ESTUDO SOBRE IMPLEMENTAÇÃO E OTIMIZAÇÃO DE CONSULTAS POR SIMILARIDADE SOB DOMÍNIO DE DADOS COMPLEXOS EM BANCOS DE DADOS RELACIONAIS

Leonardo Luca Ribeiro[1], **Daiane Mastrangelo Tomazeti**[1]

[1]Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP)
Câmpus Hortolândia – São Paulo – SP – Brasil

luca_lribeiro@hotmail.com, daianetomazeti@ifsp.edu.br

Abstract. *The paper analyzes the storage and retrieval of complex data based on the idea of similarity among them, using range operators and the k nearest neighbors. The research focuses on developing similarity queries in the do- main of complex data, using SQL (Structured Query Language) language in a relational database. The paper also addresses the use of hints with the aim of manipulating the execution plan of queries, in order to understand and demons- trate how the query optimization process is carried out by the Oracle optimizer, the CBO (Cost-Based Optimizer). The results show that, even though simila- rity queries are not foreseen in the relational model, it is possible to implement and optimize them without the need for any extension, merely by adapting the existing tools.*

Resumo. *O artigo analisa o armazenamento e recuperação de dados complexos com base na ideia de similaridade entre eles, utilizando operadores de abrangência e os k vizinhos mais próximos. A pesquisa concentra-se em desenvolver consultas por similaridade em domínio de dados complexos, utilizando a linguagem SQL (Linguagem de Consulta Estruturada) em um banco de dados relacional. O artigo também abordará a utilização de dicas com o objetivo de manipular o plano de execução das consultas, a fim de compreender e demonstrar como o processo de otimização da query é realizado pelo otimizador da Oracle, o CBO (Cost-Based Optimizer). Os resultados mostram que, mesmo que con- sultas por similaridade não estejam previstas no modelo relacional, é possível implementá-las e otimizá-las sem a necessidade de nenhuma extensão, apenas adaptando as ferramentas já existentes.*

1. Introdução

A recuperação de dados complexos por meio da similaridade é uma técnica amplamente utilizada na comparação de objetos reais. Para realizar essa busca, é necessário transformar os objetos em vetores de características e, em seguida, comparar esses vetores em um determinado espaço métrico [Queiroz and Pinto 2014].

Dados complexos se referem a informações que possuem estruturas com múltiplas dimensões ou relações interconectadas entre seus elementos. Esses tipos de dados são difíceis de serem representados ou compreendidos por meio de abordagens tradicionais de modelagem e análise de dados. Um exemplo bastante abrangente de dados complexos são conteúdos multimídia como: imagens, sons, vídeos, documentos contendo muitas páginas, entre outros [Kaster 2012]. Esse conceito de dados complexos tem sido objeto de diversos estudos sobre armazenamento, indexação e recuperação desses dados.

Este artigo tem como proposta apresentar uma análise consistente sobre a recuperação de dados complexos, baseada na ideia de similaridade entre eles e sua implementação em um banco de dados relacional. Para tanto foram desenvolvidas consultas por similaridade, utilizando operadores de abrangência e dos k vizinhos mais próximos, respeitando suas respectivas regras de equivalência para a implementação em um ambiente relacional. O plano de execução das consultas foi analisado e alterado utilizando *hints* [1] suportados pelo Banco de Dados Oracle para manipular o plano de execução e, assim, estabelecer uma comparação entre a consulta otimizada pelo otimizador da Oracle e uma consulta manipulada por meio de *hints*.

A aplicação dos *hints* servirá unicamente para provar que o sistema do Oracle é capaz de identificar as diferentes possibilidades de planos de execução possibilitadas pelas regras de equivalência dos operadores por similaridade, uma vez que o Oracle possui um otimizador que avalia esses planos e otimiza a execução dessas consultas.

Este assunto tem sido cada vez mais explorado na literatura, principalmente no que se refere à análise das funcionalidades de otimização dos Bancos de Dados Relacionais e à implementação de operadores por similaridade para lidar com dados complexos de forma prática no modelo relacional. Por isso, é importante verificar se é possível alterar o Plano de Execução de consultas por similaridade de forma adequada utilizando os Bancos de Dados Relacionais tradicionais, como o Oracle, que já possuem em sua estrutura ferramentas de otimização.

