Planos físicos de consulta para Divisão Relacional

Letícia de Campos Lima[1], **Lúcio F. D. Santos**[2], **Marcos V. N. Bedo**[3], **Enzo Seraphim**[1], **Luiz Olmes Carvalho**[1]

[1]Universidade Federal de Itajubá (UNIFEI) - Itajubá (MG), Brasil

[2]Instituto Federal do Norte de Minas Gerais (IFNMG) - Montes Claros (MG), Brasil

[3]Universidade Federal Fluminense (UFF) - Niterói (RJ), Brasil

{leticialima, seraphim, olmes}@unifei.edu.br

lucio.santos@ifnmg.edu.br, marcosbedo@ic.uff.br

Abstract. *The Relational Division operator does not have a specific command in the SQL language. Therefore, the same query can be written in different ways, but the execution performance of them are not the same. Based on some representations of the Relational Division in SQL, this article investigates the use of two strategies for optimization: the use of a composite index and the direct use of the Hash Join operator in the query execution plan, while performing the join of tuples. The experiments were executed on the PostregeSQL DBMS and the obtained results show that the proposed strategies improved the performance in 47% regarding to the database query planner.*

Resumo. *O operador de Divisão Relacional não possui um comando específico na linguagem SQL. Como consequência, uma mesma consulta pode ser escrita de diversas maneiras, mas não necessariamente com o mesmo desempenho de execução. A partir das formas de representação da Divisão Relacional em SQL, este trabalho investiga o uso de duas estratégias para otimização: uso de índice composto e uso do operador Hash Join diretamente no plano físico de consulta, durante o processo de junção de tuplas. O experimentos foram realizados com o SGBD PostregeSQL e os resultados obtidos mostram que as estratégias propostas melhoram o desempenho em 47% em relação ao obtido pelo planejador de consultas do banco de dados.*

1. Introdução

A linguagem SQL é amplamente empregada em ambientes transacionais para definição e execução de consultas em Sistemas de Gerenciamento de Bases de Dados (SGBD). Porém, o caráter declarativo desta linguagem faz com que a linguagem expresse *o que* fazer, mas não *como* fazer [Date 2015]. Dessa forma, as consultas são definidas sem considerar os operadores envolvidos em seu processamento.

As consultas sobre dados relacionais são baseadas nos operadores fundamentais da Álgebra Relacional. Este conjunto de operadores de consulta, como originalmente definidos [Codd 1972], inclui o operador de Divisão Relacional. A Divisão Relacional é uma operação binária que permite responder consultas do tipo "para todos os/as" ou então "nenhum dos/das" [Vasconcelos et al. 2018, Gonzaga and Cordeiro 2019]. Por exemplo,

Pessoa	
nome	*idioma*
Ana	Inglês
Ana	Espanhol
José	Francês
José	Inglês
Pedro	Inglês

÷

Idioma
nome
Inglês
Espanhol

=

Resposta
nome
Ana

Figura 1. Exemplo da operação de Divisão Relacional

seja uma relação *Pessoa* contendo o nome e os idiomas que uma pessoa domina. A consulta "*Quais pessoas falam, conjuntamente, inglês e espanhol?*" é respondida através de uma Divisão Relacional, como mostrado na Figura 1. Dada a tabela *Pessoa*, o objetivo é filtrar as pessoas que falam *todos* os idiomas desejados na busca.

Dentre os operadores algébricos, o operador de Divisão Relacional não possui um comando específico em SQL. Dessa forma, para que esse operador seja executado, é necessário a combinação de dois ou mais outros operadores para compor a consulta. Em consequência dessas combinações, a mesma consulta pode ser descrita de diversas maneiras distintas, todas produzindo o mesmo resultado, mas não necessariamente com o mesmo desempenho de execução. De fato, alguns trabalhos presentes na literatura [Matos and Grasser 2002, Gonzaga and Cordeiro 2016] avaliaram formas distintas de representação SQL da Divisão Relacional e concluíram que a mais eficiente é a Divisão Baseada em Contagem de Tuplas (Secão 2).

