de regras de casamento que levam em consideração o tipo de cada atributo [Fan et al. 2009, Azeroual et al. 2022]. Outros trabalhos buscam estabelecer quais atributos são mais importantes no processo de casamento, quais funções de similaridade são mais adequadas para cada tipo de atributo, e quais pontos de corte devem ser utilizados para separar os casamentos bons dos ruins [Chaudhuri et al. 2007, Wang et al. 2011].

Caso o objetivo seja encontrar correspondências entre registros de duas fontes de dados volumosas, é importante reduzir a quantidade de registros que preci-

sam ser comparados, para fins de desempenho. Existem inúmeros trabalhos na literatura que buscam dividir os registros em buckets, de modo que apenas os registros que pertençam ao mesmo bucket precisem ser comparados entre si [Whang et al. 2009, Papadakis et al. 2020, De Vries et al. 2011].

De modo geral, nosso trabalho se distingue em três pontos fundamentais. Em primeiro lugar, o volume de registros é baixo: trata-se do resultado obtido em consultas sobre uma única base de dados. Em segundo lugar, espera-se que os registros correspondentes tenham uma alta similaridade entre seus atributos, uma vez que eles são derivados de uma mesma origem. Por exemplo, se duas consultas devolvem o mesmo atributo, caso não seja aplicada uma função de transformação, o conteúdo desse atributo será o mesmo. Para finalizar, um registro terá no máximo uma correspondência (casamento 1-1), justamente porque espera-se detectar a equivalência entre resultados. Essas deduções permitem que sejam usadas simplificações no cálculo da similaridade de atributos. Além disso, a restrição de casamento entre registros 1-1 permite o uso de uma técnica mais adequada para a definição da correspondência, como será visto a seguir.

3. Contextualização

Pode haver diversas variações de respostas em um exercício SQL. Muitas delas, mesmo diferindo da solução oficial, podem ser consideradas parcialmente corretas. Esta diversidade é ilustrada através de alguns exemplos, conforme descrito na Figura 1.

Para cada situação apresentada na figura, há duas tabelas de resultados. As cores são usadas para indicar as correspondências encontradas, seja na dimensão das linhas ou das colunas. Por exemplo, duas linhas iguais são representadas pela mesma cor nas duas tabelas.

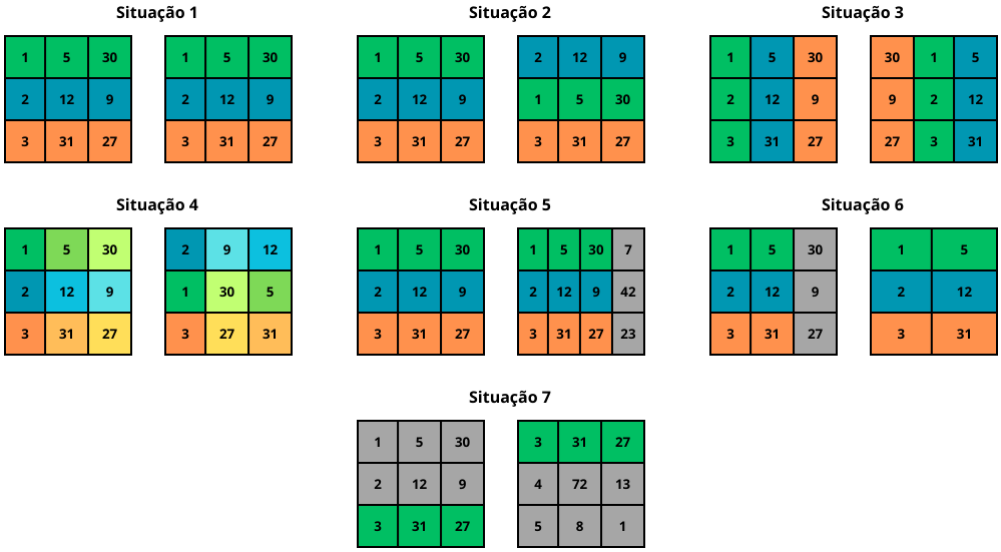

Figura 1. Situações identificadas pelo algoritmo

A situação 1 apresenta um caso em que dois resultados são equivalentes. Isso não implica que as consultas são necessariamente equivalentes, ou seja, que sempre produzirão os mesmos resultados. Porém, resultados iguais servem pelo menos como um indicativo de que as consultas sejam de fato idênticas.

