SQLSIM: consultas analíticas por similaridade em SGBD Relacionais

Antônio Lívio Cruz de Mendonça

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2025

Antônio Lívio Cruz de Mendonça

SQLSIM: consultas analíticas por similaridade em SGBD Relacionais

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Humberto Luiz Razente
Coorientador: Prof[a]. Maria Camila Nardini Barioni

Uberlândia
2025

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

M539 Mendonça, Antonio Lívio Cruz de, 1994-
2025 SQLSIM: consultas analíticas por similaridade em SGBD Relacionais [recurso eletrônico] / Antonio Lívio Cruz de Mendonça. - 2025.

Orientador: Humberto Luiz Razente.
Coorientadora: Maria Camila Nardini Barioni.
Dissertação (Mestrado) - Universidade Federal de Uberlândia, Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2025.149
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Razente, Humberto Luiz,1977-, (Orient.). II. Barioni, Maria Camila Nardini,1978-, (Coorient.). III. Universidade Federal de Uberlândia. Pós-graduação em Ciência da Computação. IV. Título.

CDU: 681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Computação
Av. João Naves de Ávila, 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Dissertação, 04/2025, PPGCO				
Data:	27 de fevereiro de 2025	Hora de início:	14:30	Hora de encerramento:	16:30
Matrícula do Discente:	12212CCP003				
Nome do Discente:	Antônio Lívio Cruz de Mendonça				
Título do Trabalho:	SQLSIM: consultas analíticas por similaridade em SGBD Relacionais				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Ciência de Dados				
Projeto de Pesquisa de vinculação:	----------				

Reuniu-se por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Maria Camila Nardini Barioni - FACOM/UFU (Coorientadora), Felipe Alves da Louza - FEELT/UFU, Josiel Maimone de Figueiredo - IC/UFMT e Humberto Luiz Razente - FACOM/UFU, orientador do candidato.

Os examinadores participaram desde as seguintes localidades: Josiel Maimone de Figueiredo - Cuiabá/MT. Os outros membros da banca e o aluno participaram da cidade de Uberlândia.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Humberto Luiz Razente, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação da Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir ao candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

Aprovado

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Humberto Luiz Razente**, **Professor(a) do Magistério Superior**, em 28/02/2025, às 15:36, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Felipe Alves da Louza**, **Professor(a) do Magistério Superior**, em 02/03/2025, às 18:30, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Josiel Maimone de Figueiredo**, **Usuário Externo**, em 05/03/2025, às 19:36, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Maria Camila Nardini Barioni**, **Professor(a) do Magistério Superior**, em 06/03/2025, às 07:56, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **6100015** e o código CRC **F978D04F**.

Referência: Processo nº 23117.009383/2025-36 SEI nº 6100015

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

Agradecimentos

Esse trabalho não teria sido possível sem a ajuda de inúmeras pessoas que me apoiaram e incentivaram durante todo essa trajetória. Quero agradecer à Fernanda Mayra, minha companheira que sempre acreditou no êxito. Ao Humberto e à Maria Camila, meus orientadores que me guiaram nesse projeto. À minha família consanguínea e à minha família afetiva, que mesmo de longe sempre me incentivaram. E aos meus amigos, que contribuíram direta ou indiretamente nesse trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

"Onde há vitória já houve dor."
(Em busca de nós mesmos)

Resumo

Com o crescimento exponencial de dados multidimensionais, torna-se essencial o desenvolvimento de abordagens eficientes para consultas analíticas por similaridade. No entanto, os Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR) tradicionalmente não oferecem suporte nativo para esse tipo de consulta, tornando necessário o uso de soluções externas que podem impactar a eficiência e a escalabilidade. Neste trabalho, propomos o SQLSIM, uma estratégia para execução de consultas por similaridade diretamente em SGBDRs, eliminando a necessidade de processamento externo e reduzindo o *impedance mismatch*. A abordagem utiliza funções definidas pelo usuário (UDFs) na linguagem procedural SQL para implementar operações de agrupamento por similaridade em múltiplas dimensões. Além disso, exploramos técnicas para otimizar a execução dessas consultas, aproveitando estruturas de indexação e operadores avançados. A solução foi implementada no PostgreSQL e avaliada por meio de experimentos que demonstraram sua flexibilidade no processamento de grandes volumes de dados. Os resultados indicam que o SQLSIM permite a realização de análises multidimensionais complexas de forma integrada, preservando a compatibilidade com a linguagem SQL padrão.

Palavras-chave: Consultas por similaridade. SQL. SGBDR. Agrupamento. Análise multidimensional.

Abstract

With the exponential growth of multidimensional data, developing efficient approaches for similarity-based analytical queries has become essential. However, Relational Database Management Systems (RDBMS) do not natively support this type of query, often requiring external solutions that can impact efficiency and scalability. In this work, we propose SQLSIM, a strategy for executing similarity-based queries directly within RDBMS, eliminating the need for external processing and reducing impedance mismatch. The approach leverages user-defined functions (UDFs) in procedural SQL to implement similarity-based clustering operations across multiple dimensions. Additionally, we explore techniques to optimize the execution of these queries by utilizing indexing structures and advanced operators. The solution was implemented in PostgreSQL and evaluated through experiments that demonstrated its flexibility in processing large volumes of data. The results indicate that SQLSIM enables complex multidimensional analysis in an integrated manner, maintaining scalability and compatibility with standard SQL.

Keywords: Similarity queries. SQL. RDBMS. Clustering. Multidimensional analysis.

Lista de ilustrações

Lista de siglas

BIRCH Balanced Iterative Reducing and Clustering Using Herarchies

CURE Clustering Using Representatives

MAM Método de Acesso Métrico

MBR Minimum Bounding Rectangle

OLAP Online Analytical Processing

SGBD Sistema Gerenciador de Banco de Dados

SGBDR Sistema Gerenciador de Banco de Dados Relacional

SQL Structured Query Language

SGB Similarity Group-By

UDF User Defined Function

Sumário

Capítulo 1

Introdução

Com o avanço tecnológico e a crescente disponibilidade de dados, especialmente multimídia e multidimensionais, emergem desafios significativos relacionados ao processamento, análise e extração de conhecimento. Estes desafios tornam-se ainda mais complexos em aplicações que exigem consultas por similaridade em grandes bases de dados, abrangendo dimensões como características métricas, espaciais e temporais.

Este cenário demanda soluções eficientes e integradas que possibilitem a análise de grandes volumes de dados de maneira escalável, mantendo a precisão e a relevância dos resultados. Dentre os métodos existentes, os Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDRs) apresentam-se como uma base consolidada para armazenar e processar dados estruturados.

No entanto, a integração de algoritmos complexos, como agrupamentos por similaridade, dentro desses sistemas ainda apresenta lacunas significativas, particularmente no que diz respeito à eficiência e à simplicidade de implementação.

Este trabalho se propõe a abordar essas lacunas por meio do desenvolvimento de uma estratégia que possibilite consultas por similaridade diretamente no ambiente do SGBDR, eliminando a necessidade de transferência de dados para processamento externo.

1.1 Motivação

A evolução dos sistemas de informação tem levado a uma crescente demanda por SGBDRs capazes de lidar com dados muito mais complexos do que tipos básicos como números ou cadeias de caracteres. Em um mundo onde imagens, áudios, vídeos, informações georreferenciadas e séries temporais são parte fundamental dos sistemas modernos, as operações tradicionais de comparação baseadas em igualdade e ordem tornam-se insuficientes para atender às necessidades dessas aplicações.

Nesse contexto, consultas por similaridade emergem como uma abordagem essencial para lidar com objetos complexos. Por exemplo, em aplicações de recuperação de imagens

médicas, é importante identificar imagens similares a um caso de referência, permitindo diagnósticos mais precisos e rápidos.

Embora os SGBDRs sejam amplamente utilizados para armazenar e processar dados estruturados, a *Structured Query Language (SQL)*, padrão para manipulação de dados nesses sistemas, não oferece suporte nativo para consultas por similaridade. Isso gera um desafio significativo, pois soluções externas são frequentemente empregadas para implementar essas operações, resultando em um aumento na complexidade operacional, no tempo de processamento e no risco de inconsistências.

Além disso, a crescente adoção de aplicações multidimensionais em áreas como saúde, geoprocessamento e mineração de dados reforça a necessidade de sistemas que integrem consultas por similaridade diretamente no ambiente dos SGBDRs. A ausência dessa integração não apenas limita a eficiência dos sistemas, mas também restringe a capacidade dos usuários de explorar e analisar dados de forma robusta e interativa.

Diante desse cenário, este trabalho aborda essas limitações ao desenvolver uma solução integrada, denominada SQLSIM, para realizar consultas por similaridade diretamente no SGBD. A proposta busca permitir a realização desse tipo de consulta de forma nativa, mantendo os dados no próprio banco de dados durante o processamento.

1.2 Objetivos e Desafios da Pesquisa

O objetivo principal desta pesquisa é apresentar uma abordagem para a execução de consultas *ad hoc* analíticas, onde o critério de agrupamento pode ser baseado na similaridade. Para alcançar este objetivo, foram definidos os seguintes objetivos específicos:

- ❏ Propor, implementar e avaliar a abordagem considerando os contextos métrico, espacial e temporal;
- ❏ Empregar uma abordagem que permite a criação de uma solução integrada no SGBDR, sem a necessidade de extensão da linguagem SQL padrão.

Entre os desafios enfrentados, destacam-se a adaptação da linguagem SQL para operações complexas, a implementação de algoritmos de agrupamento no ambiente relacional utilizando a Linguagem Procedural SQL e a manutenção da eficiência em grandes volumes de dados multidimensionais.

1.3 Hipótese

A hipótese central deste trabalho é que a integração de funções para consultas por similaridade diretamente em SGBDRs pode melhorar significativamente a usabilidade de operações multidimensionais, ao permitir que esse tipo de consulta seja realizado de forma nativa no banco de dados.

1.4 Contribuições

A principal contribuição deste trabalho é a apresentação de uma estratégia para a realização de agrupamentos por similaridade para consultas analíticas utilizando a linguagem SQL padrão. Os agrupamentos por similaridade permitem a criação de tabelas cruzadas ou cubos de dados dos fatos, permitindo a sua visualização multidimensional e as operações *Online Analytical Processing (OLAP)* tradicionais.

Um protótipo de código aberto foi desenvolvido para testar a viabilidade do conceito de integração de operações por similaridade em SGBDRs, eliminando a necessidade de soluções externas ou alteração da linguagem SQL padrão e proporcionando maior eficiência e flexibilidade.

Além disso, este trabalho contribui com a definição de um arcabouço de funções para agrupamentos por similaridade em SGBDRs, oferecendo uma base modular e extensível para futuras adaptações e aprimoramentos.

Essa contribuição estabelece uma base sólida para futuras pesquisas e implementações que explorem a expansão de funcionalidades nativas dos SGBDRs para dados complexos e consultas avançadas, promovendo novas possibilidades de análise de dados.

1.5 Organização da Dissertação

A dissertação está estruturada da seguinte forma:

- Capítulo 1: Introdução, apresenta o contexto, a motivação, os objetivos e as contribuições do trabalho.
- Capítulo 2: Fundamentação teórica, revisa conceitos e métodos relacionados ao tema, incluindo consultas por similaridade, algoritmos de agrupamento e estruturas de dados.
- Capítulo 3: Trabalhos correlatos, detalha os trabalhos correlatos intimamente ligados à proposta evidenciando seus impactos na mesma.
- Capítulo 4: Descrição detalhada do SQLSIM, incluindo sua arquitetura, implementação e integração no PostgreSQL.
- Capítulo 5: Apresenta os experimentos realizados e análise dos resultados obtidos, destacando as vantagens e limitações da abordagem proposta.
- Capítulo 6: Conclusão, sintetiza as principais contribuições, avaliando o alcance dos objetivos e sugerindo direções para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Atualmente, uma quantidade massiva de dados é gerada por sensores, sistemas embarcados e redes sociais. Esses dados, em sua maioria, não são do tipo simples, como inteiros ou *strings*, mas sim dados complexos, tais como vídeos, imagens, áudios e documentos. O processamento desse tipo de dado apresenta desafios significativos, pois sua estrutura, frequentemente multidimensional, torna inviável abordagens baseadas exclusivamente em comparações por igualdade. Assim, busca-se o desenvolvimento de métodos eficientes e escaláveis que permitam análises baseadas em similaridade, atendendo às necessidades de aplicações modernas.

A transformação desses dados em representações vetoriais, mediada por extratores de características específicos, como cor, textura e forma, é um passo crucial para viabilizar comparações quantitativas em diversos cenários. Por exemplo, na recuperação de imagens, esses extratores convertem as características visuais das imagens em vetores, possibilitando o cálculo de similaridade (ou dissimilaridade) por meio de funções de distância, como a Euclidiana ou a Manhattan. Essas funções não apenas fundamentam teoricamente as consultas baseadas em similaridade, mas também desempenham um papel central na definição da escalabilidade e da eficiência de sistemas que operam sobre grandes volumes de dados multimídia. Mais detalhes sobre o tema podem ser vistos em (RAZENTE; LIMA; BARIONI, 2017; LU et al., 2017).

A base teórica para consultas baseadas em similaridade está intrinsecamente ligada ao conceito de espaços métricos. Após a transformação dos dados complexos em representações vetoriais, como ocorre em vídeos, imagens e áudios, é necessário estabelecer um modelo que permita medir quantitativamente a proximidade entre os vetores. Os espaços métricos fornecem esse arcabouço, definindo formalmente as propriedades que as funções de distância devem obedecer, como identidade, simetria, não-negatividade e desigualdade triangular. Essas propriedades são essenciais para otimizar algoritmos de busca, uma vez que possibilitam a aplicação de técnicas de poda e particionamento em grandes volumes de dados.

O domínio dos dados $\mathbb{S}$ e a função de distância $\delta()$ definem um espaço métrico quando

$\delta()$ satisfaz as seguintes propriedades para quaisquer elementos x, y, $z \in \mathbb{S}$:

- $\delta(x, x) = 0$ (*identidade*);
- $\delta(x, y) = \delta(y, x)$ (*simetria*);
- $0 \leq \delta(x, y) < \infty$ (*não-negatividade*);
- $\delta(x, y) \leq \delta(x, z) + \delta(z, y)$ (*desigualdade triangular*)

Entre as funções de distância mais utilizadas estão as da família Minkowski ou métricas L_p, que são aplicadas em domínios multidimensionais, dentre elas destacam-se as distâncias Manhattan (L_1), Euclidiana (L_2) e Chebychev (L_∞). A desigualdade triangular é uma propriedade importante pois permite a criação de algoritmos e estruturas de dados eficientes para a organização e particionamento do espaço dos dados. Essa propriedade é crucial para algoritmos de busca por similaridade, pois permite descartar partições inteiras de conjuntos de dados previamente particionados durante a busca por elementos similares a um elemento de referência, reduzindo significativamente o esforço computacional. Extensas revisões sobre a indexação métrica podem ser encontradas em (SAMET, 2006; MANOLOPOULOS et al., 2005; CHEN et al., 2023).

2.1 Algoritmos de Agrupamento

O agrupamento de elementos similares é um problema com grande utilidade em diversos tipos de análise. O objetivo de um processo de agrupamento é dividir os elementos de um conjunto de dados em grupos de elementos similares, de modo que cada grupo seja composto de elementos mais similares entre si e dissimilares dos elementos de outros grupos (HAN; KAMBER; PEI, 2011). Os métodos de agrupamento podem ser divididos em hierárquicos e por particionamento.

2.1.1 Algoritmos de Agrupamento Hierárquico

Um algoritmo de agrupamento hierárquico produz uma hierarquia de partições, organizada de forma aninhada, que pode ser visualizada como uma árvore ou dendrograma. Essa hierarquia permite a divisão dos dados em múltiplos níveis, variando desde um único grupo contendo todos os elementos até o caso onde cada elemento é tratado como um grupo. Essa abordagem é particularmente útil para explorar a estrutura interna dos dados e identificar padrões em diferentes escalas.

Entre os algoritmos hierárquicos, os principais são:

- Aglomerativos (ou *bottom-up*): cada elemento é tratado como um grupo separado. Em seguida, os grupos mais próximos são combinados iterativamente até que todos os elementos formem um único *cluster*.

- Divisivos (ou *top-down*): começam tratando todos os dados como um único grupo e, a cada etapa, dividem os grupos maiores em *cluster* menores com base em critérios de similaridade.

Exemplos e melhorias dos algoritmos aglomerativos podem ser vistas nos métodos *Clustering Using Representatives (CURE)* em (GUHA; RASTOGI; SHIM, 1998) e *Balanced Iterative Reducing and Clustering Using Herarchies (BIRCH)* em (ZHANG; RAMAKRISHNAN; LIVNY, 1996). O algoritmo CURE faz uso de múltiplos representantes para cada *cluster* para reduzir a sensibilidade a *outliers* melhorando assim a robustez do processo de agrupamento. O BIRCH foi projetado para lidar com grandes conjuntos de dados, tirando proveito da estrutura de árvores para particionar os dados eficientemente.

2.1.2 Algoritmos de Agrupamento por Particionamento

Os algoritmos de particionamento buscam dividir o conjunto de dados em exatamente k grupos, onde k é um valor pré-definido. Diferente dos métodos hierárquicos, esses algoritmos não produzem uma estrutura em múltiplos níveis, mas sim uma única divisão que otimiza um critério específico, como a minimização da soma das distâncias dos elementos ao centróide de seus respectivos *cluster*.

Os algoritmos de particionamento mais conhecidos são:

- *k-means*: Divide os dados em k *clusters*, minimizando a soma das distâncias quadradas entre os pontos de cada *cluster* e o centróide correspondente.
- *k-medoids*: Similar ao *k-means*, mas usa um ponto real do *cluster* como seu representante, ao invés de calcular a média para cada dimensão.