Ao analisar o exemplo da Figura 1, observa-se que o processamento de uma Divisão Relacional normalmente envolve poucos atributos, empregados com o propósito de realizar comparações por igualdade, e também a combinação de mais de uma tabela do banco usando junção. Embora o SGBD obtenha bons resultados ao definir o plano físico de execução, algumas estratégias podem ser exploradas para melhorar ainda mais o desempenho da operação de divisão. Neste contexto, este trabalho tem como objetivo investigar e validar duas estratégias, que podem ser aplicadas sobre o plano físico de uma Divisão Relacional, de forma a reduzir o tempo de execução desta operação. São elas:

1. *Criação de índice composto*: um único índice composto, formado pela combinação dos atributos necessários à divisão, em uma mesma estrutura pode aumentar a eficiência da busca.
2. *Uso do operador Hash Join*: este é um dos operadores físicos de junção mais rápidos presente nos SGBDs. Visto que a operação de junção estará presente na execução da Divisão, este operador auxilia na redução do tempo da consulta.

As próximas Seções do artigo estão organizadas como segue: a Seção 2 apresenta o referencial teórico. A Seção 3 descreve as técnicas propostas. A Seção 4 discute os experimentos, que foram realizados em uma base com milhares de tuplas, usando o SGBD PostrgreSQL. A Seção 5 apresenta a conclusão do trabalho.

2. Referencial Teórico

Divisão Relacional. Dadas duas relações R e S, onde R é chamada de Dividendo e S atua como Divisor, a Divisão Relacional $R \div S$ é expressa em notação de cálculo por:

$$R[A \div B]S = R[\bar{A}] - \big(\big(R[\bar{A}] \times S[B]\big) - R\big)\,[\bar{A}]$$

onde, $A \subseteq R$ e $B \subseteq S$ são listas de atributos compatíveis em união, sobre os quais a Divisão é realizada, e $\bar{A}$ contém os atributos que não estão em A [Codd 1972].

Considerando a inexistência de um comando SQL para representar a Divisão Relacional, seu resultado pode ser computado por ao menos sete formas distintas de consultas escritas em SQL [Date 2015]. Diversos trabalhos da literatura já investigaram estas representações de consulta [Matos and Grasser 2002, Leinders and Bussche 2007, Celko 2009, Gonzaga and Cordeiro 2016], sendo que a expressão SQL tida como a mais eficiente é chamada de Divisão Relacional Baseada em Contagem de Tuplas.

Trabalhos correlatos. Diversos estudos têm sido realizados com o operador de Divisão Relacional. A primeira proposta de escrita desta operação usando Contagem de Tuplas, da qual o presente trabalho faz uso, é encontrada em [Matos and Grasser 2002]. Nesta mesma linha, em [Gonzaga and Cordeiro 2016] tem-se a implementação do operador de Divisão Relacional através de *stored procedures*, que, internamente, fazem uso de índices. A diferença para o presente trabalho é que o índice aqui criado é composto por vários atributos, o que permite processar dados consultando uma única estrutura. Além disso, o presente trabalho também altera operadores físicos do plano de execução da consulta.

Em [Gonzaga and Cordeiro 2019] é proposto um operador de divisão por similaridade, variando-se a característica dos dados, pois estes não seguem uma relação de igualdade/precedência entre si, mas estão dispostos em espaços de distância. O emprego de Divisão Relacional sobre dados em espaços de distância e espaços métricos também é encontrado em [Vasconcelos et al. 2018], onde a operação é implementada no SGBD comercial Oracle, juntamente com uma extensão da linguagem SQL, para possibilitar a execução da Divisão Relacional por similaridade. O presente trabalho não engloba o conceito de similaridade, mas sim, relações de igualdade e ordem total entre os dados.

A proposta encontrada em [Leinders and Bussche 2007] apresenta um estudo matemático mostrando que a complexidade da operação é minimamente quadrática, porém, não aborda uma experimentação prática e nem discute possíveis representações SQL. Em [Leinders and Bussche 2007] ainda são discutidas possibilidades de uso de operações como contagem e ordenação no processamento. A questão da ordenação, que está fora do escopo do presente estudo, é empregada em [Vaverka and Vychodil 2016], onde são propostas formas de Divisão Relacional para banco de dados baseados em ranqueamento de resultados. Na mesma linha, que foge da teoria relacional clássica, em [Tamani et al. 2013] é possível encontrar a operação de Divisão Relacional aplicada a bases de dados baseadas na teoria da lógica e conjuntos difusos.