Os demais exemplos trazem situações de divergência. Por exemplo, na situação 2, os dois resultados trazem as mesmas linhas, porém fora de ordem. Isso seria um indicativo de que existe algum problema com a cláusula ORDER BY. Já na situação 3, os registros estão na mesma ordem. Contudo, as colunas aparecem em posições diferentes, indicando que foi usado outro arranjo de colunas na clausula SELECT. A situação 4 combina os problemas descritos pelas situações 2 e 3, com posições diferentes tanto para linhas quanto para colunas.

A situação 5 demonstra um caso em que as linha são iguais, com exceções de uma coluna excedente do lado esquerdo. De forma complementar, na situação 6, a coluna excedente encontra-se do lado direito. Os dois casos sugerem que são necessárias adequações na cláusula SELECT, seja removendo colunas (situação 5) ou adicionando colunas (situação 6).

Por fim, a situação 7 apresenta um caso em que, pelo menos para algumas linhas, não foi possível encontrar qualquer tipo de semelhança. Isso é um indicativo de que a tentativa do aluno possui problemas de formulação que necessitam ajustes mais profundos.

A diversidade de semelhanças entre as tabelas de resultado indica a importância em identificar com precisão os pontos de divergência, de modo a fornecer mensagens mais valiosas, tanto para quem corrige os exercícios quanto para os alunos, que podem usar as mensagens para aprimorar as suas consultas.

4. Proposta

O algoritmo proposto neste trabalho recebe duas consultas em SQL, sendo que uma delas é considerada como o gabarito da questão e a outra é a tentativa do aluno. O algoritmo então executa as consultas e transforma os resultados recebidos em duas matrizes, sendo que cada linha da matriz corresponde a um registro retornado. A partir dessas duas matrizes, uma sequência de etapas é seguida;

- É realizada a comparação linha-a-linha entre as matrizes;
- A comparação linha-a-linha gera uma matriz de escores de similaridade;
- A matriz de escores é analisada para encontrar a melhor correspondência 1-1 entre as linhas;
- A partir das correspondências encontradas, o usuário recebe uma mensagem que classifica a consulta do usuário de acordo a sua relação com o gabarito.

Dado que o algoritmo opera extensivamente com matrizes, é apropriado fornecer uma definição formal que será adotada daqui em diante. Uma matriz M é constituída por uma lista de linhas l_i, onde i representa a posição da linha. Por sua vez, cada linha é formada por uma lista de colunas c_j, onde j denota a posição da coluna dentro da linha.

É importante destacar que o objetivo principal é identificar se uma linha da tabela do gabarito corresponde a uma linha da tabela de tentativa. Neste trabalho, linhas que correspondem são chamadas de linhas equivalentes. A falha em encontrar uma correspondência indica que existem erros relevantes na consulta (conforme ilustrado pela situação 7 (Figura 1)

Para que duas linhas sejam equivalentes, deve-se verificar se elas provém de uma mesma origem, ou seja, se elas foram geradas a partir dos mesmos registros do banco

de dados. Por isso, se o conteúdo de duas colunas não for igual, assume-se que elas não provenham do mesmo atributo. Convém destacar que, em cenários reais, duas colunas podem prover de uma mesmo atributo mesmo quando seus conteúdos diferem. Isso pode ocorrer, por exemplo, quando uma função de transformação é aplicada (Ex. adicionar uma máscara sobre um numero de CPF). A resolução desses casos exige o uso de funções de similaridade (como a distância de edição), o que está fora do escopo deste estudo.

Duas linhas também podem ser consideradas equivalentes mesmo que as colunas estejam em ordens diferentes (conforme ilustrado nas situações 3 e 4). Contudo, devido à limitação de escopo, estabeleceu-se que só existe equivalência caso toda a linha menor esteja contida dentro da linha maior.