O algoritmo *k-means* (JAIN, 2010) recalcula os centróides de maneira iterativa até que os *clusters* se estabilizem. Algumas implementações possuem um número pré-definido de iterações que, quando atingido, param a execução, retornando os *clusters*. É um algoritmo amplamente utilizado devido à sua simplicidade e eficiência, mas é sensível a *outliers* e requer que os *clusters* sejam aproximadamente esféricos. Por sua vez, o algoritmo *k-medoids* utiliza pontos reais do *cluster* como representante em uma tentativa de minimizar o efeito dos *outliers* fazendo um *trade-off* entre robustez e custo computacional.

Os algoritmos de particionamento apresentam diversas vantagens, como a simplicidade de implementação e eficiência para conjuntos de dados de tamanhos moderados. No entanto, sua dependência da escolha inicial de k e a sensibilidade às condições iniciais podem afetar a qualidade dos agrupamentos. Estratégias como a execução do algoritmo múltiplas vezes com diferentes inicializações ou critérios de validação como o índice de silhueta podem ajudar a mitigar esses problemas.

Uma vantagem dos métodos hierárquicos sobre os métodos de particionamento é que aqueles não requerem a definição prévia do número de *clusters*, mas sim um critério de

corte no dendrograma para determinar a granularidade desejada. No entanto, sua principal limitação é o custo computacional elevado, com a complexidade podendo chegar a $O(n^3)$ nos casos mais simples ou a $O(n^2\ log\ n)$ quando se utilizam estruturas de dados otimizadas, que pode se tornar proibitivo para conjuntos de dados muito grandes, especialmente em versões que exigem cálculos exatos de distância entre todos os pares de elementos.

2.2 O Modelo Relacional

Os bancos de dados relacionais são fundamentados no modelo relacional, que organiza os dados em tabelas (ou relações) dentro de um esquema lógico bem definido (ELMASRI; NAVATHE, 2011). Nesse modelo, cada tabela representa uma entidade enquanto cada linha da tabela, chamada de tupla, corresponde a uma instância específica dessa entidade. As colunas, por sua vez, são denominadas atributos e representam as propriedades ou características da entidade descrita.

Uma das principais vantagens do modelo relacional é sua capacidade de garantir consistência e integridade dos dados por meio de restrições como chaves primárias, que identificam unicamente cada tupla em uma tabela, e chaves estrangeiras, que estabelecem relacionamentos entre tabelas. Além disso, o modelo relacional é suportado por uma linguagem declarativa padrão, o SQL, que permite a manipulação eficiente dos dados armazenados e consultas complexas para recuperação e análise.

A flexibilidade e escalabilidade dos bancos de dados relacionais os tornaram a base de muitos sistemas computacionais modernos. Contudo, o crescente volume e diversidade de dados em aplicações contemporâneas, como dados não estruturados e consultas baseadas em similaridade, têm exigido adaptações e extensões ao modelo relacional tradicional; por isso, muitos SGBDs como o PostgreSQL utilizam bibliotecas e plugins para oferecer suporte a vários tipos de dados avançados e indexação extensível. Em (SIQUEIRA et al., 2018) são apresentadas várias opções de trabalhar a busca por similaridade dentro da SQL, cada uma com suas vantagens e desvantagens.

A partir da década de 1980, surgiram os Sistemas OLAP, projetados para complementar bancos de dados relacionais, atendendo às demandas por análises complexas e consultas rápidas em grandes volumes de dados. Esses sistemas são amplamente utilizados em aplicações de *business analytics*, onde a eficiência e a capacidade de explorar dados detalhadamente são essenciais para a tomada de decisões estratégicas.

O processamento OLAP é caracterizado pela sua habilidade de lidar com dados agregados e organizados em estruturas multidimensionais, como os cubos de dados (ou hipercubos). Após a coleta e tratamento inicial dos dados, eles são geralmente armazenados em *Data Warehouses*, que funcionam como repositórios centralizados. Esses dados podem então ser transferidos e transformados em sistemas OLAP para serem filtrados e

organizados de forma a facilitar a sua análise, focando exclusivamente nos aspectos mais relevantes para um determinado estudo ou decisão.

Os cubos OLAP são particularmente úteis em análises multidimensionais, permitindo a exploração dos dados sob diferentes perspectivas. Cada dimensão de um cubo representa um atributo ou variável, como tempo, localização ou produto, enquanto os níveis hierárquicos dessas dimensões oferecem diferentes granularidades, permitindo que os analistas ajustem a profundidade das consultas de acordo com suas necessidades. Por exemplo, a dimensão tempo pode ser explorada em níveis como ano, trimestre, mês ou dia, dependendo do nível de detalhamento desejado.

Em aplicações de *business analytics*, os cubos de dados são extensivamente utilizados devido a vantagens como:

- ❏ Velocidade: A organização multidimensional permite consultas rápidas, mesmo em grandes volumes de dados.
- ❏ Facilidade de uso: Ferramentas OLAP frequentemente oferecem interfaces intuitivas para que analistas explorem os dados sem necessidade de conhecimento técnico profundo.
- ❏ Capacidade de lidar com grandes volumes de dados: Sistemas OLAP são projetados para agregar e resumir informações de forma eficiente.

Em (SIQUEIRA et al., 2018), os autores estudam as diferentes abordagens ao problema de armazenar e consultar dados complexos em bancos de dados relacionais usando SQL. Eles comparam quatro tipos diferentes de armazenamento, sendo eles: binário, relacional, objeto-relacional e semi-estruturado, avaliando seus impactos no desempenho de consultas por similaridade. Além disso, apresentam uma modelagem abrangente para funções de distância, integrando operadores ao processador de consultas do SGBDR, sem prejudicar o processo de otimização.

Os autores também discutem diferentes métodos para realizar buscas por similaridade, destacando a importância das noções de dissimilaridade e das métricas associadas. Além disso, propõem maneiras de expandir a linguagem SQL para incorporar operadores de similaridade de forma nativa, possibilitando um suporte mais robusto para consultas baseadas em métricas de distância, ou sugerem a utilização de métodos externos como alternativa.

2.3 Estruturas de Dados para Organização por Similaridade

Em sistemas de bancos de dados, os termos “objeto”, “elemento” e “elemento de referência” são conceitos fundamentais e possuem nuances importantes. O termo objeto

refere-se a uma entidade ou item sobre o qual se deseja manter informações. Em consultas tradicionais, um objeto é frequentemente representado por um conjunto de atributos escalares, como números, *strings* ou datas, onde relações de igualdade e ordem são suficientes para a recuperação de informações.

No entanto, no contexto de consultas por similaridade, os objetos podem ser significativamente mais complexos, incluindo tipos como imagens, áudios, séries temporais ou documentos textuais. Nesses casos, a comparação direta por igualdade se torna inadequada, e é necessário medir a similaridade entre os objetos usando funções de distância.

O termo elemento é frequentemente usado como sinônimo de objeto em muitos contextos, mas com um escopo mais específico. Um elemento refere-se a uma ocorrência individual de um objeto dentro de um conjunto de dados.

O elemento de referência, também chamado de *query object* ou objeto de consulta, é um elemento específico usado como base para comparação em consultas por similaridade. Ele serve como o ponto de partida para a busca, e a consulta tem como objetivo recuperar os elementos mais similares a ele.

Em uma consulta de seleção por similaridade, o elemento de referência é único. Nas consultas por abrangência, o elemento de referência é usado para encontrar todos os elementos dentro de um raio especificado. Na consulta aos k vizinhos mais próximos, o elemento de referência é usado para encontrar os k elementos mais similares a ele.

A escolha do elemento de referência é crucial para definir a consulta por similaridade. Essa escolha do elemento de referência e da função de distância a ser usada influenciam diretamente o resultado da consulta, permitindo a recuperação dos elementos mais relevantes de acordo com um critério de similaridade definido.

Consultas por similaridade é o termo mais geral para uma gama de consultas que recuperam os elementos que são menos dissimilares em relação a um elemento de referência. São normalmente utilizadas em grandes repositórios de dados onde, apesar de poderem ser recuperados por suas chaves, em geral os requisitos das aplicações requerem a recuperação por similaridade, pela qual não há uma ordenação natural, sendo a similaridade entre pares de objetos o principal meio de comparação.

No contexto de consultas por similaridade, cada elemento do conjunto de dados é representado em um espaço métrico apropriado, e a função de distância é então utilizada para calcular a proximidade entre dois elementos nesse espaço. A escolha da função de distância é crítica, pois depende das características do domínio e da natureza dos dados. Para dados numéricos vetoriais, podemos usar as funções da família Minkowski, entre elas as distâncias Manhattan, euclidiana e norma suprema, entre outras (SAMET, 2006; CHEN et al., 2023).

A distância calculada entre dois elementos é utilizada como base para comparação, permitindo determinar quão similares eles são. Quanto menor a distância, maior a similaridade entre os elementos.

As duas principais consultas por similaridade são consulta por abrangência e consulta dos k vizinhos mais próximos. Nas consultas por abrangência, buscamos todos os pontos que estão a uma distância menor ou igual ao limite estabelecido em relação ao elemento de referência. Enquanto as consultas dos k vizinhos mais próximos retornam os k elementos mais próximos, não importando a que distância estão do elemento de referência.

2.3.1 Métodos de Acesso

Para realizarmos buscas por similaridade de maneira eficiente, foram criados os Métodos de Acesso, que consistem em técnicas para manipulação e organização de informações em uma base de dados. Através deles, definimos a estrutura de armazenamento assim como as formas de consulta, escrita e exclusão dos dados nas chamadas árvores.

Existem variados Métodos de Acesso, cada um tem sua própria estrutura e particularidades, sendo mais indicados em casos específicos por terem sido construídos com esse propósito. Mas todos tentam lidar com o tamanho crescente das bases de dados, alta dimensionalidade e diversidade nas funções de distância.

A escolha da estrutura ideal depende de fatores como o tipo de consulta (por abrangência ou k vizinhos mais próximos) e as características do espaço métrico. É possível também estabelecer uma forma de indexação dos dados para tornar a consulta mais rápida e eficiente.

2.3.1.1 Métodos de Acesso Métrico

O pior caso para o problema das consultas por similaridade deve ser o acesso sequencial ao conjunto de dados, de modo que uma passagem por todos os objetos é suficiente para encontrar os resultados tanto da consulta por abrangência quanto da consulta aos k vizinhos mais próximos, sem a necessidade de qualquer organização específica. Os Métodos de Acesso Métrico (MAMs) são estruturas projetadas para organizar os dados com base em uma métrica predefinida, geralmente representada por uma função de distância. O objetivo desses métodos é armazenar os elementos de forma que os mais similares entre si estejam localizados próximos dentro da estrutura, facilitando a recuperação eficiente de informações em consultas, com custo computacional menor que o do acesso sequencial (ZEZULA et al., 2006).

Durante a execução de uma consulta, o algoritmo não precisa percorrer todos os elementos da base de dados, haja vista que os MAMs aplicam um mecanismo de filtragem onde apenas os elementos potencialmente relevantes, ou seja, aqueles mais próximos ao elemento de referência, são considerados na busca.

Os demais são descartados por um processo de poda, que reduz significativamente o espaço de busca ao eliminar regiões da estrutura onde a resposta não pode estar. Os métodos de acesso propostos, em geral, sofrem com a maldição da dimensionalidade (KORN;

PAGEL; FALOUTSOS, 2001), que resulta na degradação dos algoritmos de otimização com o aumento no número de dimensões do conjunto de dados.

Uma estrutura típica dos Métodos de Acesso Métrico são as *M-trees*. Essas estruturas são árvores balanceadas que lidam bem com dados dinâmicos e não precisam de reorganização recorrente (CIACCIA; PATELLA; ZEZULA, 1997), são armazenadas em memória secundária e seus índices são construídos previamente. O objetivo dessas estruturas é criar uma indexação de modo que as buscas por similaridade sejam respondidas mais rápido utilizando a desigualdade triangular para auxiliar no processo de poda da árvore, agilizando assim a consulta. No caso da *M-tree*, as distâncias são pré-calculadas e armazenadas na árvore de tal forma que tanto o custo de leitura quanto o de escrita sejam reduzidos, como também o custo computacional da distância.

As pesquisas acerca da *M-tree* têm buscado formas de melhorar o seu desempenho geral assim como a sua adaptação para os diferentes tipos de dados. Técnicas de pré-processamento também foram propostas para lidar com dados de alta dimensionalidade, como na *PM-trees* (LOKOC et al., 2014).

Outros exemplos de árvores muito utilizadas são as *VP-trees* (YIANILOS, 1993) e *MVP-trees* (BOZKAYA; OZSOYOGLU, 1999). Essas árvores utilizam pontos de vista (*pivots*) para particionar os dados em grupos de elementos similares e dissimilares ao *pivot*. Dessa forma, os dados já ficam indexados de maneira que a busca é bem mais rápida. Tais árvores são estruturas estáticas de construção rápida que são armazenadas na memória principal e normalmente utilizadas para consultas *ad hoc*.

2.3.1.2 Métodos de Acesso Espaciais

Os Métodos de Acesso Espaciais desempenham um papel fundamental na recuperação eficiente de informações baseadas na localização de objetos. Eles são amplamente utilizados em aplicações que envolvem dados geoespaciais, como navegação por satélite, rastreamento de veículos e sistemas de informações geográficas (SIG).

Diferentemente dos métodos tradicionais, essas estruturas são projetadas para lidar diretamente com coordenadas e formas geométricas, otimizando operações espaciais, como consultas por proximidade, interseção e inclusão dentro de uma determinada região. Entre as diversas abordagens existentes, destacam-se as *R-trees* (GUTTMAN, 1984), que oferecem uma organização hierárquica eficiente para a indexação e busca de dados espaciais.

As *R-trees* são uma das estruturas de dados mais utilizadas para indexação espacial em bancos de dados por permitirem uma busca eficiente em conjuntos de dados geográficos e multidimensionais. Essa estrutura organiza os elementos em regiões retangulares hierárquicas, onde cada nó interno armazena um retângulo delimitador mínimo, *Minimum Bounding Rectangle (MBR)*, que engloba todos os seus filhos.

Esse formato facilita operações espaciais como consultas de abrangência, interseção e vizinhança, pois permite descartar grandes porções do espaço durante a busca, reduzindo o número de comparações necessárias. Como veremos no Capítulo 4, essas operações serão utilizadas através da biblioteca PostGIS e serão de suma importância.

Uma característica essencial da *R-tree* é sua construção dinâmica e balanceada, semelhante a árvores B, onde novos elementos são inseridos de forma a manter a eficiência da estrutura. Para evitar sobreposição excessiva entre os MBRs e melhorar o desempenho das consultas, diversas variações foram propostas. Entre elas, a R^**-tree* aprimora o balanceamento e minimiza a sobreposição entre regiões, otimizando o desempenho em cenários de alta densidade de dados. Já a R^+*-tree* modifica a estrutura para evitar sobreposições de MBRs em nós internos, garantindo que cada elemento pertença a um único nó, tornando buscas exatas mais eficientes. Revisões sistemáticas dos métodos de acesso métricos e espaciais podem ser obtidas em (SAMET, 2006; MANOLOPOULOS et al., 2005).

Capítulo 3

Trabalhos Correlatos

Neste capítulo, é apresentada uma revisão de trabalhos que tratam sobre agrupamento por similaridade no contexto de bancos de dados relacionais, abordando aspectos fundamentais como eficiência computacional, integração com SQL e otimização de consultas em grandes volumes de dados.

A revisão de trabalhos correlatos é essencial para posicionar esta pesquisa no contexto das contribuições existentes, de forma que possamos compreender quais avanços ainda são necessários, e destacar sua relevância frente às soluções previamente propostas.

Para isso, este capítulo apresenta e discute pesquisas que tiveram impacto significativo na área, seja por sua relevância ao longo dos anos, seja por sua inovação ao introduzir novas abordagens.

3.1 Expansão e Otimização de Consultas SQL

3.1.1 Operadores Avançados para Consultas Agregadas

O operador GROUP BY permite agregar dados com base em uma ou mais colunas, criando sumarizações úteis para análises estatísticas e geração de relatórios. Isso é particularmente interessante porque facilita a identificação de padrões e tendências em grandes conjuntos de dados, como totalizações por categoria, médias de desempenho e contagens de ocorrências. No entanto, sua limitação surge quando há a necessidade de explorar múltiplas perspectivas simultaneamente, pois ele só permite a agregação em um único nível de granularidade por vez.

Pensando nisso, (GRAY et al., 1997) propuseram o operador *Data Cube*, uma generalização do GROUP BY que possibilita a agregação de dados em múltiplas dimensões de forma simultânea. Em vez de produzir apenas uma tabela com as agregações solicitadas, o *Data Cube* gera todas as combinações possíveis das colunas especificadas, formando uma estrutura hierárquica que permite análises mais profundas e flexíveis.

O artigo introduz dois conceitos principais relacionados ao *Data Cube*:

- CUBE *Operator*: Um operador que estende GROUP BY, permitindo a agregação em todas as combinações possíveis das colunas especificadas.
- ROLLUP *Operator*: Um caso especial do *Data Cube*, que permite a agregação hierárquica ao longo de uma única dimensão, útil para relatórios estruturados em níveis como tempo (ano → mês → dia) ou organização (país → estado → cidade).

Além disso, os autores demonstram que os cubos de dados podem ser tratados como relações dentro do modelo relacional, permitindo consultas SQL mais complexas e a integração desses operadores diretamente no SQL padrão.