3. Materias e Métodos

Seja a descrição de um *minimundo*: "*uma escola de idiomas possui alunos e professores. Os professores lecionam idiomas em cursos (inglês, espanhol, etc.). Os alunos podem se matricular nos idiomas oferecidos, em diferentes níveis (básico, intermediário, etc.)*". Neste exemplo, após realizar a modelagem Entidade-Relacionamento e posterior mapeamento para o Modelo Relacional, algumas das tabelas do banco de dados (simplificadamente, apenas para mostrar os campos envolvidos na Divisão Relacional) são:

```
Aluno(id, nome)      Professor(id, nome)
Idioma(nome)         Nivel(nome)
Curso(prof(Professor.id), idioma(Idioma.nome))
Turma(aluno(Aluno.id), idioma(Idioma.nome))
Matricula(aluno(Aluno.id), nivel(Nivel.nome))
```

Neste modelo, seja a definição de duas consultas: (C1) "*Quais são os professores poliglotas?*" e (C2) "*Quais os alunos que se formaram?*". Em C1, o interesse é encontrar os professores que lecionam *todos* os idiomas da escola. Em C2, os alunos que se formaram são aqueles que concluíram *todos* os níveis de um idioma. Portanto, para as relações `Curso` (`C`) e `Idioma` (`I`), a consulta C1 pode ser expressa de acordo com (1), ao passo que a consulta C2, para as relações `Matricula` (`M`) e `Nivel` (`N`), é dada por (2).

$$\texttt{C}\,[idioma \div nome]\,\texttt{I} = \texttt{C}[prof] - (\,(\texttt{C}[prof] \times \texttt{I}[nome]) - \texttt{C})\,[prof] \quad (1)$$

$$\texttt{M}\,[nivel \div nome]\,\texttt{N} = \texttt{M}[aluno] - (\,(\texttt{M}[aluno] \times \texttt{N}[nome]) - \texttt{M})\,[aluno] \quad (2)$$

Estratégia 1: (1) e (2) destacam os atributos envolvidos na operação de Divisão Relacional. A depender do modelo e das consultas de interesse, os atributos usados na divisão não são chave, nem parte de chave. A criação de índices sobre eles é capaz de acelerar a operação. Porém, como cada índice constitui um arquivo separado do banco de dados, a Estratégia 1 consiste na criação de *índices compostos* por mais de um atributo, na forma $idx\langle atr_1, atr_2\rangle$. Se algum atributo já está indexado devido às chaves primárias, ele entra como atr_2, de modo que o campo principal do índice (atr_1) seja o atributo ainda não indexado. Como todas as informações necessárias à consulta se encontram no índice composto, isto é, em um único arquivo, o planejador de consultas do SGBD pode se beneficiar desta única estrutura, aumentando o desempenho da operação de Divisão Relacional.

Estratégia 2: na análise dos planos físicos de consulta dos diversos tipos de representação SQL da Divisão Relacional, inclusive na Contagem de Tuplas, foi possível observar que é frequente com que o planejador de consultas, após ler uma tabela usando busca sequencial (*Seq. Scan*) ou índices (*Index Scan*), crie na árvore de execução um nó que atua como mecanismo de *cache* (*Memoize*), para que os dados (ou parte deles) sejam consumidos pelo nó imediatamente superior. Frequentemente, este nó seguinte é uma junção por laço aninhado (*Nested Loop*). A Estratégia 2 consiste em forçar a substituição do mecanismo de *cache* por uma tabela *hash*. Portanto, a junção executada na sequência também é alterada, do operador *Nested Loop* para um *Hash Join*, de modo que o custo de criação do *hash* é compensado na execução da junção. No SGBD PostgreSQL, este procedimento é realizado através do comando `SET enable_nestloop = FALSE`. Se o operador *Nested Loop* for empregado no plano de execução como base das operações *anti-join* ou *semi-join*, não será possível substituí-lo. Isto não ocorre para a forma de Divisão Baseada em Contagem de Tuplas, mas não se mantém para outras variantes SQL da divisão.

4. Experimentos

Para executar a validação das estratégias propostas, o modelo apresentado na Seção 3 foi implementado sobre o SGBD PostgreSQL 14. As tabelas foram preenchidas com dados sintéticos, com as seguintes cardinalidades: Aluno (350.000 tuplas), Curso (2.403.283), Idioma (10), Matricula (1.678.215), Nivel (5), Professor (250.000), Turma (3.228.726). Toda a base de dados ocupa exatos 555 MB de espaço em disco. As consultas C1 e C2 foram escritas em SQL na forma de Divisão Relacional Baseada em Contagem de Tuplas.