Nas próximas seções, as principais etapas do processo de comparação entre as matrizes e a geração de mensagens de resposta são apresentadas em detalhes.

4.1. Comparação das matrizes

A similaridade entre as matrizes é verificada linha por linha, ou seja, cada linha l_i da matriz M_1 é comparada com cada linha l_j da matriz M_2.

A comparação linha-a-linha calcula um valor de similaridade baseada na quantidade de colunas iguais, ou seja, na intersecção entre as colunas de cada linha. A similaridade entre duas linhas l_i e l_j é dada pela função $indexJaccard$, conforme descrito na Equação 1. Esta função encontra um escore de similaridade entre 0 e 1, tal que, quanto maior a intersecção entre as colunas de cada linha, maior é a similaridade.

$$indexJaccard(l_i, l_j) = \frac{|l_i \cap l_j|}{|l_i \cup l_j|} \tag{1}$$

É importante ressaltar que a intersecção entre duas linhas busca por colunas cujo conteúdo seja equivalente (independente da ordem). Essa é uma forma de inferir que essas colunas provém de uma mesma origem (de um mesmo atributo).

4.2. Geração da matriz de escores

A partir do processo explicado na seção anterior, é gerada uma matriz mxn, tal que m é o número de linhas do resultado da consulta 1(gabarito), e n é o número de linhas do resultado da consulta 2(tentativa do aluno), e um escore$_{i_j}$ representa a similaridade entre uma linha l_i da consulta 1 e uma linha l_j da consulta 2.

A Figura 2 demonstra um exemplo de cálculo dos escores de similaridade para duas matrizes de resultado. Percebe-se que os dois resultados são iguais, porém, com as duas últimas linhas invertidas. A matriz da direita apresenta os escores de similaridade obtidos para todas as combinações de linhas. Os escores variam de 0 a 1, conforme o retorno obtido pela aplicação da função de Jaccard. O escore máximo aparece em destaque na figura.

Com os escores gerados, deve-se encontrar qual é o melhor casamento entre uma linha l_i da consulta 1 e uma linha l_j da consulta 2. Neste caso específico, é trivial encontrar os melhores casamentos, uma vez que apenas as linhas correspondentes obtiveram o escore mais alto. (linhas 1,1, linhas 2,2, linhas 3,3, linhas 4,5 e linhas 5,4)

No entanto, pode ocorrer casos em que as linhas correspondentes não apresentem o escore máximo (ex. quando as linhas são as mesmas, mas o número de colunas difere, como na situação 6). Além disso, em alguns casos, é melhor minimizar o escore de similaridade entre duas linhas para poder maximizar o escore global entre todas as linhas casadas.

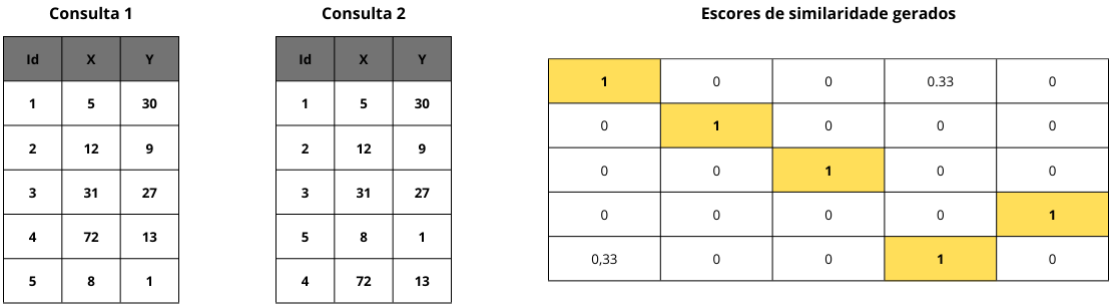

Figura 2. Exemplo de escores gerados pelo algoritmo

Note que tal problema se assemelha ao problema de atribuição, em que n tarefas devem ser atribuídas a m agentes de modo a maximizar a eficácia da atribuição. No nosso caso, tantos os agentes quanto as tarefas são as linhas das consultas a serem comparadas. Dessa forma, pode-se utilizar um dos métodos de atribuição conhecidos para resolver o problema em questão.