A importância do *Data Cube* reside no fato de que ele otimiza análises exploratórias de dados, sendo amplamente utilizado em aplicações OLAP por exemplo. Ele melhora significativamente o desempenho de consultas que requerem agregações complexas em grandes volumes de dados. Simplificando operações complexas de GROUP BY que agora podem ser alcançadas por simples *roll-up* ou *drill-down*.

Os autores (ORDONEZ; CHEN, 2012) trazem uma agregação horizontal com a proposta de completar a linguagem SQL para também suportar essa direção de agregação. Como foi apontado, essa é uma tarefa comum na preparação dos dados e, para tanto, foram testados e comparados 3 métodos de agregação, sendo eles CASE, SPJ e PIVOT. O método SPJ não é escalável, enquanto os demais são.

A agregação vertical gera apenas um único valor enquanto a agregação horizontal gera um vetor de valores. Isso se traduz em uma tabela com poucas linhas (resultado da agregação) e várias colunas. O método de agregação horizontal proposto pode ser usado como um método para gerar automaticamente resultados eficientes para consultas SQL com três tipos de parâmetros: agrupamentos de colunas, subagrupamento de colunas e agregação de colunas.

3.1.2 Algoritmos de Agrupamento no SQL

É apresentada em (GARCIA-ALVARADO; ORDONEZ, 2015) uma ideia de como usar uma agregação *k-means* para acelerar agrupamento em bancos de dados muito grandes e com alta dimensionalidade. Essa solução se faz necessária pois, para resolver agregação baseada em algoritmos de particionamento espacial, são feitos vários acessos na base que ficam mais caros conforme a base cresce. Assim, os autores empregaram o algoritmo proposto *K-means incremental* para criar um agrupamento em apenas um acesso à base e esses *clusters* que serão usados para resolver o GROUP BY. Apesar de ser um método aproximado, é mais eficiente e permite a criação das agregações baseadas nos resultados de algoritmos de agrupamento por particionamento, apresentando-se como um modo mais natural de interpretar os dados do GROUP BY nos *clusters* gerados, do que na base toda.

Os dados percentuais são muito usados na análise de dados e, em virtude disso, (ZHANG et al., 2019) propuseram um método de gerar o cubo dos valores percentuais dos

dados. Contudo, o algoritmo requer várias passagens pelos dados para gerar o cubo completo com todas as dimensões dos dados. Apesar disso, os experimentos demonstraram um bom desempenho para o caso de consultas individuais.

3.1.2.1 *Single Pass Clustering*

O *Single Pass Clustering*, ou agrupamento de passagem única, é uma abordagem utilizada para processar grandes volumes de dados de maneira eficiente. Diferente dos métodos iterativos tradicionais, que realizam múltiplas passagens sobre os dados, os algoritmos *single pass* atribuem registros a grupos de forma incremental em uma única varredura dos dados, o que pode reduzir significativamente o custo computacional.

Vários trabalhos exploraram e aprimoraram os métodos de *single pass clustering*. Entre eles, (FANG et al., 2017), que propõe um algoritmo distribuído de agrupamento em tempo real baseado no *framework Storm*. Esse método melhora a detecção e rastreamento de tópicos ao dividir o processo de *clustering* em duas etapas: a primeira executa agrupamentos em múltiplas threads paralelas, e a segunda mescla os grupos gerados, atualizando os *clusters* globais. Os resultados demonstram que essa abordagem mantém a precisão dos métodos tradicionais, mas melhora significativamente a eficiência computacional ao escalar para múltiplos nós de processamento.

Outro avanço relevante é o algoritmo de *Single-Pass Pivot* para *Correlation Clustering*, apresentado por (CHAKRABARTY; MAKARYCHEV, 2023). Esse método é um refinamento do algoritmo *Pivot*, oferecendo uma boa aproximação e otimizando o uso de memória. A abordagem permite lidar com grandes conjuntos de dados de forma eficiente, utilizando uma única passagem sobre as arestas do grafo e mantendo apenas um subconjunto de vizinhos mais relevantes na memória.

3.2 Similaridade no Ambiente Relacional

3.2.1 Extensões do GROUP BY para Agrupamento por Similaridade

O operador GROUP BY organiza dados em grupos baseados em valores comuns, permitindo análises agregadas, como somas ou contagens. É indispensável para a sumarização eficiente e análise de grandes conjuntos de dados. Devido à sua importância, em (TANG et al., 2016), os autores apresentam uma nova estratégia para os casos de agregações baseadas em similaridade na linguagem SQL, o *Similarity Group-By (SGB)*. O trabalho destaca a importância de estender o operador GROUP BY para lidar com agrupamentos que considerem similaridade entre valores, ampliando sua aplicação em cenários onde a igualdade não é suficiente para representar adequadamente os dados.

O agrupamento padrão no SQL opera com base na igualdade entre atributos, o que muitas vezes não reflete a natureza dos dados do mundo real, onde valores podem ser apenas próximos ou semelhantes. Além disso, os autores também levantam a discussão sobre a multidimensionalidade, pois as soluções existentes para agrupamento por similaridade geralmente são projetadas para dados unidimensionais e tratam atributos multidimensionais de forma independente, o que ignora correlações entre eles. Essa limitação é crítica em aplicações onde atributos multidimensionais estão intimamente ligados.

Um outro problema levantado é o fato de precisarmos mover dados para fora do banco de dados para aplicar algoritmos de *clustering* e, em seguida, reimportá-los. O que cria um *impedance mismatch* significativo, tornando o processo caro e ineficiente. Para resolver esses problemas, os autores propõem dois novos operadores de agrupamento por similaridade: DISTANCE-TO-ALL e DISTANCE-TO-ANY, que operam diretamente dentro do banco de dados, por meio de uma extensão na sintaxe da linguagem SQL.

Existe também um tratamento de sobreposição projetado para lidar com tuplas que possam pertencer a mais de um grupo. Isso é importante pois, sem um mecanismo bem definido para gerenciar essas ocorrências, os resultados dos agrupamentos podem se tornar ambíguos ou inconsistentes, comprometendo a qualidade das análises.

Os autores implementaram 3 opções para lidar com as sobreposições, são elas:

- JOIN-ANY: nesta abordagem, o elemento sobreposto é alocado arbitrariamente a um dos grupos que satisfazem os critérios de similaridade. Útil em cenários onde a precisão não é uma prioridade, mas o desempenho deve ser maximizado.

- ELIMINATE: o elemento sobreposto é removido dos agrupamentos. Isso resulta em grupos mutuamente exclusivos e evita qualquer ambiguidade causada pela duplicidade. Ideal para situações em que a exclusividade dos grupos é essencial.

- FORM-NEW-GROUP: nesta opção, os elementos sobrepostos são colocados em um grupo separado. Isso assegura que a informação não seja descartada, mas ainda mantém a exclusividade entre os grupos originais.

A estratégia SGB-All agrupa tuplas de dados em um espaço multidimensional, garantindo que todos os elementos de um grupo estejam dentro de uma distância predefinida uns dos outros. Essa estratégia é útil quando se deseja que cada elemento do grupo tenha um alto grau de proximidade com os demais, já que o elemento só pode pertencer ao grupo se estiver próximo o suficiente de **todos** os outros elementos do mesmo. Por isso, essa estratégia resulta em grupos mais homogêneos e compactos.

A Listagem 3.1 apresenta a semântica para consultas com o operador DISTANCE-TO-ALL.

Listagem 3.1 – Consulta com SGB-ALL.

```
SELECT column, aggregate-func(column)
```

```
      FROM table-name
     WHERE condition
  GROUP BY column DISTANCE-TO-ALL [L2|LINF]
    WITHIN ε
ON-OVERLAP [JOIN-ANY|ELIMINATE|FORM-NEW-GROUP];
```

Já a estratégia SGB-Any relaxa a restrição de proximidade, permitindo que tuplas pertençam a um grupo se estiverem dentro da distância especificada de pelo menos um elemento do grupo. Isso significa que a formação dos grupos se dá por conexões indiretas, o que pode resultar em grupos mais amplos e menos rígidos.

A Listagem 3.2 apresenta a semântica para consultas com o operador DISTANCE-TO-ANY.

Listagem 3.2 – Consulta com SGB-ANY.

```
SELECT column, aggregate-func(column)
  FROM table-name
 WHERE condition
GROUP BY column DISTANCE-TO-ANY [L2|LINF]
 WITHIN ε;
```

O *framework* desenvolvido foi implementado no PostgreSQL e utiliza estratégias de filtro e refinamento para identificar candidatos a agrupamento e emprega índices espaciais como *R-tree* para otimizar o processamento dos dados.

A utilização de índices espaciais como a *R-tree* é especialmente importante, pois permite uma abordagem baseada em regiões delimitadoras. Dada a métrica escolhida pelo usuário, primeiramente, o SGB se preocupa em definir a topologia do grupo. De forma que essa região pode ser representada por um retângulo delimitador mínimo (MBR) ou por outras formas geométricas que encapsulam os elementos do grupo de maneira eficiente.

Uma das decisões mais importantes no agrupamento por similaridade é como verificar a inclusão de novos elementos em um grupo já existente. As duas abordagens principais são (*i*) a comparação direta com todos os pontos do grupo e (*ii*) a verificação contra a região que representa o grupo. Os autores também discutem esse *trade-off* pois decidiram optar pela segunda já que essa estratégia reduz o número de comparações necessárias, tornando o processo mais eficiente, mas pode introduzir falsos positivos, uma vez que o MBR da região pode não representar com exatidão todos os seus elementos.

Com isso, a busca é acelerada pelo uso de índices espaciais como *R-trees*, permitindo que um elemento seja rapidamente identificado como pertencente ou não a um grupo sem a necessidade de percorrer todos os seus membros individualmente. Pois, em vez de realizar múltiplas comparações diretas, os algoritmos do SGB verificam se um novo elemento pertence a um grupo testando sua inclusão dentro da região que define a topologia do grupo.

Essa estratégia reduz drasticamente a complexidade computacional e melhora a escalabilidade do método, tornando viável o processamento de grandes volumes de dados multidimensionais.

Os experimentos apresentados pelos autores confirmam que os operadores não apenas superam métodos externos em desempenho, mas também ampliam as capacidades analíticas do SQL tradicional, facilitando novas aplicações em diversas áreas.

O artigo destaca ainda que os operadores propostos são extensíveis, permitindo variações futuras no critério de agrupamento.

3.3 Análise Multidimensional de Dados

3.3.1 Cubo Spatio-Textual-Temporal (STT)

O artigo (IQBAL; LISSANDRINI; PEDERSEN, 2022) propõe um novo modelo para análise de dados multidimensionais que combina informações espaciais, textuais e temporais dentro de um único *framework* analítico. A principal motivação do trabalho é a crescente necessidade de integrar e analisar dados provenientes de diversas fontes, como redes sociais, sensores geográficos e registros históricos, de maneira eficiente e estruturada.

O autores comparam as várias opções para análise de dados espaciais, textuais e temporais existentes até então e fica evidente a deficiência para lidar com múltiplas dimensões.

A análise desse tipo de conjunto de dados é frequentemente realizada de maneira fragmentada. Sistemas de bancos de dados geoespaciais permitem consultas eficientes sobre localização, mas não são projetados para lidar com textos de maneira estruturada.

Da mesma forma, mecanismos de busca textual são eficientes para encontrar informações baseadas em palavras-chave, mas não conseguem lidar adequadamente com aspectos espaciais e temporais.

O problema piora quando se deseja realizar uma análise simultânea das dimensões, pois a maioria dos sistemas não oferece suporte nativo para consultas que envolvem localização, texto e tempo de forma integrada.

Diante desse cenário, os autores propõem o conceito do Cubo Espacial-Textual-Temporal (*Spatio-Textual-Temporal Cube - STT Cube*), que estende a modelagem tradicional de cubos de dados para incorporar dados que possuem uma relação simultânea entre essas três dimensões. A demonstração do cubo STT é feita com uma base de dados de *tweets* onde o objetivo principal é permitir consultas OLAP que possam responder perguntas complexas como:

- "Quais são os principais tópicos discutidos em uma determinada localização e período de tempo?"

- ❏ "Como a popularidade de certas palavras-chave evolui em diferentes regiões ao longo do tempo?"
- ❏ "Quais tendências emergiram em diferentes cidades e qual o impacto espacial delas?"

Pensando em responder essas perguntas, os autores propõem um *framework* analítico baseado em cubos multidimensionais, permitindo que os dados sejam modelados e armazenados de forma otimizada para suportar operações como agregações, *drill-down*, *roll-up* e filtragens específicas. Além disso, os autores exploram técnicas para otimizar o processamento dessas consultas, reduzindo a latência e tornando o modelo escalável para grandes volumes de dados.

A estrutura do cubo STT segue uma abordagem multidimensional, onde os dados são organizados em fatos e dimensões:

- ❏ Fatos: Representam os eventos ou ocorrências registradas no banco de dados, como postagens em redes sociais, registros de sensores ou transações comerciais.
- ❏ Dimensões: Definem os diferentes aspectos dos fatos e permitem a organização dos dados para análise.

Como dito anteriormente, no caso do cubo STT para a base de dados de *tweets*, há três principais dimensões. Sendo elas: Espacial que representa a localização geográfica do *tweet*, Textual que refere-se ao conteúdo textual do *tweet*, e Temporal que representa a data e a hora do *tweet*.

A principal inovação do cubo STT está na forma como essas dimensões são combinadas e estruturadas, permitindo que as consultas analisem interseções entre localização, tempo e texto.

As hierarquias subdividem as dimensões facilitando a análise dos dados em diferentes níveis de granularidade. No caso do cubo STT, as hierarquias são definidas, em ordem decrescente, da seguinte maneira:

- ❏ Hierarquia Textual: categoria, tópico, tema e termo
- ❏ Hierarquia Espacial: país, região, cidade e localização
- ❏ Hierarquia Temporal: ano, trimestre, mês e dia

As hierarquias permitem explorar os dados em diferentes níveis de detalhe e são fundamentais para a aplicação de medidas definidas. Os autores apresentam diferentes medidas que capturam propriedades importantes em cada uma dessas dimensões, e hierarquias, permitindo a comparação e agregação dos dados de maneira estruturada.

Uma das medidas apresentadas é *localização de palavra-chave* que é uma medida das dimensões espaço-texto que retorna a lista de palavras-chave pareadas com as localizações

em que elas aparecem. É importante perceber o papel das hierarquias na aplicação das medidas pois, ao mudarmos o nível de granularidade, isto é, o nível hierárquico, a medida também muda. Podemos então medir as palavras-chave em relação a cidades ou um país inteiro.

A medida de *volatilidade de palavra-chave* é um exemplo de medida que atua nas três dimensões. Essa medida retorna a forma geométrica (da região em questão), a palavra-chave, o intervalo de tempo e a variação na densidade da palavra durante esse intervalo de tempo. Essas informações são utilizadas para calcular a volatilidade da palavra-chave. Muitas outras medidas são apresentadas mas o mais importante é perceber como elas se relacionam com as dimensões e com as hierarquias e como essa possibilidade de medida em várias dimensões simultâneas permite uma análise rica.

Os operadores OLAP *slice*, *dice*, *roll-up* e *drill-down* foram estendidos para o cubo STT. Denominados, pelos autores, como operadores STT-OLAP, os mesmos, em suas versões "STT-" atuam de maneira similar ao seus correspondentes mas com suporte às dimensões e hierarquias, adaptando-se à natureza hetereogênea dos dados. Por exemplo, o operador STT-Roll-Up agrega o valor das medidas em um nível hierárquico partindo de um nível anterior para o próximo.

A materialização do cubo STT é o processo de pré-computação e armazenamento de agregações multidimensionais para otimizar as consultas futuras. Em cubos OLAP tradicionais, todas as combinações possíveis das dimensões podem ser materializadas para garantir tempos de resposta rápidos, mas essa abordagem não é possível no cubo STT devido à grande quantidade de dados.

Para resolver esse problema, os autores propõem estratégias de materialização parcial, armazenando apenas as agregações mais relevantes e que são acessadas com frequência. Isso significa que os níveis de agregação mais utilizados pelos usuários são pré-calculados e armazenados, enquanto outras combinações podem ser computadas apenas se solicitadas.

Por exemplo, se a maioria das consultas analisa tendências semanais em vez de diárias, o sistema pode priorizar a materialização das agregações semanais, reduzindo o custo computacional e o espaço de armazenamento necessário. Além disso, os autores exploram o uso de estruturas de indexação eficientes e técnicas de compressão para armazenar dados textuais, otimizando o desempenho das consultas espaciais e textuais.

Outro ponto de destaque é a manutenção do cubo STT. O desafio aqui é o fato dos dados armazenados serem continuamente atualizados conforme novos *tweets* são registrados. Para lidar com isso, os autores propõem um mecanismo de atualização incremental, no qual as agregações já materializadas são ajustadas com base nos novos dados sem a necessidade de recalcular o cubo inteiro. Essa abordagem reduz drasticamente o tempo de atualização e garante que as consultas sejam executadas de forma eficiente, mesmo com a entrada constante de novos dados. Como resultado, o cubo STT pode operar de maneira escalável, suportando grandes volumes de dados enquanto mantém consultas analíticas

rápidas, otimizadas e atualizadas.

O Cubo STT reduziu o tempo de resposta de consultas em 1 a 5 ordens de magnitude em comparação à abordagem sem materialização. Além disso, houve uma redução de 97% a 99,9% no custo de armazenamento em relação à materialização completa, e o impacto no tempo de construção do cubo que pode ser desprezado. Os cálculos aproximados mantiveram uma precisão entre 90% e 100%, enquanto a manutenção incremental do cubo apresentou melhorias em relação aos métodos tradicionais. E se diferencia ao propor um modelo unificado, que combina hierarquias espaciais, textuais e temporais, além de métricas específicas para essas dimensões combinadas.