Para cada consulta, os experimentos realizados têm como principal objetivo aferir o tempo de execução (Figura 2) dos planos físicos de consulta (*i*) produzido pelo planejador do PostgreSQL (*Planner*), apenas com a definição das tabelas e chaves, (*ii*) produzido pelo uso do operador *Hash Join* (Estratégia 2), (*iii*) usando índices únicos para os atributos da divisão e (*iv*) usando índice composto (Estratégia 1). Para a consulta C1, também

são apresentados os planos de execução obtidos em cada abordagem (Figura 3). Além disso, sobre as tabelas definidas na Seção 3, o código SQL por Contagem de Tuplas para a consulta C1 é dado por:

```
SELECT c.prof
FROM Curso c JOIN Idioma i ON c.idioma = i.nome
GROUP BY c.prof
HAVING COUNT(*) = (SELECT COUNT(*) FROM Idioma);
```

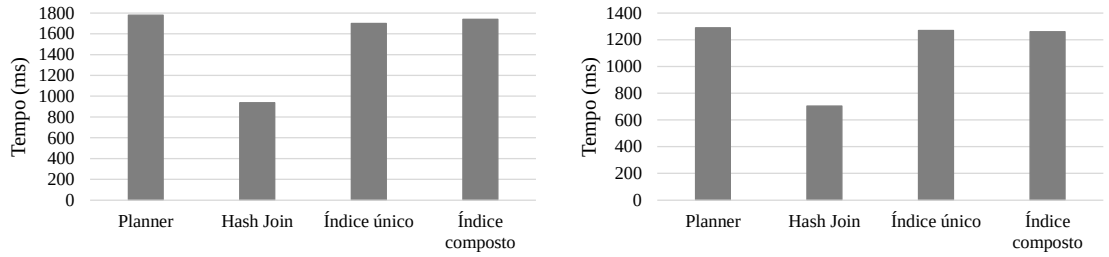
Os experimentos foram realizados em uma máquina com processador Intel Core i5-8265U (1.6 GHz), 8 GB RAM e 1 TB HDD, sobre o sistema operacional Ubuntu 22.04. Os valores apresentados correspondem à média de 10 execuções com as *caches* do sistema operacional e do SGBD sempre limpas antes de proceder a execução das consultas. Demais parâmetros do SGBD (como tamanhos de *buffers* e *cache*) não foram alterados.

A Figura 2(a) mostra os tempos obtidos na execução da consulta C1, para cada abordagem. Em relação ao *Planner*, as abordagens de Índice único e Índice composto obtiveram, respectivamente, ganhos de 4% e 2%. Quando diretamente comparadas, as duas abordagens de índices diferiram em apenas 2%, com vantagem para o Índice único. O *Hash Join* obteve o ganho mais expressivo, sendo 47% mais rápido que o *Planner*.

A Figura 3(a) apresenta o plano físico produzido pelo *Planner* e, coincidentemente, pela estratégia de Índice composto, onde é possível observar o uso do mecanismo de *cache*, via *Memoize*, seguido de *Nested Loop*. A Figura 3(b) ilustra o plano de execução quando a Estratégia 2 é aplicada. Destaca-se o uso do operador *Hash Join* em substituição ao *Nested Loop* e a execução concorrente dos operadores *Parallel Seq Scan* e *Sort*. O *Gather Merge* consome o resultado do operador *Sort* e mantém a ordenação na resposta. Por fim, a estratégia de Índice único resultou no plano mostrado na Figura 3(c), onde apenas as chaves primárias foram consideradas.

A Figura 2(b) mostra os tempos obtidos na execução da consulta C2. As abordagens de Índice único e Índice composto obtiveram, respectivamente, ganhos de apenas 1% e 2% sobre o *Planner*. Em C2, o Índice composto foi mais rápido que o Índice único, mas a diferença é menor que 1% (apenas 10 ms). O ganho de tempo no uso do operador *Hash Join* em relação ao *Planner* foi de 45%.

Para as consultas C1 e C2, verifica-se que a diferença entre os Índices único e composto é, em média, apenas 1%. O planejador de consultas priorizou o uso dos índices que constituem as chaves primárias e o índice composto mostrou-se minimamente efetivo.