Além disso, é interessante verificar se é possível implementar regras de equivalência de operadores por similaridade utilizando esses Bancos de Dados, mostrando que não é necessário implementar novas ferramentas para utilizar esses operadores, mas sim fazer pequenas adaptações nos Bancos de Dados já existentes.

Como objetivos específicos será desenvolvido um banco de dados complexo utilizando a álgebra relacional para implementar consultas com foco em operadores de similaridade. Além da implementação das consultas, será realizada uma investigação das ferramentas de otimização já existentes em bancos de dados relacionais.

[1] Os *hints* da Oracle são sugestões ou instruções fornecidas ao otimizador de consultas para influenciar o plano de execução de uma consulta SQL.

2. Referencial Teórico

2.1. Armazenamento de dados complexos através de vetores de características

Inicialmente é preciso entender que os vetores de características são utilizados tanto para representar como também para armazenar as formas numéricas ou simbólicas, conhecidas como recursos de um objeto, de maneira matemática para que seja fácil de ser analisada. Eles são essenciais para diversas áreas de aprendizado de máquina e processamento de padrões. Isso porque os algoritmos de aprendizado de máquina costumam precisar de uma representação numérica de objetos para que os algoritmos realizem o processamento e o estudo estatístico [Liu et al. 2007].

Um vetor de característica bem conhecido são os que descrevem as cores RGB (conhecidas como vermelho, verde e azul). Uma cor pode ser definida pela quantidade de vermelho, azul ou verde que ela possui. Um vetor com aspectos para isso seria cor = [R,G,B] [Wang et al. 2013]. Para uma compreensão melhor é preciso ter em mente que um vetor é uma série de números, como uma matriz que tenha uma coluna e também diversas linhas que podem ser demonstrados de forma espacial. Um recurso é uma propriedade numérica ou simbólica referente a um aspecto do objeto que será armazenado. O vetor de características engloba diversos aspectos de um objeto. Quando adicionamos vetores de características aos objetos criamos um local de recursos [Wang et al. 2013].

Os aspectos podem demonstrar, de modo geral, um simples pixel ou ainda uma imagem. Assim, é possível descrever os aspectos de uma maneira tridimensional com um vetor indicando sua altura, largura, profundidade entre outros [Wang et al. 2013]. Os vetores que possuem aspectos amplos são usados no aprendizado de máquinas por conta de sua eficiência e praticidade de representar objetos de uma maneira numérica para cooperar em diversas formas de análises [Wang et al. 2013].

No armazenamento de imagem, os recursos são inerentes a imagem, possuindo cor, intensidade de escala de cinza, bordas, entre outros. Os vetores de características são, em especial, bastante conhecidos para observar os processamentos de imagens conforme a forma conveniente como os atributos sobre uma imagem, como os exemplos listados, que podem ser comparados numericamente após serem colocados em vetores de características [Chen et al. 2015]. A Figura 1 representa a criação de um vetor de característica.

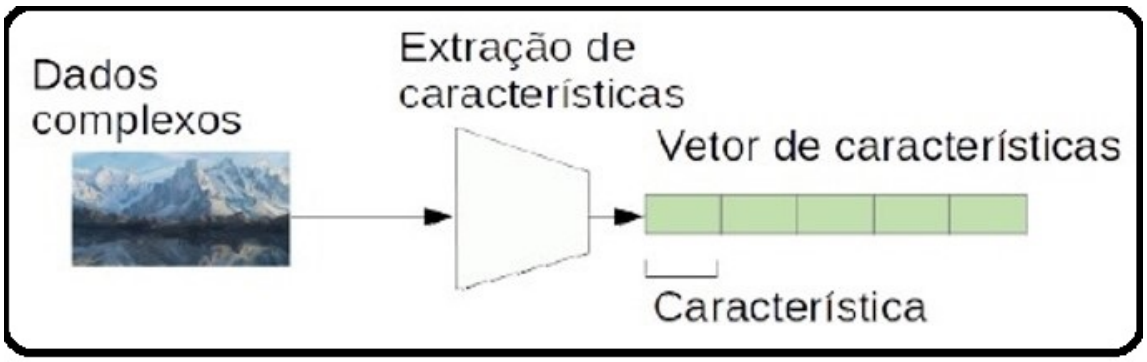

Figura 1. Criando um vetor de características.
Fonte: Própria.