3.4 UDFs em Consultas SQL

As Funções Definidas pelo Usuário (*User Defined Functions - UDFs*) são funções criadas dentro de um SGBDR usadas para estender suas funcionalidades. Elas permitem a execução de operações personalizadas que não são suportadas nativamente pelo SQL, facilitando a reutilização de código e a modularização de consultas complexas.

Apesar de suas vantagens, as UDFs podem gerar problemas de desempenho, pois operam em um paradigma procedural, enquanto o SQL é declarativo. Isso pode impedir que o otimizador do SGBD gere planos de execução eficientes, resultando em tempos de resposta elevados para consultas que fazem uso intensivo dessas funções.

3.4.1 Eficiência de UDFs

A principal forma de otimizar UDFs é também a principal causa dos seus problemas de desempenho. A técnica de *inlining* converte a UDF em uma subconsulta SQL equivalente, permitindo que o otimizador do banco de dados decida a forma mais eficiente de processar a função. Embora essa técnica traga benefícios de desempenho, nem sempre o *inlining* completo da função é a melhor solução, pois pode gerar subconsultas muito complexas, dificultando o processo de otimização e execução eficiente das consultas.

Buscando melhorar o desempenho de UDFs, (ARCH et al., 2024) propõem o *framework* PRISM, centrado principalmente em *outlining*, técnica que consiste em decompor a UDF em partes menores, realizando o *inlining* apenas nos trechos que realmente beneficiam o plano de execução da consulta e deixando outras partes separadas. Essa abordagem reduz a complexidade do código gerado, tornando-o mais amigável para os otimizadores de consultas dos SGBDs.

Para otimizar as partes da função onde o *inlining* não é ideal, o PRISM combina outras cinco estratégias de otimização de UDFs:

1. Antecipação de predicados

Consiste em extrair condições de filtragem (predicados) de dentro da UDF para a consulta principal. Isso evita que a função seja executada desnecessariamente para elementos que poderiam ser descartados logo no início da execução da consulta.

Normalmente, as UDFs são chamadas para cada linha do conjunto de dados, o que pode ser ineficiente se a função inclui condições que poderiam eliminar certas tuplas mais cedo. Então mover esses predicados para a cláusula WHERE da consulta principal reduz o número de execuções da UDF permitindo que o SGBD descarte linhas irrelevantes antes mesmo de invocar a função.

2. *Outlining* baseado em regiões

   Essa é a técnica central do PRISM. Ela identifica regiões de código dentro da UDF que podem ser destacadas como funções auxiliares menores, reduzindo a complexidade do código que será tratado com *inlining*. Essa abordagem evita que a consulta gerada se torne excessivamente grande, o que poderia dificultar a otimização pelo banco de dados.

3. Eliminação de Instruções Redundantes

   Tem como objetivo remover operações redundantes dentro da UDF detectando expressões que são computadas mais de uma vez dentro da função e as substitui por uma única avaliação armazenada em uma variável. Além disso, remove operações que não impactam o resultado final, como atribuições desnecessárias.

4. Remoção de Subconsultas Desnecessárias

   Essa estratégia lida com um dos maiores problemas do uso de UDFs: a geração excessiva de subconsultas dentro do SQL gerado. Muitas vezes, quando uma UDF é transformada em SQL, ela pode conter subconsultas aninhadas desnecessárias, que aumentam a complexidade e degradam o desempenho da consulta.

   O PRISM analisa as subconsultas geradas pelo *inlining* e identifica aquelas que não são essenciais. Quando possível, simplifica essas consultas transformando-as em expressões escalares diretas, reduzindo a profundidade do plano de execução: menos subconsultas significa menos operações de junção e menos carga computacional.

5. Reorganização de Consultas

   O PRISM pode reorganizar a posição dos blocos de expressões SQL geradas para garantir melhor aproveitamento dos índices e do plano de execução do SGBD. O PRISM avalia onde as expressões SQL devem ser posicionadas na consulta para maximizar a eficiência da execução. Reposiciona cálculos, junções e agregações para que sejam feitos no momento mais oportuno dentro do pipeline de execução.

Antes de qualquer transformação, o PRISM realiza uma análise detalhada da UDF para entender sua estrutura, separando em partes menores antes de decidir qual técnica aplicar. Essa etapa envolve:

- ❏ Identificar expressões puramente funcionais, que podem ser substituídas diretamente.
- ❏ Detectar dependências entre chamadas de UDFs dentro da consulta.
- ❏ Classificar as operações dentro da UDF em cálculos simples, junções, agregações.

Com essa análise, o PRISM determina quais partes da UDF podem ser otimizadas com *inlining* diretamente e quais devem ser isoladas (*outlining*). O próximo passo consiste em utilizar as estratégias acima, escolhendo dinamicamente qual técnica aplicar em diferentes partes do código para melhorar o desempenho das consultas.

Esse estudo reforça a importância de avaliar criticamente o impacto das UDFs nas consultas SQL e buscar estratégias eficientes para sua utilização.

Capítulo 4

SQLSIM: Uma estratégia para agrupamento por similaridade

Este capítulo apresenta as estratégias estudadas e implementadas para a execução de consultas ad hoc analíticas, nas quais o critério de agrupamento pode ser baseado na similaridade por meio de funções definidas pelo usuário (UDFs) na linguagem procedural SQL. Essa abordagem, denominada SQLSIM, resultou tanto em um arcabouço teórico, que definiu a linguagem e as propriedades comutativa e associativa das agregações no contexto do SQL, quanto em um protótipo funcional, implementado no SGBD de código aberto PostgreSQL.

O SQLSIM tem como objetivo resolver desafios de aplicações que lidam com grandes volumes de dados e alta dimensionalidade nas consultas baseadas em similaridade que demandam não apenas eficiência computacional, mas também uma integração fluida com as bases de dados, eliminando o *impedance mismatch* que ocorre ao transferir dados para processamento externo. A principal inovação do SQLSIM está na eliminação do *impedance mismatch* sem alterar a linguagem SQL, sendo de fácil implementação e permitindo a modificação das funções de similaridade.

Tradicionalmente, algoritmos de *clustering* e agregação são executados fora do banco de dados, exigindo transferência de dados entre diferentes ambientes, o que gera aumento no tempo de execução, riscos de inconsistência e maior complexidade de desenvolvimento. O SQLSIM elimina essa barreira ao integrar diretamente os algoritmos no SGBD, permitindo que todas as operações sejam realizadas dentro do mesmo ambiente.

Uma outra maneira de executar as análises diretamente dentro do SGBD é utilizando bibliotecas e extensões que alterem a linguagem SQL. Essas soluções normalmente são desenvolvidas com um propósito específico e não permitem alterações.

Em contrapartida, o SQLSIM reduz a complexidade do processo de integração de algoritmos de *clustering* aproveitando a linguagem procedural SQL para implementar diretamente as operações necessárias através de funções definidas pelo usuário. Essas funções operam em múltiplas dimensões e podem ser ajustadas às necessidades do usuário.

A abordagem é flexível, permitindo que novos algoritmos e métricas sejam incorporados para o caso de uso específico do usuário sem necessidade de reestruturar o sistema.

4.1 Descrição da Abordagem

A concepção da abordagem se baseou na estrutura de um cubo de dados tridimensional, onde três dimensões principais foram consideradas:

1. Dimensão Espacial: representa a proximidade geográfica dos elementos analisados;
2. Dimensão Temporal: permite a criação de agrupamentos baseados em restrições de tempo arbitrárias;
3. Dimensão Métrica: permite a criação de agrupamentos por similaridade com base em funções de distância.

Cada dimensão é tratada de forma independente, utilizando UDFs para realizar os cálculos de similaridade dentro do SGBD. Essas funções recebem como parâmetros os atributos a serem analisados e um limiar de distância ou similaridade, que determina a granularidade do agrupamento.

A Listagem 4.1 apresenta um exemplo da sintaxe SQL utilizada no SQLSIM:

Listagem 4.1 – Exemplo da sintaxe do SQLSIM.

```
SELECT
       spatial_sim(lat, long, d) as spatial,
       temporal_sim(data, intervalo) as temporal,
       metric_sim(attr1, attr2, attr3, ϵ) as metric,
       count(id)
    FROM TabelaFatos;
GROUP BY
    CUBE(spatial, temporal, metric)
```

Nessa abordagem, cada UDF (*spatial_sim*(), *temporal_sim*() e *metric_sim*()) processa os atributos correspondentes e retorna um identificador que representa o grupo ao qual cada registro pertence. A granularidade dos agrupamentos é controlada por parâmetros ajustáveis, como o limiar de similaridade, ϵ, ou distância, d, no caso das dimensões Métrica e Espacial e o nível de detalhamento do *intervalo* de tempo no caso da dimensão Temporal. O uso da cláusula CUBE processa a combinação das três dimensões permitindo uma análise conjunta das mesmas.

A Figura 1 apresenta um exemplo de cubo de dados tridimensional, ilustrando como as diferentes dimensões foram combinadas para formá-lo.

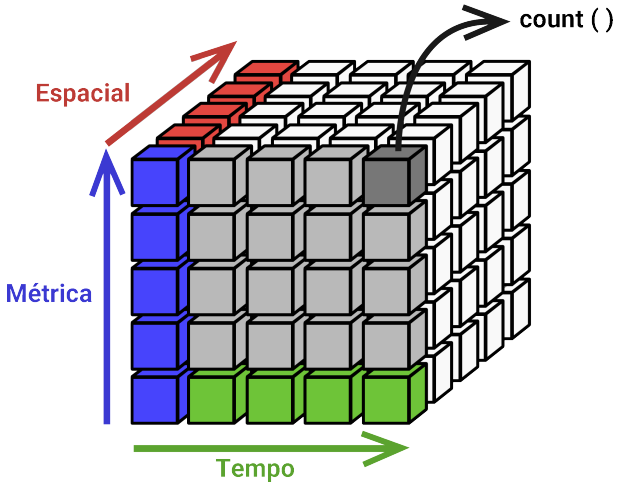

Figura 1 – Exemplo de cubo de dados tridimensional.

No restante desse capítulo utilizaremos como exemplo para apresentar a abordagem o contexto de análise de exames de sangue. Dado um conjunto de exames de sangue, deles podemos extrair os dados sobre a localização da clínica onde o exame foi realizado, a data do exame e também as medidas do exame em si, como glóbulos vermelhos, glóbulos brancos, plaquetas, glicemia, colesterol HDL, colesterol LDL, entre outros.

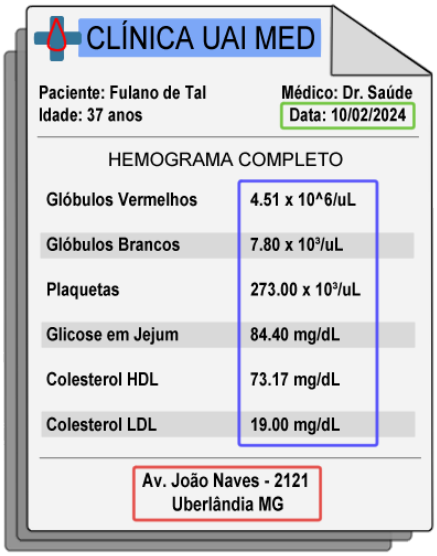

Figura 2 – Exame médico e as dimensões Temporal (verde), Métrica (azul) e Espacial (vermelho).

Os dados do exame de sangue da Figura 2 passam por um processo de extração e transformação para serem inseridos no banco de dados. A localização da clínica é registrada como latitude (*lat*) e longitude (*long*), cada medida do exame é registrada como um atributo separado e a data de realização do exame é registrada em formato de *data*. A escolha do formato em que os dados são armazenados é puramente opcional. Visando deixar a abordagem mais genérica possível, os dados foram armazenados cada um em

um atributo mas também poderiam ser salvos como vetores. Por exemplo, um atributo *localização* no formato de um ponto $(lat,\ long)$ e também um vetor de características $(RBC,\ WBC,\ PLT)$ para os dados dos exames, que nesse caso é tridimensional mas poderia ter outra dimensionalidade.

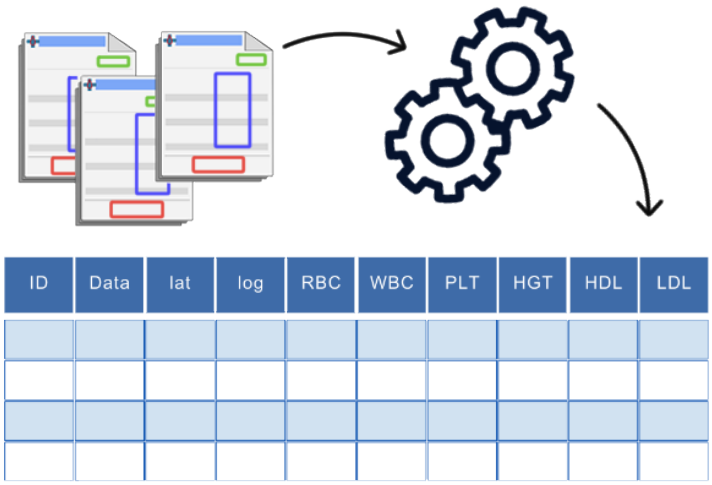

Figura 3 – Extração dos dados dos exames.

A estrutura da tabela de fatos que armazena os dados dos exames seguiu o modelo:

```
CREATE TABLE exames (
    id INTEGER PRIMARY KEY,
    data DATE,
    lat NUMBER,
    long NUMBER,
    globulos_vermelhos NUMBER,
    globulos_brancos NUMBER,
    plaquetas NUMBER,
    glicemia NUMBER,
    hdl NUMBER,
    ldl NUMBER);
```

Com os dados estruturados no banco, foi possível realizar consultas para agrupar exames médicos por similaridade, utilizando a sintaxe da Listagem 4.2.

Listagem 4.2 – Agrupamento na dimensão Métrica utilizando o SQLSIM.

```
SELECT
    metric_sim(g_ver, g_bra, plaq, 3) AS metric,
    count(id)
  FROM exames
GROUP BY metric;
```

Na Listagem 4.2, o agrupamento por similaridade sobre a relação *exames* é gerado pela função *metric_sim*() em conjunto com o operador GROUP BY. A *metric_sim*() recebe como parâmetros: (*i*) os atributos sobre os quais irá operar e (*ii*) um limiar de similaridade (ϵ) para gerar grupos de tuplas não sobrepostos. E retorna um identificador de segmento para cada grupo, de modo que tuplas que compartilham o mesmo identificador pertencem ao mesmo grupo. O operador GROUP BY utiliza os identificadores de grupos retornados pelo SQLSIM para agrupar os dados, nesse caso, pela função *count*().

A Figura 4 ilustra o resultado desse agrupamento, demonstrando como os exames foram particionados com base na similaridade dos resultados laboratoriais.

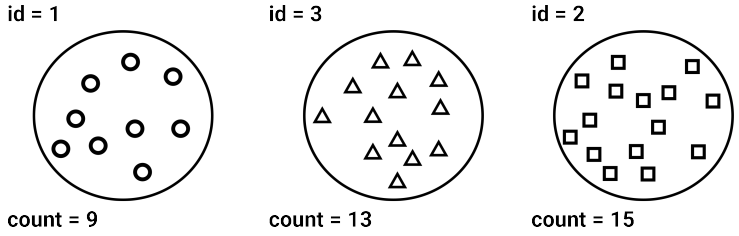

Figura 4 – Exemplo do particionamento dos exames segundo a dimensão Métrica.

4.1.1 UDFs e Tabelas Auxiliares

As UDFs no SQLSIM são responsáveis por particionar os dados em cada dimensão. Três funções principais foram utilizadas:

1. *spatial_sim*(): agrupa os registros considerando coordenadas geográficas;
2. *temporal_sim*(): agrupa os registros com base na proximidade temporal;
3. *metric_sim*(): determina a similaridade entre os registros.

Cada uma dessas funções recebe os atributos da dimensão em questão e um limiar para determinar quais elementos devem ser considerados pertencentes ao mesmo grupo. Como resultado, retornam um identificador de grupo que é utilizado na agregação dos dados.

As tabelas auxiliares desempenham um papel de organização e armazenamento dos resultados intermediários para evitar recálculos desnecessários e permitem que os identificadores dos grupos sejam referenciados posteriormente garantindo que o SQLSIM seja escalável e eficiente, mesmo em cenários com grandes volumes de dados. Além disso, é importante que sejam tabelas temporárias para garantir o isolamento das transações.

4.1.1.1 *metric_sim*()

Nesse exemplo prático, a dimensão Métrica é composta pelos exames e, diferentemente das dimensões Espacial e Temporal, onde os dados possuem um formato bem definido

(coordenadas geográficas e datas, respectivamente), aqui a comparação ocorre diretamente entre os exames em si.

Cada exame é interpretado como um vetor de características no espaço métrico, onde os atributos numéricos capturam diferentes aspectos dos resultados clínicos. Dessa forma, a similaridade entre dois exames pode ser medida com base na proximidade desses vetores em um espaço de alta dimensionalidade.

Embora seja possível utilizar todas as medidas do exame para compor o vetor de características, optamos por definir um vetor menor para a demonstração da abordagem. Assim, utilizamos os atributos *globulos_vermelhos*, *globulos_brancos* e *plaquetas* como representação reduzida dos exames, permitindo a execução da função *metric_sim*() de maneira mais objetiva e interpretável.

A função *metric_sim*() recebe esse vetor de características junto com um limiar de similaridade, determinando quais exames devem ser considerados semelhantes e agrupados. O resultado da função é um identificador de grupo, que indica a qual grupo (*cluster*) cada exame pertence. Na Listagem 4.2 demonstramos esse caso de uso onde o limiar de similaridade de 3 unidades foi escolhido.