(a) Consulta 1 **(b) Consulta 2**

Figura 2. Tempo de execução

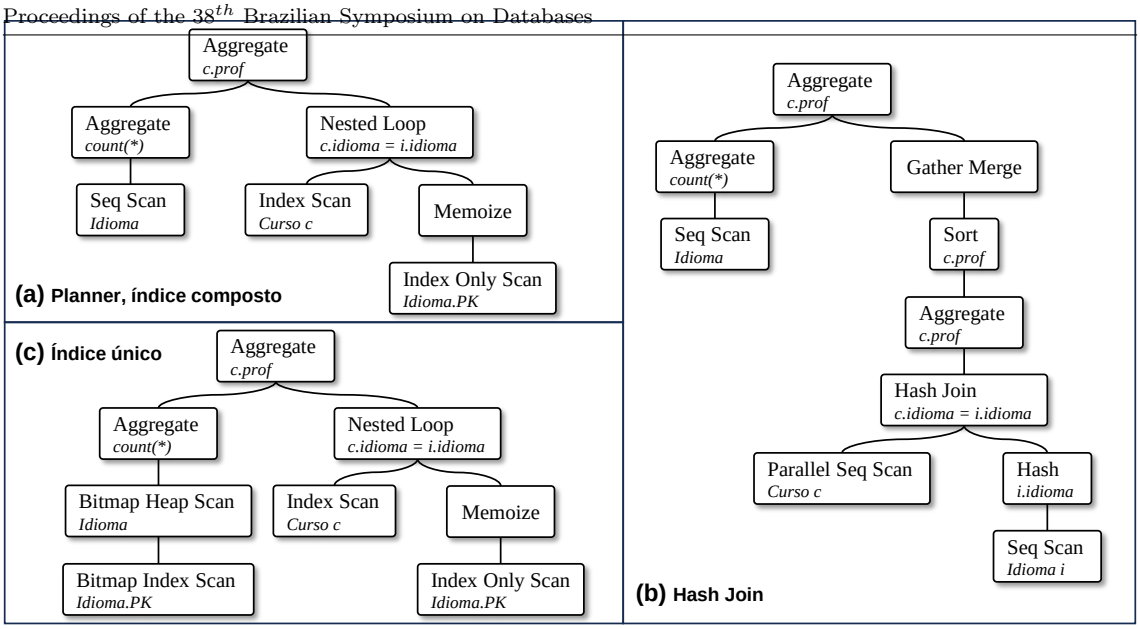


Figura 3. Planos físicos de consulta

5. Conclusão

Este trabalho apresentou uma análise dos planos físicos de consulta gerados por uma das formas de representação SQL da Divisão Relacional. Duas estratégias foram propostas para otimizar o tempo da operação. A primeira, baseada em índices compostos, apresentou variação de apenas 2% em relação ao *Planner* do SGBD. A segunda, baseada no uso do operador *Hash Join*, mostrou-se eficiente, reduzindo quase que pela metade o tempo de execução da consulta. Propostas futuras envolvem investigar o operador *Merge Join* e, para aprofundamento na análise, utilizar o *benchmark* TPC-C.

Referências

Celko, J. (2009). Divided we stand: The SQL of relational division.

Codd, E. F. (1972). Relational completeness of data base sublanguages. *Res. Report*, 987.

Date, C. J. (2015). *SQL and Relational Theory*. O'Reilly, 3rd edition.

Gonzaga, A. S. and Cordeiro, R. L. F. (2016). Fast and scalable relational division on database systems. In *Simpósio Brasileiro de Banco de Dados*.

Gonzaga, A. S. and Cordeiro, R. L. F. (2019). The similarity-aware relational division database operator with case studies in agriculture and genetics. *Inform. Systems*, 82.

Leinders, D. and Bussche, J. V. d. (2007). On the complexity of division and set joins in the relational algebra. *Journal of Computer and System Sciences*, 73.

Matos, V. M. and Grasser, R. (2002). A simpler (and better) SQL approach to relational division. *Journal of Information Systems Education*, 13.

Tamani, N., Liétard, L., and Rocacher, D. (2013). A relational division based on a fuzzy bipolar r-implication operator. In *IEEE International Conference on Fuzzy Systems*.

Vasconcelos, G. Q., Kaster, D. S., and Cordeiro, R. L. F. (2018). On the support of the similarity-aware division operator in a commercial RDBMS. In *Advances in Databases and Information Systems*.

Vaverka, O. and Vychodil, V. (2016). Relational division in rank-aware databases. *Information Sciences*, 366.