Para o reconhecimento de fala, os recursos conseguem ser comprimentos de som, nível de ruído, taxas de ruído entre outros. Para tentar evitar *spam*, a qualidade de características pode ser elevada. Elas podem possuir localização de IP, estrutura de texto, frequência de algumas palavras ou alguns cabeçalhos de e-mail. Esses vetores de características são utilizados em problemas de classificação, redes neurais artificiais e algoritmos de vizinhos k-mais pertos em aprendizado de máquina [Wang et al. 2013].

Os vetores de características para serem armazenados no computador precisam passar por um procedimento de virtualização do objeto. Após isso, para que as características sejam compactadas no banco de dados da aplicação de maneira correta, serão realizados processos de mudanças e seleção de características, onde eles possuem como foco a diminuição dessas características [Joshi and Haspel 2020].

2.2. Operadores de dados complexos por similaridade

2.2.1. Funções de Distância

O cálculo de similaridade/dissimilaridade entre os vetores de características é feito por meio de funções de distância, considerando um cálculo real não negativo, ou seja, quanto menor, esse valor mais similar são os objetos observados. Há várias formas de distância catalogadas na literatura [Pola 2010], estabelecidas para espaços multidimensionais, dados dimensionais e também para dados adimensionais, que são aqueles dados que não tem dimensão estabelecida. Certas funções de distância são em especial interessantes, já que permitem a definição de espaços métricos. A Figura 2 ilustra o funcionamento de uma função de distância.

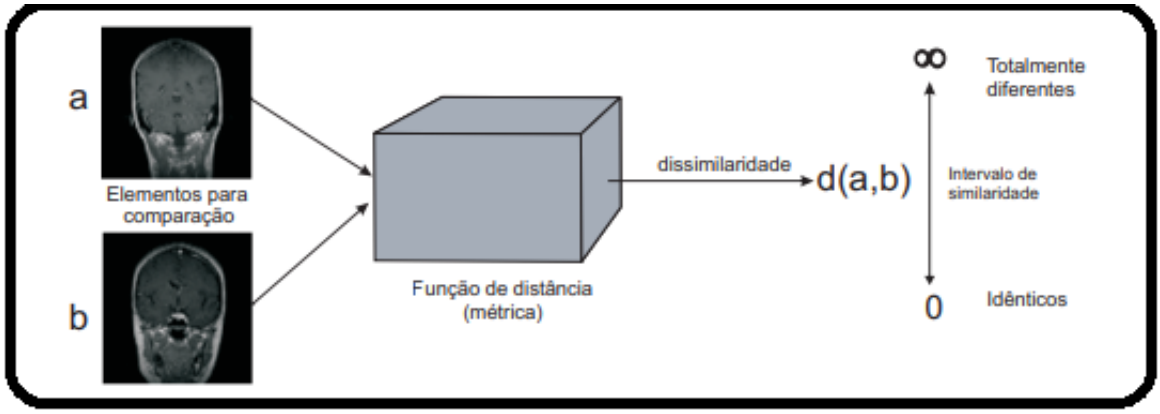

Figura 2. Função de distância.
Fonte: [Pola 2010].

Importante compreender que um espaço métrico é formalmente estabelecido da seguinte forma: $M = \langle S, \delta \rangle$, onde S é um domínio de dados e δ é uma função de distância. Um ponto que deve ser observado é que essas funções de distância atuam com dados numéricos dimensionais, como as distâncias da família Minkowski que abrange as distâncias Manhattan (L_1), a distância Euclidiana (L_2) e a distância Chebyshev (L_∞) [Cuellar 2021]. A Figura 3 ilustra a formação de um espaço métrico.