A execução da função *metric_sim*() utiliza duas tabelas auxiliares:

- *centroids*: armazena os centróides dos grupos formados;
- *cent_assign*: mantém a relação entre cada registro e seu grupo correspondente, permitindo rastrear as associações.

Listagem 4.3 – Criação da tabela *centroids*.

```
CREATE TABLE centroids (
    id INTEGER PRIMARY KEY,
    center geometry(pointZ));
```

Listagem 4.4 – Criação da tabela *cent_assign*.

```
CREATE TABLE cent_assign (
    obs_id INTEGER PRIMARY KEY,
    centroid integer);
```

A implementação da função *metric_sim*() está descrita no Algoritmo 1 e ela realiza as seguintes etapas ao processar um conjunto de dados:

1. Verifica se a tabela *centroids* está vazia, em caso afirmativo, temos o caso inicial e o ponto é inserido como centróide do primeiro *cluster*;
2. Verifica se a instância já pertence a um *cluster* existente consultando a tabela *cent_assign*;

3. Caso a instância não tenha um *cluster* atribuído, a função chama a função *aggr_metric_sim*() para verificar a proximidade do ponto em relação aos centróides armazenados na tabela *centroids*;

Algoritmo 1 *metric_sim*()

Require: *pid*, *x*, *y*, *z*, *dist*
Ensure: *gid* ▷ *id do grupo*
Declare *gid* (int), *p* (geometry) ▷ *inicializa variáriveis*

$p \leftarrow$ makepoint(x, y, z)

if (count(*) FROM *centroids*) = 0 **then**
    INSERT INTO *centroids*VALUES (1, *p*);
    gid ← 1
else
    gid ← SELECT *centroid* FROM *cent_assign* WHERE *obs_id* = *pid*
    if *gid* ≠ NULL **then**
        return *gid*
    gid ← *aggr_metric_sim*(*p*, *dist*)
INSERT INTO *cent_assign* VALUES (*pid*, *gid*);
return gid

Analisando o Algoritmo 1, é possível perceber que ele não realiza cálculos de proximidade ou similaridade. Isso ocorre porque a função foi projetada apenas para gerenciar a atribuição dos registros aos grupos, enquanto o Algoritmo 2 foi desenvolvido especificamente para lidar com os cálculos de distância. Essa abordagem proporciona um isolamento claro de responsabilidades, melhorando a organização do código e facilitando sua manutenção e extensão.

O Algoritmo 2 descreve a implementação da função *aggr_metric_sim*() e ela realiza os seguintes passos:

1. Se o ponto estiver dentro do limiar de distância, ele é atribuído ao *cluster*;
2. Caso o ponto seja atribuído a um *cluster* existente, seu centróide é atualizado;
3. Se não houver *clusters* próximos, um novo centróide é criado e armazenado na tabela *centroids*, formando um novo agrupamento.

Para decidir em qual grupo o registro em questão irá ingressar, o Algoritmo 2 faz comparações por similaridade seguindo a métrica e o limiar de distância definidos. O processo de *clustering* do Algoritmo 2 é uma implementação simplificada do algoritmo *k-means* adotada no trabalho correlato da Seção 3.2.1 adaptada para uma abordagem de *single pass*, onde os centróides são recalculados dinamicamente. Quando um registro

Algoritmo 2 *aggr_metric_sim*()

Require: p, $dist$
Ensure: gid ▷ *id do grupo*

```
Declare i (int), g (centroids linha)

i ← SELECT count(*) FROM centroids;

for g ← SELECT * FROM centroids do
    if ST_3DDWinthin(g.center, p, dist) then
        UPDATE centroids SET center = ST_MakePoint() WHERE id = g.id
        return gid

INSERT INTO centroids VALUES (i + 1, p)
return i+1
```

ingressa em um grupo existente, Algoritmo 2 atualiza o centróide pela média entre o centróide atual e o ponto ingressante.

É importante destacar que o SQLSIM foi projetado com flexibilidade em mente, permitindo sua adaptação para diversos casos de uso. O Algoritmo 2, responsável pela agregação por similaridade, pode ser substituído e diferentes implementações de algoritmos de *clustering* podem ser integradas ao Algoritmo 1, conforme as necessidades específicas da aplicação.

Além disso, o cálculo do centróide também pode ser ajustado para melhor representar as características do grupo, seja utilizando médias ponderadas, medianas ou outras estratégias que reflitam mais adequadamente o comportamento dos dados em análise. Essa flexibilidade garante que o SQLSIM possa ser personalizado para uma ampla gama de domínios e contextos.

4.1.1.2 *spatial_sim*()

A dimensão Espacial nesse exemplo prático é composta pelas localizações geográficas das clínicas ou hospitais onde os exames foram registrados. Diferente da dimensão Métrica, onde a similaridade é determinada pela proximidade de vetores de características, aqui a comparação ocorre com base na posição dos objetos no espaço físico.

Cada ponto espacial é representado por suas coordenadas geográficas (latitude e longitude) e agrupados de acordo com sua proximidade geográfica, de modo que pontos próximos sejam classificados dentro da mesma região.

O agrupamento espacial no SQLSIM é realizado pela função *spatial_sim*(), descrita no Algoritmo 3, que utiliza a biblioteca PostGIS para facilitar operações geométricas avançadas e a tabela auxiliar *regions*, que armazena as áreas geográficas agrupadas e suas respectivas topologias.

Listagem 4.5 – Criação da tabela *regions*.

```
CREATE TABLE regions (
```

```
    id INTEGER PRIMARY KEY,
    topology geometry(polygon));
```

Entre as operações geométricas advindas da biblioteca PostGIS, destacam-se:

- ❏ *ST_Contains*: Verifica se um ponto está dentro de uma região definida.
- ❏ *ST_Intersection*: Calcula a interseção entre duas regiões.

A função *spatial_sim*() compara cada novo ponto espacial (*lat*, *long*) com as regiões já registradas na tabela *regions*. O processo ocorre da seguinte forma:

1. Verifica se há regiões registradas na tabela regions e cria uma nova região caso não existam registros anteriores;
2. Verifica se o ponto está dentro de alguma região;
3. Se o ponto pertencer a uma região existente, a topologia do grupo é recalculada e atualizada na tabela regions;
4. Se o ponto não pertencer a nenhuma região existente, uma nova região é criada.

Algoritmo 3 *spatial_sim*()

Require: p, dist
Ensure: gid
Declare *circle* (geometry), *lines* (int), *r* (RECORD), *tplg* (geometry)
SELECT count(*) INTO *lines* FROM regions;
circle ← ST_Buffer(*p*, *dist*)
if *lines* = 0 **then**
  INSERT INTO regions VALUES (1, *circle*);
  return 1
for *r* IN SELECT * FROM regions **do**
  if ST_Contains(*r.topology*, *p*) **then**
    tplg ← ST_Intersection(*r.topology*, *circle*)
    UPDATE regions SET *topology* = *tplg* WHERE *id* = *r.id*;
    return *r.id*
INSERT INTO regions VALUES ((*lines* + 1), *circle*);
return *lines* + 1

O Algoritmo 3 é uma simplificação em relação aos algoritmos da dimensão Métrica, refletindo a menor complexidade intrínseca da dimensão Espacial. Diferentemente do que ocorre com os dados métricos, o conceito de similaridade aqui é tratado como proximidade geográfica, o que permite uma implementação mais direta. Apesar disso, a estrutura do algoritmo é flexível, possibilitando seu desmembramento em múltiplas etapas, caso análises mais detalhadas sejam necessárias.

Na Listagem 4.6, a localização das clínicas é utilizada para agrupá-las por proximidade dentro de um raio de 5 unidades. Essa consulta retorna as clínicas organizadas em grupos geográficos, permitindo uma análise eficiente da sua distribuição regional. A granularidade do agrupamento pode ser ajustada alterando o valor do raio de proximidade, o que permite explorar agrupamentos mais compactos ou mais abrangentes.

Listagem 4.6 – Agrupamento por similaridade espacial.

```
SELECT
      spatial_sim(id, lat, long, 5) AS spatial,
      count(id)
  FROM exames
GROUP BY spatial;
```

A flexibilidade do SQLSIM também é evidenciada aqui. Assim como na dimensão Métrica, o Algoritmo 3 pode ser modificado para atender a diferentes necessidades. Por exemplo, podem ser utilizadas métricas alternativas de agrupamento geográfico ou diferentes métodos de manipulação de topologias. A biblioteca PostGIS, embora amplamente utilizada para operações espaciais, não é obrigatória, e o método pode ser implementado em outros SGBDs com suporte a operações geográficas.

4.1.1.3 *temporal_sim*()

A dimensão Temporal é responsável por agrupar dados com base em períodos de tempo, como dias, semanas, meses ou anos, e, nesse exemplo prático, representa a data em que o exame foi realizado.

Diferentemente das dimensões Métrica e Espacial, a análise temporal no SQLSIM é realizada diretamente utilizando operadores nativos do PostgreSQL, como EXTRACT, que permitem manipular atributos temporais de forma eficiente. Não há uma função dedicada como nas outras dimensões, mas uma função *temporal_sim*() pode ser criada para simplificar o uso encapsulando as funções nativas.

O agrupamento temporal utiliza a estrutura nativa do SQL e é feito *on-the-fly*, ou seja, diretamente durante a execução da consulta, sem a necessidade de tabelas auxiliares específicas para armazenar resultados intermediários, e segue as seguintes etapas:

1. O atributo temporal de cada registro é convertido para a granularidade especificada (ex.: semanas, meses);

2. Os registros são agrupados com base no valor resultante.

Operadores como DATE_TRUNC e EXTRACT fornecem flexibilidade para ajustar a granularidade, enquanto o operador GROUP BY processa os agrupamentos.

Na Listagem 4.7, o operador EXTRACT extrai o número da semana a partir do atributo *data*, e a cláusula GROUP BY agrupa os registros com base nesse valor. O

resultado é uma agregação dinâmica que mostra o número de registros agrupados por semana. O usuário pode ainda implementar funções arbitrárias para lidar com requisitos específicos de tempo.

Listagem 4.7 – Agrupamento por tempo.

```
  SELECT
        EXTRACT(week FROM data) AS temporal,
        count(id)
    FROM exames
GROUP BY temporal;
```

4.1.2 Cubos de Dados

O principal objetivo do SQLSIM é possibilitar análises OLAP integradas ao combinar dimensões métricas, espaciais, temporais e atributos tradicionais, em um único sistema. Por meio do particionamento realizado pelas UDFs e do uso da cláusula GROUP BY CUBE, é possível construir cubos que organizam os dados em diferentes níveis de granularidade.

As consultas que contêm cláusulas GROUP BY com dois ou mais atributos permitem a criação de tabelas cruzadas (*cross-tabs*) ou cubos de dados (*data cubes*) dos fatos, para os quais em geral aplicam-se funções de agregação (*sum, min, max, avg, count*) sobre seus atributos. A abordagem SQLSIM se integra ao SQL tradicional, em especial com os operadores GROUP BY e CUBE, como por exemplo, na Listagem 4.8 que permite agregar a contagem de tuplas pela similaridade dos dados biométricos, pela localização dos hospitais, e no tempo por semana.

Listagem 4.8 – Agrupamento por similaridade em SQL em 3 dimensões.

```
  SELECT Biometric, Hospitais, Tempo, count(*)
    FROM exames
GROUP BY metric_sim(g_ver, g_bra, plaq, 5) as Biometric,
         spatial_sim(lat, long, 3) as Hospitais,
         temporal_sim(data, ''week'') as Tempo;
```

Após a execução da consulta, operações *slice* e *dice* podem ser realizadas pela ferramenta de visualização empregada para exibição do cubo. E as operações *roll-up* e *drill-down* são obtidas com a reexecução das consultas com novos limiares de distância. A granularidade é definida pelo limiar de distância para a formação dos grupos de tuplas, pois expande ou restringe o espaço de similaridade.

O cubo tridimensional gerado pela Listagem 4.8 permite analisar simultaneamente a relação entre dados biométricos (dimensão métrica), localização geográfica (dimensão

Espacial) e períodos de tempo (dimensão Temporal). Essa abordagem oferece análises valiosas para aplicações em diversos domínios, como saúde e logística.

4.2 Considerações Finais

A abordagem proposta pelo SQLSIM oferece uma solução poderosa para análise exploratória de dados, tornando possível realizar consultas de similaridade multidimensional diretamente no ambiente do SGBD.

Mesmo que o SQLSIM tenha sido implementado no PostgreSQL, o método pode ser levado para SGBDs relacionais de vários fornecedores que implementem os recursos descritos da linguagem SQL padrão, mediante a tradução das funções, resguardadas as diferenças nas sintaxes de suas linguagens procedurais.

Além disso, o método também apresenta flexibilidade para ser configurado tanto de forma genérica, na forma de uma solução amplamente aplicável a diversos domínios, quanto de forma acoplada, integrando-se diretamente aos dados e estruturas específicas de um sistema particular.

Embora o exemplo prático tenha sido no contexto de dados biométricos, o método pode ser estendido para outros tipos de dados complexos como imagens, vídeos e textos. A capacidade do SQLSIM de processar diferentes formatos de dados está diretamente relacionada à definição das funções de similaridade. Para novos tipos de dados, UDFs específicas podem ser desenvolvidas para lidar com representações particulares.

Para suportar diferentes domínios, ajustes na estrutura das tabelas também podem ser necessários. Dependendo do caso de uso, novas tabelas podem ser criadas ou adaptadas para armazenar dados específicos da aplicação.

A modularidade do SQLSIM permite que essas adaptações sejam feitas de maneira incremental, sem impactar a estrutura central da abordagem. Dessa forma, a ferramenta pode ser aplicada a um amplo conjunto de cenários analíticos. O código-fonte dos algoritmos implementados está disponível em <https://github.com/liviomendonca/sqlsim>.

Capítulo 5

Experimentos e Análise dos Resultados

Este capítulo apresenta os experimentos realizados para avaliar o SQLSIM. Os experimentos foram conduzidos utilizando o SQLSIM implementado no PostgreSQL na versão 16 e tiveram como principal objetivo verificar a integração das UDFs com a linguagem SQL tradicional dentro do ambiente do PostgreSQL. Mais especificamente, buscou-se avaliar se as funções desenvolvidas poderiam ser corretamente incorporadas às consultas SQL e utilizadas em conjunto com a cláusula GROUP BY para realizar agrupamentos.

Para isso, foram realizadas consultas SQL utilizando as UDFs, aplicadas sobre a base de dados *breastcancer* (WOLBERG et al., 1993) e um conjunto de dados sintéticos complementares para as dimensões Espacial e Temporal. O critério para determinar o sucesso dos experimentos foi a capacidade das funções de processar os registros e gerar identificadores de agrupamento compatíveis com a cláusula GROUP BY, sem a necessidade de processamento externo. Os resultados demonstraram que as UDFs operaram corretamente dentro do PostgreSQL, retornando os agrupamentos esperados e confirmando que a abordagem do SQLSIM pode ser aplicada de forma integrada ao SQL padrão.

Nos tópicos seguintes, serão detalhados os procedimentos adotados nos testes, a estrutura das queries utilizadas e as observações sobre os agrupamentos gerados.

5.1 Método para a Avaliação

A avaliação do SQLSIM foi conduzida por meio da execução de consultas SQL que incorporaram as funções *spatial_sim*() e *metric_sim*() descritas no Capítulo 4. O objetivo principal foi verificar a correta integração dessas funções ao PostgreSQL e sua execução em conjunto com a cláusula GROUP BY e o operador CUBE, garantindo que os registros fossem agrupados conforme os critérios estabelecidos.

O método de avaliação iniciou com a implementação das UDFs, sendo que as funções *spatial_sim*() e *metric_sim*() foram desenvolvidas para processar os registros, realizar os agrupamentos com base em critérios de similaridade e atribuir identificadores de grupo. E seguiu com a criação e execução de consultas que utilizavam essas funções em conjunto

com a cláusula GROUP BY, permitindo que o próprio SGBD realizasse os agrupamentos sem necessidade de processamento externo.

Após a execução de cada consulta, foi verificado se as mesmas foram processadas sem erros no PostgreSQL e se os registros foram corretamente processados e organizados em grupos distintos conforme os parâmetros da consulta, confirmando o funcionamento estrutural da abordagem.

O processo iniciou-se com consultas simples, testando o agrupamento métrico para segmentar registros com características similares. A partir disso, a análise foi expandida para múltiplas dimensões, incorporando dados espaciais e temporais, onde utilizamos consultas SQL estruturadas em duas abordagens principais:

- Cubos bidimensionais

  Analisamos as interações entre atributos espaciais e métricos, observando se a segmentação respeitava a estrutura esperada dos dados.

- Cubos tridimensionais

  Introduzimos a dimensão Temporal para verificar a evolução dos agrupamentos ao longo do tempo e testar a flexibilidade do SQLSIM.

5.1.1 Conjunto de Dados do Experimento

Para validar a abordagem, foi utilizada a base de dados *breastcancer* que contém atributos numéricos que podem ser representados como vetores de características. E para as dimensões Espacial e Temporal, foram gerados dados sintéticos simulando coordenadas geográficas dos hospitais e datas dos exames.

5.1.1.1 Base de Dados *Breastcancer*

A base de dados *breastcancer* é amplamente conhecida na comunidade acadêmica e foi escolhida para representar a dimensão Métrica dos experimentos. Ela possui os seguintes atributos principais:

- Tamanho: Composto por 569 linhas e 30 colunas.

- Natureza dos dados: Inclui atributos numéricos relacionados a características biométricas de células cancerígenas, como raio do nódulo, textura, e perímetro.

- Estrutura dos dados: Os dados estão organizados em múltiplos atributos numéricos, como *radius*, *texture*, *perimeter*, entre outros, que descrevem características biométricas de células cancerígenas.