O método escolhido foi uma adaptação do Método da Matriz Húngara [Kuhn 1955]. Esse método usa matrizes para representar custos de atribuição entre dois conjuntos, como tarefas e recursos. O pseudo-código descrito no Algoritmo 1 apresenta as ideias principais desse método. De modo resumido, o método inicia reduzindo linhas e colunas, simplificando a matriz. Zeros são então criados para facilitar a busca de soluções. Encontram-se atribuições iniciais e, através de passos alternados, a solução é aprimorada até alcançar a ótima. A estrutura da matriz permite operações eficientes, tornando o método valioso para problemas de alocação.

Duas modificações forem realizadas a partir do método inicial. A primeira é multiplicar a matriz de escores geradas por -1, pois o método húngaro tradicionalmente é utilizado para minimizar os pesos da matriz. Ao negar a matriz de entrada, em vez de minimizar os pesos da matriz, o método irá maximizá-los.

A segunda modificação feita é em relação aos tamanhos da matriz de entrada e da matriz de saída. O método utilizado foi criado para ser utilizado em matrizes quadradas nxn. Porém, como deseja-se detectar quando quaisquer uma das consultas estiver contida dentro da outra, o algoritmo precisa funcionar mesmo quando o número de linhas das consultar for diferente. Caso seja necessário, a matriz de entrada é alterada adicionando-se linhas ou colunas até que se torne uma matriz nxn tal que $n = max(l1, l2)$, em que $l1$ é o número de linhas na primeira consulta e $l2$ é o número de linhas na segunda consulta; e a matriz de saída será uma matriz mx2 tal que $m = min(l1, l2)$.

4.3. Geração da mensagem de retorno

Após a aplicação do método húngaro, obtém-se os melhores casamentos entre as linhas dos dois resultados. A partir desses casamentos, deve-se gerar uma mensagem resumida

Algoritmo 1: Algoritmo do Método Húngaro

Entrada: Matriz de pesos: M = ($M_(ij)$) de ordem n
Saída: Matriz de coordenadas otimizadas: O = ($O_(ij)$) de ordem m
início
  for *linha in* $M_(ij)$ **do**
    encontra o menor valor k da linha
    subtrai k de todos os elementos da linha
  end
  for *coluna in* $M_(ij)$ **do**
    encontra o menor valor k da coluna
    subtrai k de todos os elementos da coluna
  end
  marca os zeros da matriz com o menor número x de marcações possível
  while $x < n$ **do**
    encontra o menor valor k não marcado
    subtrai k de todos os valores não marcados
    soma k aos elementos nas intersecções das marcações
    remove as marcações x
    marca os zeros da matriz com o menor número x de marcações possível
  end
fim

de comparação. Essa mensagem deve indicar se a resposta está correta, incorreta ou se precisa de alguma modificação.

Para gerar a mensagem, primeiro deve-se verificar a ocorrência de uma série de situações de divergência que faz com que as consultas difiram. As situações são: 'Equivalência', 'Ordem das Linhas', 'Ordem das Colunas' e 'Dimensionalidade.'

Equivalência: Esta situação determina se as linhas do gabarito possuem uma linha equivalente na resposta do usuário. Para cada linha do gabarito, é buscada a linha casada, e verifica-se se uma está contida dentro da outra. Caso qualquer linha do gabarito não satisfaça essa condição, a seguinte mensagem é gerada: 'A consulta não retorna os resultados esperados'.

Ordem das Linhas: Esta situação determina se as linhas casadas entre as matrizes estão na mesma ordem. Isso pode ser facilmente verificado na solução retornada pelo método Húngaro: caso a melhor solução possua algum par de linhas casadas cujas posições não sejam iguais (como 1-1, 2-2, 3-3, etc), isso significa que as linhas do resultado não estão na mesma ordem do que as linhas do gabarito. Nesse caso, a mensagem de retorno é: 'o resultado gerado não está na ordem esperada. Revise a cláusula ORDER BY'.