A estrutura dos dados, composta por atributos numéricos, possibilita a criação de vetores de características, que representam cada registro como um ponto em um espaço métrico multidimensional. Essa abordagem foi utilizada para calcular a similaridade métrica entre os registros, permitindo identificar agrupamentos de células com características similares.

5.1.1.2 Dados Sintéticos

Apesar de não ser uma base de dados completamente orgânica, o mais importante neste experimento é o formato dos dados e as dimensões Métrica (dados complexos), Espacial (*lat*/*lon* ou área) e Temporal (data). Em um cenário real onde os dados seriam provenientes de um sistema hospitalar, os mesmos já teriam esses atributos provavelmente no formato de metadados, assim como no conjunto de *tweets* usado no trabalho correlato da Seção 3.3.1. Os dados foram gerados artificialmente com as seguintes características:

- ❏ Dimensão Espacial: Coordenadas geográficas (latitude e longitude) atribuídas a hospitais fictícios, simulando locais de realização dos exames;
- ❏ Dimensão Temporal: Datas aleatórias associadas a registros de exames, permitindo análises sazonais e temporais.

5.1.1.3 Tabela de Fatos

A junção das bases de dados *breastcancer* e da base sintética foi realizada na tabela de fatos *breastcancer*, criada conforme a Listagem 5.1.

Listagem 5.1 – Criação da tabela *breastcancer*.

```
CREATE TABLE breastcancer (
    id integer PRIMARY KEY,
    diagnosis char(1),
    radius1 double precision,
    texture1 double precision,
    perimeter1 double precision,
    area1 double precision,
    smoothness1 double precision,
    compactness1 double precision,
    (...)
    time_stamp date,
    hospital geometry(point));
```

Uma observação importante é que as coordenadas geográficas dos hospitais não foram separadas em dois atributos *lat* e *long*. Em vez disso, os hospitais foram representados

como um ponto formado pelas coordenadas. Essa abordagem simplifica as consultas, permitindo o uso de um único atributo para localização. No entanto, é importante ressaltar que essa decisão é opcional e pode ser ajustada conforme a necessidade do usuário. A tabela de fatos utilizada nos experimentos está disponível junto ao código-fonte no endereço <https://github.com/liviomendonca/sqlsim>, e a mesma contempla os dados sintéticos gerados.

5.2 Experimentos

Os experimentos apresentados nessa seção visam demonstrar a aplicabilidade da ferramenta, destacando sua flexibilidade e usabilidade em diferentes cenários. Os mesmos foram conduzidos para avaliar a integração do SQLSIM com a linguagem SQL e para explorar como ela pode ser utilizada em consultas por similaridade que visam explorar os dados.

Diferente de abordagens tradicionais que exigem pré-processamento externo, o SQLSIM foi projetado para funcionar diretamente no SGBD, permitindo que a similaridade seja explorada de forma fluida e interativa. A cada experimento, ajustamos parâmetros como granularidade, limiares, dimensões e vetores de características, observando o impacto das mudanças nos resultados das consultas.

Nos experimentos realizados, exploramos a construção de cubos multidimensionais e a integração do SQLSIM com operadores SQL convencionais, como GROUP BY e CUBE. Essa abordagem permitiu testar como diferentes configurações impactam a formação dos agrupamentos e verificar a compatibilidade do método com o SQL tradicional.

A configuração do PostgreSQL para os testes foi feita através do carregamento da base de dados para o banco de dados; implementação das funções $spatial\_sim()$ e $metric\_sim()$ escritas em PL/pgSQL permitindo sua chamada dentro de queries SQL e instalação da biblioteca PostGIS.

5.2.1 Exploração unidimensional

O SQLSIM foi projetado com foco em análises multidimensionais, mas isso não impede que ele seja utilizado em cenários unidimensionais. Para demonstrar essa versatilidade, inicialmente realizamos experimentos utilizando apenas uma única dimensão, conforme a Listagem 5.2.

Listagem 5.2 – Agrupamento na dimensão Espacial

```
SELECT
      spatial_sim(hospital, 3) AS spatial,
      count(id)
  FROM breastcancer
```

```
GROUP BY spatial;
```

A simplicidade da modificação dos parâmetros foi um dos aspectos positivos observados nos testes. Ajustes como a alteração do limiar de similaridade foram realizados de forma direta, sem necessidade de modificar a estrutura da consulta ou reescrever as UDFs.

A Listagem 5.2 foi realizada no pgAdmin como apresentado na Figura 5. O resultado dessa consulta foi exportado em formato *.csv* e processado com a biblioteca *Plotly* disponível em Python. A visualização resultante é apresentada na Figura 6 em formato de *treemap* mostrando a segmentação dos dados baseada na similaridade espacial entre hospitais, destacando agrupamentos próximos geograficamente.

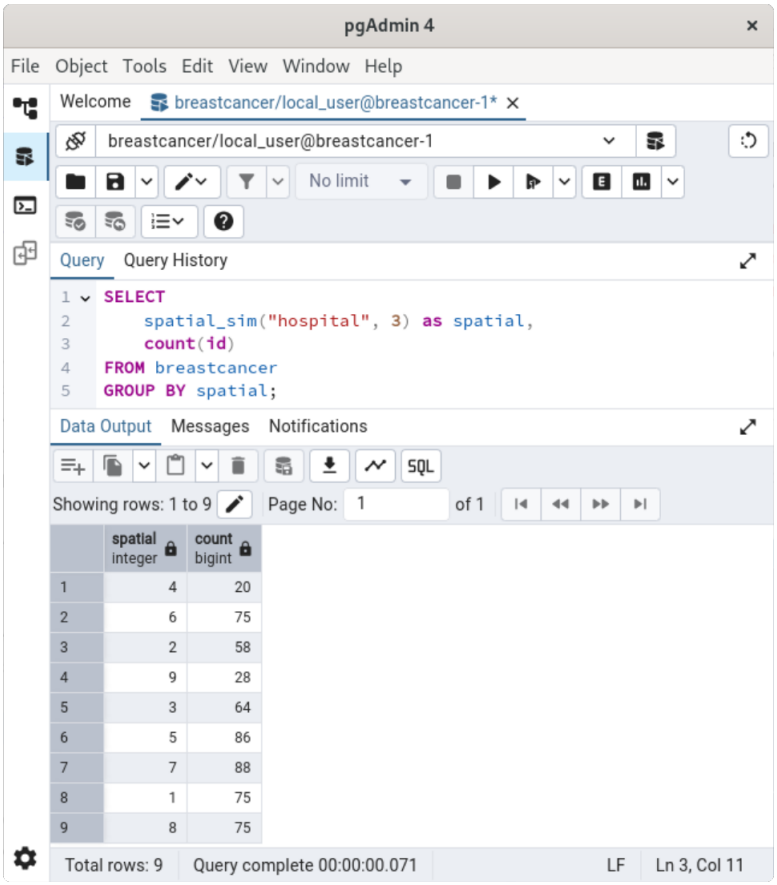

Figura 5 – Captura de tela da execução da Listagem 5.2.

Esse comportamento se estende também para as outras dimensões, como no caso de consultas utilizando a função *metric_sim*().

Listagem 5.3 – Agrupamento na dimensão Métrica

```
  SELECT metric_sim(
              id, radius1, texture1, perimeter1, 80) AS metric
              count(id)
    FROM breastcancer
GROUP BY metric;
```

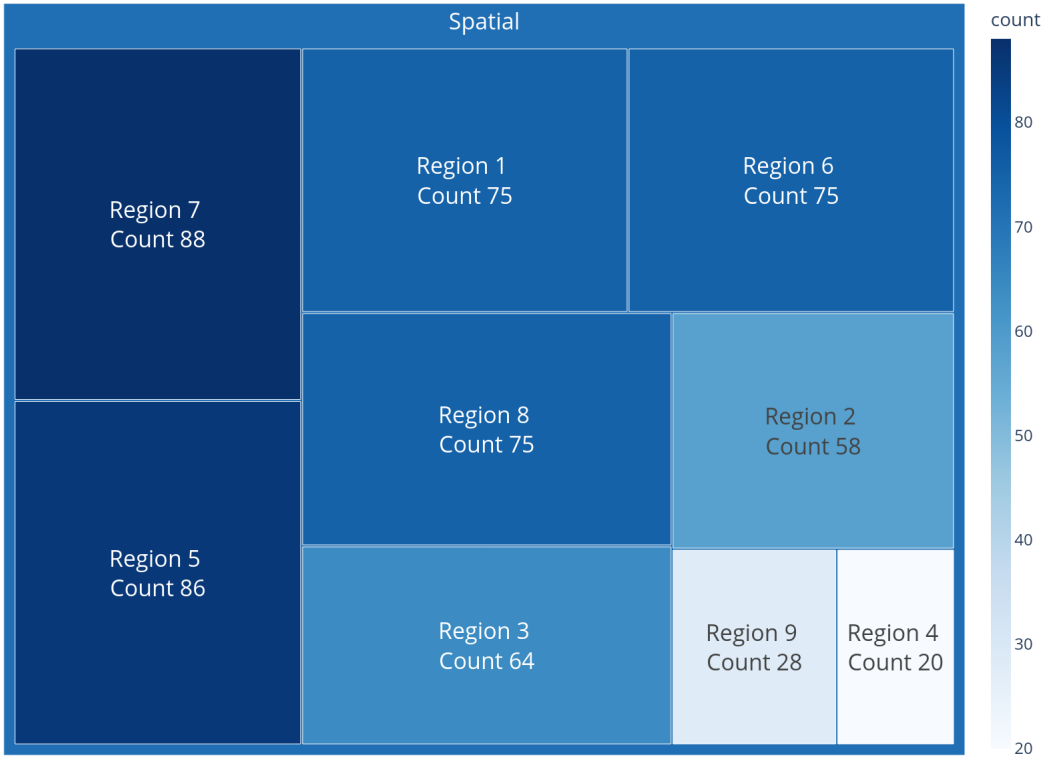

Figura 6 – Visualização da Listagem 5.2.

Na Listagem 5.3, utilizamos atributos biométricos para compor um vetor de características e determinar a similaridade entre os registros. O resultado dessa agregação pode ser visualizado na Figura 7.

Assim como na análise espacial, aumentar ou diminuir o limiar de similaridade permite ajustar a granularidade do agrupamento. Esse comportamento demonstra que o SQLSIM permite ajustar granularidades de forma fluida, sem exigir reformulações estruturais. O impacto das mudanças foi imediato, permitindo que diferentes configurações fossem testadas com facilidade.

5.2.2 Integração com SQL Tradicional

Uma das vantagens mais notáveis do SQLSIM é sua integração direta com SQL tradicional, permitindo que funções de similaridade sejam utilizadas dentro de consultas convencionais sem necessidade de modificar a sintaxe ou utilizar ferramentas externas.

Para validar essa integração, experimentamos o uso do SQLSIM com os operadores GROUP BY e CUBE, observando como as funções de similaridade interagem com as agregações nativas do SGBD.

Na Listagem 5.4, utilizamos SQLSIM para calcular um cubo bidimensional, combinando as dimensões Métrica e Espacial. Na dimensão Métrica, escolhemos o vetor de características formado pelos atributos $radius1$, $texture1$ e $perimeter1$.

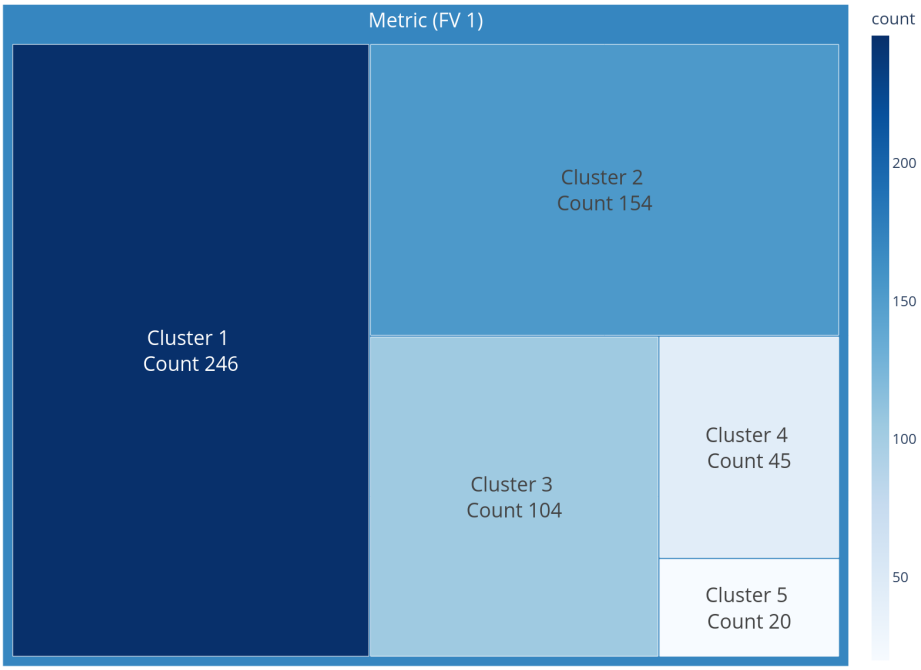

Figura 7 – Visualização da Listagem 5.3.

Listagem 5.4 – Agrupamento na dimensão Espacial e Métrica

```
  SELECT
        spatial_sim(hospital, 9) AS spatial,
        metric_sim(
            id, radius1, texture1, perimeter1, 80
        ) AS metric,
        count(id)
    FROM breastcancer
GROUP BY
        CUBE(spatial, metric);
```

Esse tipo de consulta se encaixa perfeitamente na sintaxe SQL tradicional, permitindo a exploração dos dados sem a necessidade de aprendizado de novas linguagens ou sintaxes complicadas.

A mesma abordagem foi estendida para um cubo tridimensional, Listagem 5.5, onde adicionamos a dimensão Temporal à análise.

Listagem 5.5 – Cubo tridimensional

```
  SELECT
        spatial_sim(hospital, 9) AS spatial,
        extract(month from time_stamp) as temporal,
        metric_sim(
            id, radius1, texture1, perimeter1, 80
```

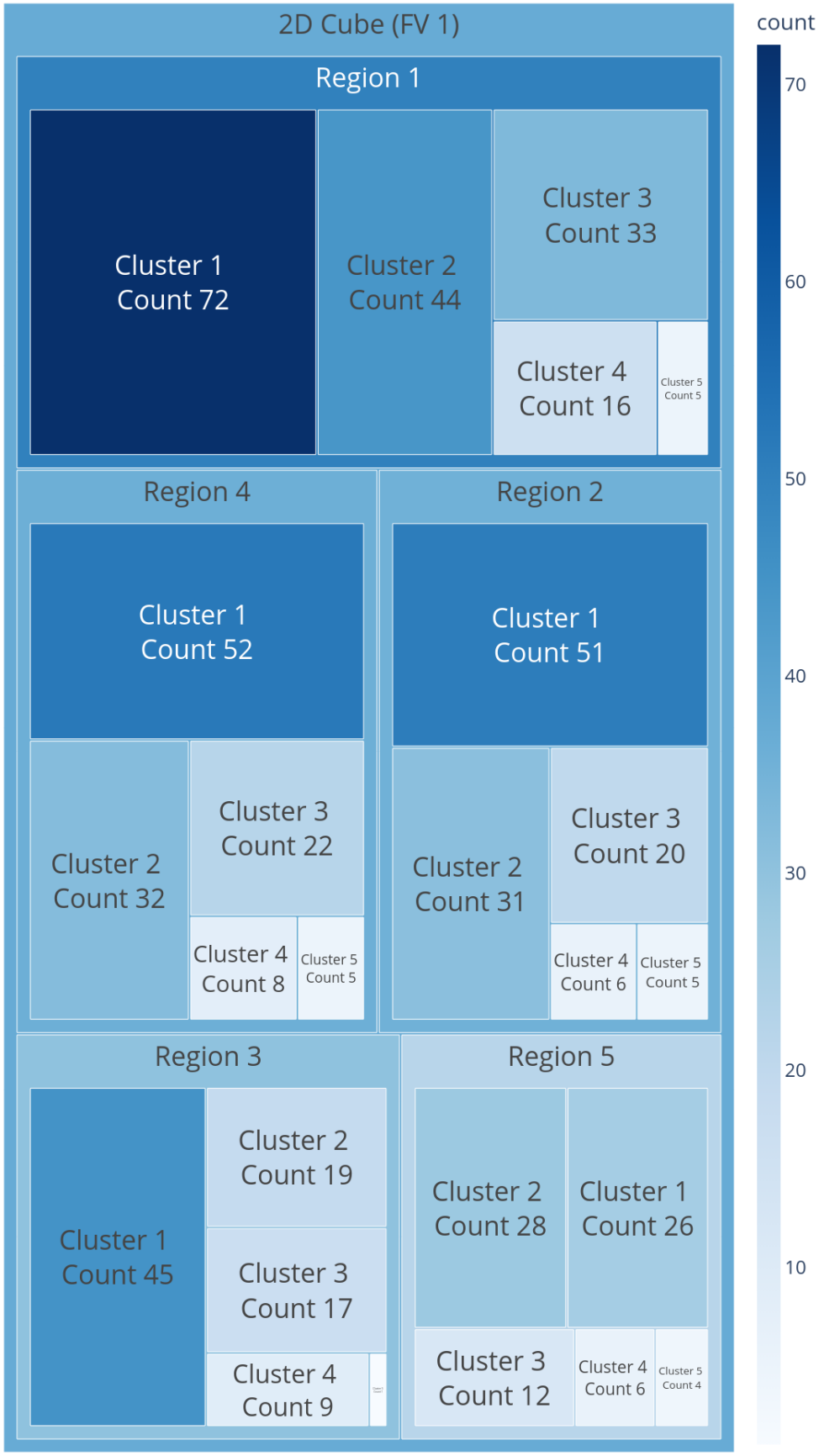

Figura 8 – Cubo 2D: espacial e métrico (Listagem 5.4).

```
        ) AS metric,
        count(id)
    FROM breastcancer
GROUP BY
        CUBE(temporal, spatial, metric);
```

Esse experimento demonstrou que o SQLSIM se adapta bem a cenários mais complexos, permitindo a criação de cubos multidimensionais que combinam diferentes aspectos dos dados.

As Figuras 8 e 9 apresentam a visualização dos cubos das Listagens 5.4 e 5.5 respectivamente.

5.2.3 Exploração Iterativa

Permitir que usuários ajustem parâmetros de similaridade faz com que o SQLSIM possibilite uma exploração iterativa que não se limita a parâmetros como limiar de distância e dimensões. Podemos fazer consultas com vetores de características completamente diferentes ajustando os mesmos diretamente na consulta mas mantendo sua estrutura inalterada. Isso torna o método altamente útil para análises exploratórias, onde a experimentação contínua é essencial.

Por exemplo, na Listagem 5.6, utilizamos um novo conjunto de atributos biométricos ($area1$, $smoothness1$ e $compactness1$) para construir um cubo 2D:

Listagem 5.6 – Cubo 2D com novo vetor de características

```
  SELECT
        spatial_sim(hospital, 9) AS spatial,
        metric_sim(
            id, area1, smoothness1, compactness1, 1600
        ) AS metric,
        count(id)
    FROM breastcancer
GROUP BY
        CUBE(spatial, metric);
```

Esse ajuste simples mas poderoso já produz uma nova organização dos agrupamentos, permitindo observar padrões distintos nos dados. A Figura 10 ilustra o resultado da Listagem 5.6.

5.3 Avaliação dos Resultados

Os experimentos demonstraram que o SQLSIM não é apenas uma abordagem que simplifica um problema complexo, mas também traz uma nova perspectiva sobre o problema

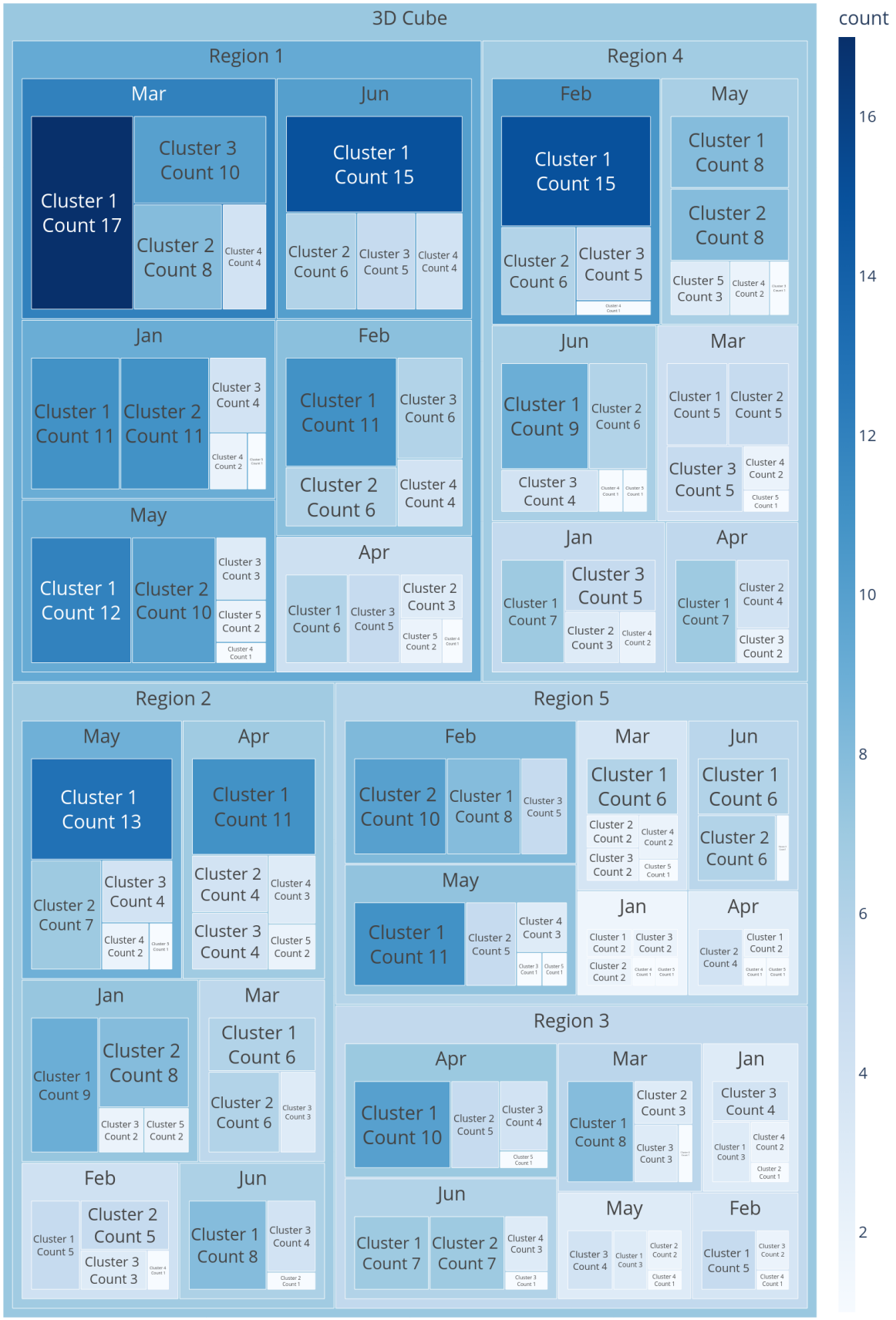

Figura 9 – Cubo 3D: temporal, espacial e métrico (Listagem 5.5).

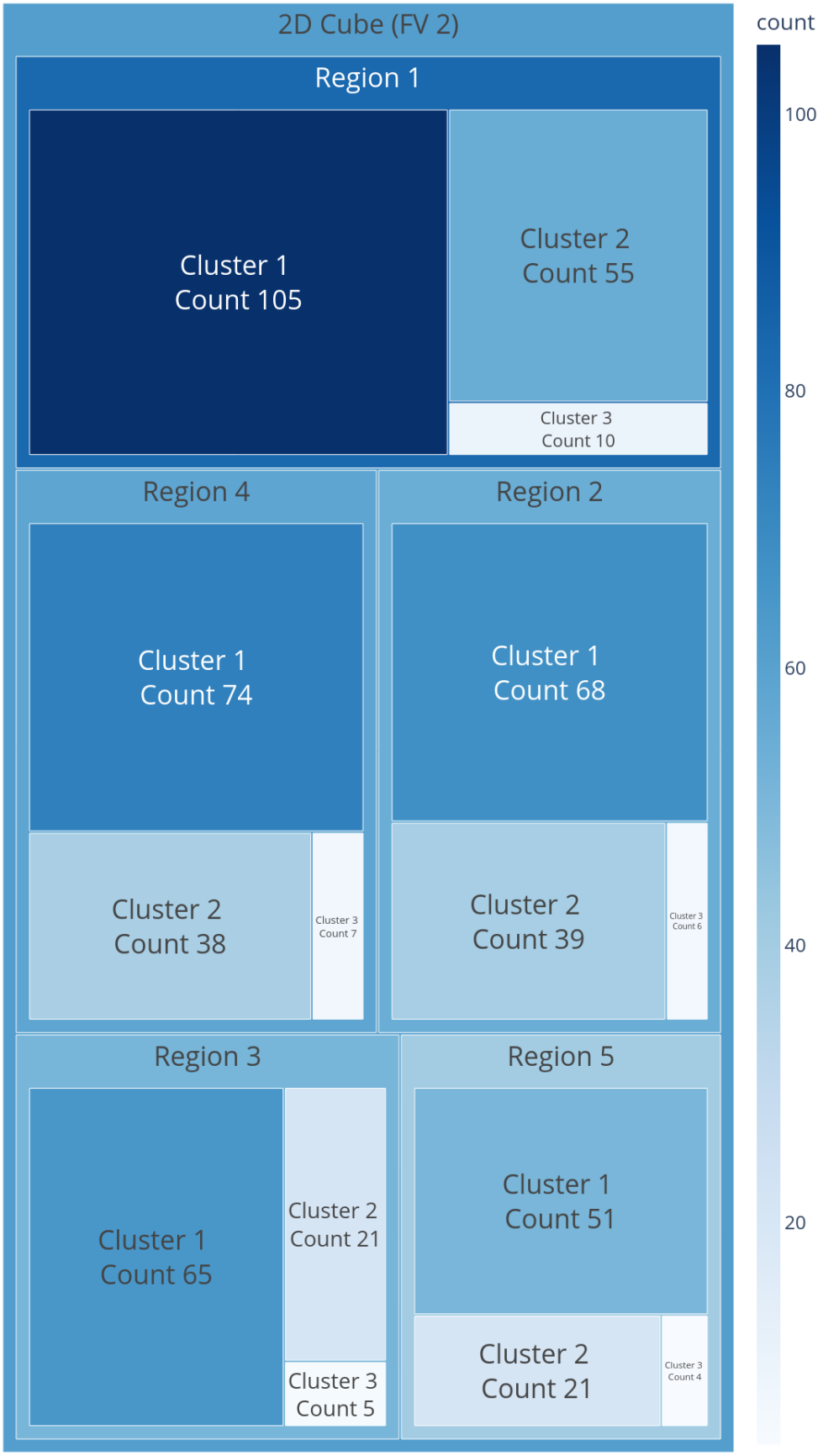

Figura 10 – Cubo 2D com novo vetor de características (Listagem 5.6).

de agrupamentos por similaridade em SGBDs. A integração nativa com SQL reduziu a complexidade da análise, permitindo que usuários explorem diferentes cenários de forma iterativa e sem sair do ambiente do banco de dados.

O experimento mais interessante surgiu na Listagem 5.5 onde combinamos todas as dimensões (Métrica, Espacial e Temporal) e percebemos que o SQLSIM conseguia manter sua estrutura objetiva e alinhada com a linguagem SQL. Isso confirmou que a abordagem pode ser utilizada para análises exploratórias dinâmicas sem comprometer a simplicidade de uso.

Os experimentos demonstraram que a abordagem do SQLSIM é uma solução promissora para análise de dados multidimensionais, permitindo consultas por similaridade de maneira escalável, eficiente e acessível a usuários que já estão familiarizados com bancos de dados relacionais ou que precisem de um controle maior sobre o *framework*.

Podemos perceber que o SQLSIM permite uma exploração fluida dos dados sem exigir grandes modificações no código SQL. O fato de os usuários finais poderem alterar granularidades, vetores de características, entre outros parâmetros de agrupamento demonstra que a ferramenta foi bem integrada ao SGBD.

Um aspecto positivo notável foi a possibilidade de alterar o nível de granularidade das consultas sem reescrever a lógica do SQL. Por exemplo, modificar o intervalo de análise temporal (de meses para semanas) exigiu apenas uma pequena alteração no parâmetro da função EXTRACT e, nesse caso, utilizamos inclusive funções nativas do PostgreSQL. Outro aspecto foi o caso de alteração do vetor de características que é feito diretamente na consulta sem necessidade de reprocessamento externo ou reconfiguração do banco.

5.3.1 Iteração e Experimentação

Diferente dos métodos convencionais de agrupamento por similaridade que exigem múltiplas etapas de processamento e, muitas vezes, a exportação de dados para ferramentas externas, o SQLSIM permitiu realizar experimentos iterativos diretamente no banco de dados. Isso significa que, ao perceber que um agrupamento estava muito granular ou muito amplo, podíamos simplesmente modificar o limiar de distância na consulta e reexecutá-la, obtendo resultados ajustados quase imediatamente.

Essa flexibilidade é fundamental para usuários que precisam testar diferentes configurações antes de definir um modelo ideal para análise e, principalmente, quando em um trabalho de análise preliminar, exploratória.

Em métodos tradicionais, ajustes como a troca da métrica de similaridade ou a alteração do critério de agrupamento geralmente exigem reprocessamento externo e conversão dos dados para estruturas específicas de bibliotecas de *machine learning* enquanto no SQLSIM, esse processo ocorreu dentro do PostgreSQL, sem necessidade de transferências de dados.

Outro ponto relevante identificado nos experimentos foi o comportamento de certos algoritmos de agrupamento que dependem de múltiplas iterações. Métodos como *k-means* tradicionalmente realizam sucessivas atualizações dos centróides, exigindo várias passagens sobre os dados, o que pode gerar desafios quando implementados dentro de um SGBD.

Para contornar essa limitação e manter a execução eficiente dentro do banco, uma abordagem promissora é o uso de estratégias de *single-pass clustering*. Essa abordagem reduz significativamente o custo computacional e se encaixa melhor no paradigma relacional, onde múltiplas iterações podem ser custosas devido à necessidade de leituras repetidas nas tabelas.

Nos experimentos realizados, identificamos que a aplicação de uma estratégia inspirada no método citado nas Seções 3.2.1 e 3.1.2 que utiliza *single-pass* reduziria a necessidade de múltiplas execuções da consulta e tornaria o SQLSIM mais eficiente para grandes volumes de dados.

5.3.2 Adaptabilidade

Um dos aspectos mais positivos observados nos experimentos foi a facilidade de adaptação do SQLSIM para diferentes domínios e aplicações. Embora os testes tenham sido conduzidos com dados biométricos e espaciais, a abordagem demonstrou potencial para ser aplicada em uma ampla variedade de cenários que envolvam agrupamentos por similaridade.

A flexibilidade do SQLSIM reside na possibilidade de trocar a função de similaridade sem alterar a estrutura geral das consultas. Isso significa que, dependendo da necessidade, é possível substituir a métrica utilizada, experimentar diferentes estratégias de agrupamento ou até mesmo modificar o critério de cálculo do centróide. Essa modularidade reforça a versatilidade da ferramenta, permitindo que ela seja adaptada para distintos conjuntos de dados sem necessidade de reformulações profundas.

Apesar dessa flexibilidade, a adaptação do SQLSIM para novos domínios exige um entendimento sólido da linguagem procedural PLpgSQL, o que pode representar um obstáculo para usuários sem experiência prévia com programação dentro de bancos de dados. A implementação de novas funções de similaridade ou o ajuste fino das estratégias de agrupamento requerem um conhecimento mais técnico sobre a estrutura do PostgreSQL, UDFs e da Linguagem Procedural.

5.3.3 Outros Aspectos e Comparações

Além dos experimentos realizados, existem outros pontos relevantes para discussão sobre o método proposto. Nessa seção, discutiremos outros aspectos que podem influenciar o desempenho e a aplicabilidade do método proposto.

5.3.3.1 Comparação com Outras Abordagens

A abordagem SGB apresentada em (TANG et al., 2016), descrita na Seção 3.2.1 segue um modelo específico de sintaxe que altera a linguagem SQL, enquanto o SQLSIM foi projetado para se integrar de forma mais natural ao SQL utilizando UDFs.

A sintaxe proposta pelo método SGB introduz os operadores WITHIN, ON-OVERLAP e DISTANCE-TO-ALL, que determinam como os elementos devem ser agrupados e avaliados quanto à similaridade. No entanto, no SQLSIM, esses operadores foram absorvidos pelas suas funções, permitindo que as consultas seguissem a sintaxe tradicional do SQL.

Listagem 5.7 – Consulta com SGB-ALL.

```
    SELECT column, aggregate-func(column)
      FROM table-name
     WHERE condition
  GROUP BY column DISTANCE-TO-ALL [L2|LINF]
    WITHIN ϵ
ON-OVERLAP [JOIN-ANY|ELIMINATE|FORM-NEW-GROUP];
```

Essa diferença estrutural tem impacto direto na forma como consultas são escritas, interpretadas e executadas dentro do banco de dados. Por exemplo, a Listagem 5.7 do método SGB, apresentada também na Seção 3.2.1, pode ser reescrita no SQLSIM na forma da Listagem 5.8, onde:

- ❏ o operador WITHIN foi incorporado como um parâmetro da função *metric_sim*().
- ❏ o operador ON-OVERLAP foi implementado internamente, e, por padrão, utiliza a opção JOIN-ANY.
- ❏ o operador DISTANCE-TO-ALL também foi absorvido dentro do cálculo das distâncias, utiliza a opção L2.

Listagem 5.8 – Agrupamento na dimensão Métrica

```
SELECT metric_sim(
          id, radius1, texture1, perimeter1, 80) AS metric
  FROM breastcancer;
```

É importante notar que as outras opções para os operadores DISTANCE-TO-ALL e ON-OVERLAP podem ser incorporadas ao SQLSIM como parâmetros adicionais dentro das funções, desde que a função seja ajustada para tal. Além disso, novas funções de similaridade podem ser implementadas para suportar diferentes métricas de distância, como Manhattan, Cosseno ou Minkowski, oferecendo maior flexibilidade na definição dos agrupamentos.

O fato do SQLSIM optar por um modelo baseado em UDFs parametrizáveis permite que diferentes critérios de similaridade possam ser aplicados diretamente nas funções sem necessidade de reescrever a estrutura da consulta em si.

Dessa forma, a sintaxe do SQLSIM permanece compacta e alinhada ao SQL tradicional, permitindo que usuários com conhecimento em SQL consigam aplicar consultas por similaridade de forma mais intuitiva.

5.3.3.2 Diferentes Métricas e Critérios de Agregação

Embora não tenhamos realizado testes específicos para a troca da função de agregação, o SQLSIM permite que a função *aggr_metric_sim*() seja reescrita conforme a necessidade do usuário. Ou seja, diferentes estratégias de agregação podem ser utilizadas, bastando substituir a função existente por uma implementação que seja mais adequada para o caso.