Ordem das Colunas: Esta situação determina se as colunas correspondentes entre as linhas casadas estão na mesma ordem. Essa situação é verifica ao se comparar a posição das colunas de todas as linhas casadas. Considerando que $\{c_a, c_b\} \in l_i$ e $\{c_x, c_y\} \in l_j$, então, para cada par de casamentos (c_a, c_x) e (c_b, c_y), é necessário que, se $a < b$, então

$x < y$. Ou seja, as colunas iguais devem estar na mesma ordem nas duas linhas. Se, para qualquer par de linhas casadas, esse posicionamento não for verificado, então a ordem não está sendo mantida. Nesse caso, a mensagem de retorno é: 'A ordem das colunas na cláusula SELECT difere do resultado oficial. Verifique se essa ordem é importante.'.

Dimensionalidade: Esta situação determina se um dos resultados possui mais linhas ou mais colunas do que o outro. Caso o número de linhas ou colunas for diferente, significa que há dados sobressalentes ou faltantes na consulta do usuário. Nesse caso, a mensagem de retorno é: 'O número de colunas ou de registros gerados difere do resultado oficial. Verifique a cláusula SELECT ou a cláusula WHERE.'.

A mensagem final de retorno é uma concatenação de todas as mensagens indicativas de erro. A exceção é a situação de equivalência. Caso ela não seja observada, uma única mensagem é gerada, indicando que a consulta possui erros que extrapolam situações específicas.

5. Experimentos Realizados

Esta seção descreve os experimentos conduzidos para validação da abordagem proposta na avaliação das soluções para tarefas de consultas a bancos de dados. Foram utilizadas sete conjuntos de exercícios passados na disciplina de Fundamentos de Banco de Dados, oferecida no semestre 2023/01 pelo Departamento de Linguagens e Sistemas de Computação da UFSM, compreendendo um total de 105 consultas. O banco de dados empregado nessas listas contém informações sobre filmes, diretores, atores e participações de atores em filmes.

Foram usados exercícios enviados por cinco alunos (selecionados aleatoriamente) para realizar os testes, totalizando 359 exercícios. Então, conduziu-se uma análise manual para determinar a precisão do algoritmo, registrando a quantidade de vezes que o algoritmo identificou corretamente e incorretamente cada uma das sete situações.

O resultado foi compilado na Figura 3. A figura apresenta as ocorrências geradas para cada uma das sete situações gerais apresentadas na Figura 1.

Nº situação	Nº acertos	Nº erros
1	161	0
2	46	0
3	2	5
4	1	6
5	41	0
6	12	0
7	63	22

Figura 3. Resultados obtidos nos experimentos

Pode-se notar que o algoritmo possui alta precisão em identificar casos de consultas equivalentes (situação 1), casos aonde as linhas possuem uma ordem di-

ferente (situação 2), bem como casos de resultados apresentando dimensionalidade diferente(situações 5 e 6).

No entanto, casos envolvendo a ordem das colunas causaram falsos positivos. Para as situações 3 (apenas colunas foram de ordem) e 4(colunas e linhas fora de ordem), as taxas de acertos foram de 28% e 16%, respectivamente. Uma das ocorrências de erro mais comuns está associada a situações em que a mesma coluna é repetida nos resultados. Para exemplificar, considere o cenário apresentado na Tabela 1, onde o objetivo é retornar dados de filmes que contaram com artistas americanos.

Gabarito	Tentativa de aluno
SELECT * **FROM** filme **WHERE** idFilme **in** (**SELECT** idFilme **FROM** elenco **NATURAL JOIN** ator **WHERE** pais = 'EUA')	**SELECT** f.* **FROM** filme f **JOIN** elenco e **ON** f.idFilme = e.idFilme **WHERE** idAtor **IN** (**SELECT** idAtor **FROM** ator **WHERE** pais = 'EUA') **GROUP BY** f.idFilme

Tabela 1. Situação de erro envolvendo colunas encontradas em ordens diferentes.