5.3.3.3 Outras Linguagens Procedurais

Um dos desafios da abordagem SQLSIM é a diferença estrutural entre as tabelas do ambiente relacional e os sets e arrays, que são frequentemente utilizados em algoritmos de agrupamento em linguagens como Python ou R.

O uso da linguagem procedural SQL (PLpgSQL) mostrou-se um ponto de atenção, pois, embora permita definir funções personalizadas para similaridade, algumas operações não são tão diretas. Isso exigiu adaptações na forma de manipulação dos dados, tornando algumas operações mais verbosas e exigindo um entendimento mais aprofundado do modelo relacional e da linguagem PLpgSQL.

O PostgreSQL oferece suporte a diversas linguagens procedurais para a criação de UDFs, cada uma com suas características e aplicações específicas. Embora o PLpgSQL tenha sido utilizado no SQLSIM devido à sua integração nativa com o banco de dados, outras linguagens podem ser exploradas para ampliar a ferramenta. Entre essas opções, destaca-se a PLpython, que permite a execução de funções escritas em Python dentro do PostgreSQL. Mais detalhes sobre essa linguagem podem ser vistos na documentação do PostgreSQL em <https://www.postgresql.org/docs/16/plpython.html>.

A principal vantagem do PLpython é a possibilidade de acessar o vasto ecossistema de bibliotecas de análise de dados e *machine learning* disponíveis em Python. Isso permitiria integrar o SQLSIM a ferramentas como:

- ❏ NumPy e SciPy para operações matemáticas.
- ❏ Pandas para manipulação de tabelas e séries temporais.
- ❏ Scikit-learn e TensorFlow para análise de agrupamentos e aprendizado de máquina.

Com essas bibliotecas, seria possível realizar cálculos estatísticos e aplicar algoritmos de clustering sofisticados diretamente no banco de dados, sem necessidade de exportar os dados para um ambiente externo.

Apesar das vantagens, a PLpython opera em modo *unrestricted*, o que significa que para usá-la, precisamos conceder permissões especiais no servidor do banco de dados. Isso pode representar um risco de segurança dependendo dos usuários que tiverem acesso. Essas permissões podem ser um obstáculo em ambientes com restrições de segurança.

Capítulo 6

Conclusão

O estudo teve como principal objetivo investigar a viabilidade da integração de consultas baseadas em similaridade diretamente em Sistemas Gerenciadores de Banco de Dados Relacionais, eliminando a necessidade de ferramentas externas e promovendo uma análise mais eficiente e integrada dos dados.

Ao longo do desenvolvimento, buscou-se validar a hipótese que a adoção do SQLSIM poderia simplificar o agrupamento por similaridade dentro do ambiente do SGBD, facilitando experimentações iterativas, aumentando a flexibilidade da análise de dados multidimensionais e melhorando a usabilidade para usuários que necessitam realizar análises exploratórias.

Os experimentos realizados demonstraram que a abordagem proposta é viável e promissora, permitindo a manipulação eficiente de dados biométricos e espaciais, com possibilidade de aplicação em outros domínios. Além disso, a modularidade do SQLSIM se mostrou um fator determinante para sua adaptabilidade, permitindo que diferentes funções de similaridade e estratégias de agregação fossem testadas sem necessidade de grandes reformulações no código SQL.

Mesmo com essas limitações, os resultados indicam que a abordagem facilita o trabalho exploratório e iterativo dentro dos SGBDRs, reduzindo o tempo e a complexidade necessários para realizar agrupamentos por similaridade. Além disso, o SQLSIM se apresenta como uma alternativa interessante para cenários que exigem experimentação rápida com diferentes parâmetros de similaridade e granularidade na análise dos dados.

6.1 Principais Contribuições

Este trabalho apresentou a proposta do SQLSIM, um método para realizar consultas por similaridade diretamente dentro de SGBDs relacionais, eliminando a necessidade de ferramentas externas e proporcionando maior eficiência e flexibilidade para análise de dados multidimensionais. A seguir, destacamos as principais contribuições alcançadas.

1. Integração de Agrupamento por Similaridade no PostgreSQL

   Uma das contribuições centrais do trabalho foi demonstrar que é possível realizar operações de agrupamento por similaridade diretamente dentro do PostgreSQL, utilizando funções definidas pelo usuário e aproveitando a estrutura do banco de dados relacional. Isso reduz significativamente o *impedance mismatch* comum em soluções que exigem processamento externo, permitindo que a similaridade seja tratada como uma operação nativa do banco.

2. Desenvolvimento de uma *Codebase* com Exemplos e Testes

   O trabalho resultou na criação de uma *codebase* funcional, disponível em <https://github.com/liviomendonca/sqlsim>, incluindo exemplos práticos e testes experimentais, que pode servir como referência para futuros estudos e aplicações. Essa implementação não apenas valida a viabilidade do método, mas também fornece uma base para que desenvolvedores e pesquisadores possam aprimorar e expandir a abordagem proposta.

3. Flexibilidade e Modularidade

   Pensado para ser modular, o SQLSIM permite fácil substituição e personalização de diversos elementos, incluindo funções de similaridade, métricas, entre outros.

4. Possibilidade de Expansão

   Além das contribuições imediatas, este trabalho abre caminho para futuras melhorias, como a criação de bibliotecas de funções agregadoras avançadas e extensões específicas para PostgreSQL, ampliando as possibilidades da abordagem SQLSIM em diferentes contextos.

6.2 Trabalhos Futuros

Com base nos experimentos realizados e nas análises conduzidas ao longo deste estudo, diversas oportunidades de aprimoramento e expansão do SQLSIM foram identificadas.

1. Desenvolvimento de uma Biblioteca de Funções de Agrupamento

   Atualmente, um dos principais desafios para usuários menos experientes é a criação de funções de agrupamento personalizadas dentro do PostgreSQL. Apesar da modularidade do SQLSIM, implementar novas funções requer conhecimento em PLpgSQL, o que pode ser um obstáculo para quem não tem familiaridade com programação procedural em bancos de dados.

   Uma possível solução seria o desenvolvimento de uma biblioteca de funções de agrupamento para SQLSIM, semelhante ao PostGIS para dados geoespaciais. Essa biblioteca poderia oferecer um conjunto de funções prontas para uso, facilitando a adoção

do SQLSIM sem necessidade de desenvolvimento manual. Com essa abordagem, o usuário teria acesso a uma API SQL mais intuitiva e padronizada, reduzindo a necessidade de criar funções do zero.

2. Extender o SQLSIM para Usar Funções Agregadoras

   Funções agregadoras são operadores que combinam múltiplos valores em um único resultado, como SUM(), AVG() e COUNT(). Apesar de o PostgreSQL permitir a criação de novas funções agregadoras via CREATE AGGREGATE, esse processo exige um alto nível de conhecimento técnico.

   Um aprimoramento natural do SQLSIM seria a incorporação de novas funções agregadoras para consultas por similaridade, permitindo que cálculos avançados fossem realizados com a mesma simplicidade dos agregadores SQL padrão.

   Por exemplo, funções como KNN() para encontrar os k vizinhos mais próximos dentro de uma agregação e K-MEANS() para calcular os centróides de clusters diretamente no SQL.

   Essas funções também poderiam ser empacotadas dentro de uma biblioteca de agregação SQLSIM, garantindo uma distribuição facilitada.

3. Suporte a Algoritmos que Requerem Múltiplas Iterações

   Atualmente, o SQLSIM opera utilizando estratégias *single-pass clustering*, o que garante maior eficiência computacional ao processar os dados em uma única varredura. No entanto, existem métodos de agrupamento mais sofisticados, como *k-means* tradicional, DBSCAN e *clustering* hierárquico, que exigem múltiplas iterações para convergir a uma solução ótima.

   Para expandir a aplicabilidade do SQLSIM, futuros estudos podem investigar formas de viabilizar a execução desses algoritmos iterativos dentro do ambiente relacional. Acreditamos que isso poderia ser alcançado por meio de funções agregadoras citadas no item acima.

   O suporte a algoritmos iterativos permitiria que o SQLSIM aumentasse seu alcance para aplicações que exigem agrupamentos mais refinados, combinando a eficiência do SGBD com a precisão de métodos iterativos mais sofisticados.

4. Integração com Bancos de Dados Vetoriais

   A crescente adoção de bancos de dados vetoriais para busca por similaridade sugere que o SQLSIM poderia se beneficiar de integrações com tecnologias especializadas, como o Projeto Korvus (<https://github.com/postgresml/korvus>). Essa integração permitiria que consultas por similaridade aproveitassem estruturas otimizadas para busca em grandes coleções de vetores, melhorando a escalabilidade do método.

5. Portabilidade para outras Linguagens

   Atualmente, a configuração do SQLSIM requer conhecimento de PLpgSQL, o que pode ser uma barreira para usuários menos experientes. Um possível aprimoramento seria o desenvolvimento de uma camada de abstração ou ORM (*Object-Relational Mapping*) para facilitar sua utilização em aplicações externas. A integração com Python e frameworks como SQLAlchemy poderia tornar a adoção do SQLSIM mais intuitiva e acessível inclusive facilitar todos os outros pontos já citados!

6. Complementação dos testes e experimentos

   Realizar teste de usabilidade para avaliar a flexibilidade na configuração da abordagem proposta para diferentes contextos de análise, comparação com o processamento externo para analisar a eficiência do processamento da abordagem proposta; Estudo de caso para uso em um caso real, explorando cenários em que a ferramente agrega valor na análise de dados multidimensionais.

6.3 Contribuições em Produção Bibliográfica

Parte dos resultados deste trabalho foram publicados no XVII *Brazilian e-Science Workshop* (Bresci) (MENDONÇA; BARIONI; RAZENTE, 2024), realizado em conjunto com o Simpósio Brasileiro de Banco de Dados que ocorreu em 2024:

- MENDONÇA, Antônio Lívio C. de; BARIONI, Maria Camila N.; RAZENTE, Humberto. Consultas analíticas por similaridade em SGBD Relacionais. In: BRAZILIAN E-SCIENCE WORKSHOP (BRESCI), 18. , 2024, Florianópolis/SC. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2024 . p. 48-55. ISSN 2763-8774. DOI: <https://doi.org/10.5753/bresci.2024.243330>.

Referências

ARCH, S. et al. The key to effective UDF optimization: Before inlining, first perform outlining. **Proc. VLDB Endow.**, v. 18, n. 1, p. 1–13, 2024. Disponível em: <https://www.vldb.org/pvldb/vol18/p1-arch.pdf>.

BOZKAYA, T.; OZSOYOGLU, M. Indexing large metric spaces for similarity search queries. **ACM Transactions on Database Systems (TODS)**, ACM, v. 24, n. 3, p. 361–404, 1999. Disponível em: <https://doi.org/10.1145/328939.328959>.

CHAKRABARTY, S.; MAKARYCHEV, K. **Single-Pass Pivot Algorithm for Correlation Clustering. Keep it simple!** 2023. Disponível em: <https://arxiv.org/abs/2305.13560>.

CHEN, L. et al. Indexing metric spaces for exact similarity search. **ACM Comput. Surv.**, v. 55, n. 6, p. 128:1–128:39, 2023. Disponível em: <https://doi.org/10.1145/3534963>.

CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-tree: An efficient access method for similarity search in metric spaces. In: **Proceedings of the 23rd International Conference on Very Large Data Bases**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997. (VLDB '97), p. 426–435. ISBN 1-55860-470-7. Disponível em: <http://dl.acm.org/citation.cfm?id=645923.671005>.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de Bancos de Dados, tradução da 6ª edição**. [S.l.]: Pearson Addison Wesley, 2011. ISBN 978-85-4301-381-7.

FANG, L. et al. Single-pass clustering algorithm based on storm. **Journal of Physics: Conference Series**, IOP Publishing, v. 806, n. 1, p. 012017, feb 2017. Disponível em: <https://dx.doi.org/10.1088/1742-6596/806/1/012017>.

GARCIA-ALVARADO, C.; ORDONEZ, C. Clustering binary cube dimensions to compute relaxed GROUP BY aggregations. **Inf. Syst.**, v. 53, p. 41–59, 2015. Disponível em: <https://doi.org/10.1016/j.is.2014.12.008>.

GRAY, J. et al. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. **Data Min. Knowl. Discov.**, v. 1, n. 1, p. 29–53, 1997. doi:10.1023/A:1009726021843. Disponível em: <https://doi.org/10.1023/A:1009726021843>.

GUHA, S.; RASTOGI, R.; SHIM, K. Cure: an efficient clustering algorithm for large databases. In: **ACM SIGMOD International Conference on the Management of Data**. Seattle, WA, USA: [s.n.], 1998. p. 73–84. Disponível em: <https://doi.org/10.1145/276305.276312>.

GUTTMAN, A. R-trees: A dynamic index structure for spatial searching. In: **Int'l Conf. on Management of Data (SIGMOD)**. Boston, MA: [s.n.], 1984. p. 47–57. Disponível em: <https://doi.org/10.1145/602259.602266>.

HAN, J.; KAMBER, M.; PEI, J. **Data mining: Concepts and techniques**. San Diego, CA: Academic Press, 2011.

IQBAL, M.; LISSANDRINI, M.; PEDERSEN, T. B. A foundation for spatio-textual-temporal cube analytics. **Inf. Syst.**, v. 108, p. 102009, 2022. Disponível em: <https://doi.org/10.1016/j.is.2022.102009>.

JAIN, A. K. Data clustering: 50 years beyond k-means. **Pattern Recognition Letters**, v. 31, n. 8, p. 651–666, 2010. Disponível em: <https://doi.org/10.1016/j.patrec.2009.09.011>.

KORN, F.; PAGEL, B.; FALOUTSOS, C. On the 'dimensionality curse' and the 'self-similarity blessing'. **IEEE Trans. Knowl. Data Eng.**, v. 13, n. 1, p. 96–111, 2001. Disponível em: <https://doi.org/10.1109/69.908983>.

LOKOC, J. et al. On indexing metric spaces using cut-regions. **Inf. Syst.**, v. 43, p. 1–19, 2014. Disponível em: <https://doi.org/10.1016/j.is.2014.01.007>.

LU, W. et al. Msql: efficient similarity search in metric spaces using sql. **The VLDB Journal**, Springer-Verlag, Berlin, Heidelberg, v. 26, n. 6, p. 829–854, dez. 2017. ISSN 1066-8888. Disponível em: <https://doi.org/10.1007/s00778-017-0481-6>.

MANOLOPOULOS, Y. et al. **R-Trees: Theory and Applications (Advanced Information and Knowledge Processing)**. Springer, doi:10.1007/978-1-84628-293-5, 2005. Disponível em: <https://doi.org/10.1007/978-1-84628-293-5>.

MENDONÇA, A. L.; BARIONI, M. C.; RAZENTE, H. Consultas analíticas por similaridade em sgbd relacionais. In: **Anais do XVIII Brazilian e-Science Workshop**. Porto Alegre, RS, Brasil: SBC, 2024. p. 48–55. ISSN 2763-8774. Disponível em: <https://sol.sbc.org.br/index.php/bresci/article/view/30580>.

ORDONEZ, C.; CHEN, Z. Horizontal aggregations in SQL to prepare data sets for data mining analysis. **IEEE Trans. Knowl. Data Eng.**, v. 24, n. 4, p. 678–691, 2012. Disponível em: <https://doi.org/10.1109/TKDE.2011.16>.

RAZENTE, H. L.; LIMA, R. L. B.; BARIONI, M. C. N. Similarity search through one-dimensional embeddings. In: **ACM Symposium on Applied Computing (SAC)**. Marrakech, Morocco: [s.n.], 2017. p. 874–879. Doi:10.1145/3019612.3019674.

SAMET, H. **Foundations of Multidimensional and Metric Data Structures**. San Francisco, CA: Morgan Kaufmann, 2006.

SIQUEIRA, P. H. B. et al. Standard SQL approaches for similarity searching. In: **XLIV Latin American Computer Conference, CLEI 2018, São Paulo, Brazil, October 1-5, 2018**. IEEE, 2018. p. 472–481. Doi:10.1109/CLEI.2018.00063. Disponível em: <https://doi.org/10.1109/CLEI.2018.00063>.

TANG, M. et al. Similarity group-by operators for multi-dimensional relational data. **IEEE Trans. Knowl. Data Eng.**, v. 28, n. 2, p. 510–523, 2016. doi:10.1109/TKDE.2015.2480400. Disponível em: <https://doi.org/10.1109/TKDE.2015.2480400>.

WOLBERG et al. **Breast Cancer Wisconsin (Diagnostic)**. 1993. UCI Machine Learning Repository. Disponível em: <https://doi.org/10.24432/C5DW2B>.

YIANILOS, P. N. Data structures and algorithms for nearest neighbor search in general metric spaces. In: **ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA)**. ACM/SIAM, 1993. p. 311–321. Disponível em: <http://dl.acm.org/citation.cfm?id=313559.313789>.

ZEZULA, P. et al. **Similarity Search - The Metric Space Approach**. [S.l.: s.n.], 2006. v. 32. ISBN 978-0-387-29146-8.

ZHANG, T.; RAMAKRISHNAN, R.; LIVNY, M. Birch: An efficient data clustering method for very large databases. In: **ACM SIGMOD International Conference on Management of Data**. Montreal, Quebec, Canada: ACM, 1996. p. 103–114. Disponível em: <https://doi.org/10.1145/235968.233324>.

ZHANG, Y. et al. The percentage cube. **Inf. Syst.**, v. 79, p. 20–31, 2019. Disponível em: <https://doi.org/10.1016/j.is.2018.01.005>.