Ao executar ambas as consultas, é possível se verificar manualmente que as duas geram os mesmos resultados. No entanto, o algoritmo identifica uma ordem de colunas diferente. Este fenômeno se deve ao fato de uma das linhas possuir mais de uma coluna com o mesmo valor, o que leva o algoritmo à uma falsa constatação de ordem de colunas distinta entre as consultas.

Conforme apresentado na Figura 3, um dos casos de erro mais comuns estão associados à consultas marcadas como não equivalentes de maneira incorreta (situação 7). Isto se dá devido as limitações do algoritmo em identificar similaridades nos valores das colunas. Para exemplificar, considere o cenário apresentado na Tabela 2, onde o objetivo é retornar filmes em ordem descrente de lucro.

Gabarito	Tentativa de aluno
SELECT titulo , ROUND((bilheteria/custo)) **FROM** filme **ORDER BY** 2 **desc**	**SELECT** titulo , bilheteria/custo **as** lucro **FROM** filme **ORDER BY** lucro **DESC**

Tabela 2. Situação de erro envolvendo colunas com conteúdo diferente.

Uma análise superficial da sintaxe de ambas as consultas é suficiente para constatar que a única diferença entre o gabarito e a consulta gerada pelo aluno é o uso da função $round()$. Portanto, ambas geram essencialmente o mesmo resultado. Porém, o algoritmo conclui que não há equivalência entre as consultas pois os valores da coluna "lucro"divergem (Por exemplo, para uma linha específica, o gabarito exibe o valor 12 enquanto a tentativa exibe o valor 12,33334).

6. Considerações Finais

Este artigo apresentou uma proposta de algoritmo para a detecção de similaridade entre os resultados de duas consultas SQL. Foram apresentados experimentos que avaliam o comportamento do algoritmo para o problema de correção de tarefas.

Cabe salientar que o objetivo da ferramenta não é automatizar as tarefas de correção de exercícios, mas servir como assistente para que uma análise mais profunda seja realizada. A proposta também pode ser empregada em ambientes virtuais de aprendizagem como um mecanismo de autocorreção, em que o aluno realiza envios sucessivos de uma tarefa, usando as mensagens de retorno como feedback que mede o progresso do aprendizado.

Como trabalhos futuros, pretende-se enriquecer os detalhes referentes à questão da dimensionalidade, destacando se o problema reside no número de linhas ou colunas. O algoritmo também será aprimorado a partir dos erros identificados nos experimentos realizados. Por exemplo, é necessário considerar casos em que o mesmo valor ocorre em colunas diferentes. Além disso, como visto no experimentos, é importante incorporar a análise de similaridade entre os valores de uma coluna, para que colunas sejam consideradas compatíveis mesmo que seus conteúdos não sejam equivalentes. Nessa mesma linha, pretende-se identificar o tipo de cada coluna, para que o método de detecção de similaridade mais apropriado seja usado.

Referências

Azeroual, O., Jha, M., Nikiforova, A., Sha, K., Alsmirat, M., and Jha, S. (2022). A record linkage-based data deduplication framework with datacleaner extension. *Multimodal Technologies and Interaction*, 6(4):27.

Chaudhuri, S., Chen, B.-C., Ganti, V., and Kaushik, R. (2007). Example-driven design of efficient record matching queries. In *VLDB*, volume 7, pages 327–338.

De Vries, T., Ke, H., Chawla, S., and Christen, P. (2011). Robust record linkage blocking using suffix arrays and bloom filters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(2):1–27.

Fan, W., Jia, X., Li, J., and Ma, S. (2009). Reasoning about record matching rules. *Proceedings of the VLDB Endowment*, 2(1):407–418.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Papadakis, G., Skoutas, D., Thanos, E., and Palpanas, T. (2020). Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys (CSUR)*, 53(2):1–42.

Wang, J., Li, G., Yu, J. X., and Feng, J. (2011). Entity matching: How similar is similar. *Proceedings of the VLDB Endowment*, 4(10):622–633.

Whang, S. E., Menestrina, D., Koutrika, G., Theobald, M., and Garcia-Molina, H. (2009). Entity resolution with iterative blocking. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 219–232.