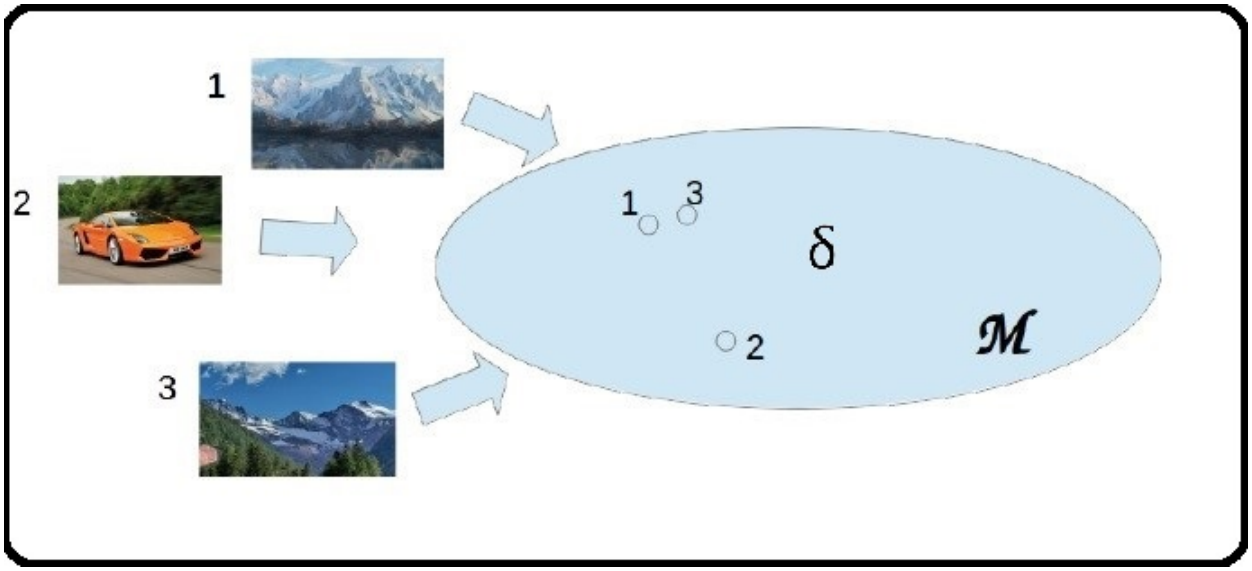

Figura 3. Espaço Métrico.
Fonte: Própria.

2.2.2. Distância Manhattan

A distância Manhattan é uma métrica utilizada para medir a distância entre dois pontos em um espaço bidimensional (ou multidimensional), considerando os deslocamentos horizontais e verticais, como se estivesse navegando pelas ruas de uma cidade [Levitin 2011].

Matematicamente, a fórmula para calcular a distância Manhattan entre dois pontos $P(x_1, y_1)$ e $Q(x_2, y_2)$ em um espaço bidimensional é representado pela fórmula:

$$D(P, Q) = |x_1 - x_2| + |y_1 - y_2|$$

Nessa fórmula, $|x_1 - x_2|$ representa a diferença absoluta entre as coordenadas x dos pontos P e Q, e $|y_1 - y_2|$ representa a diferença absoluta entre as coordenadas y desses dois pontos. A soma dessas diferenças resulta na distância de "caminhada de cidade"ou distância manhattan entre os pontos [Levitin 2011].

A Distância Manhattan tem aplicações em diversas áreas, como análise de dados, reconhecimento de padrões e ciência da computação. Em algoritmos de análise de similaridade, por exemplo, essa métrica pode ser usada para quantificar a diferença entre observações com base em seus atributos.

2.2.3. Consultas por Similaridade

O procedimento de processamento de dados complexos ocorre por meio de abstrações não triviais sobre seu conteúdo num todo [Barioni et al. 2011]. Isso faz com que ocorra o problema da representação das consultas por similaridade. Segundo Barioni et al pode-se utilizar duas estratégias para representar as consultas por similaridade, sendo elas: a abordagem dos operadores por meio do conteúdo ou a abordagem dos operadores por meio da similaridade.

Seja qual for a opção, podem ser exemplificadas por meio das consultas: regresse aos exames radiológicos que demonstrem a massa de tamanho que não seja exatamente normal ou regresse às imagens que possuam o pôr do sol no mar. [Barioni et al. 2011]. A abordagem utilizando operadores conforme o conteúdo estabelece operadores próprios para cada conceito em particular. Por meio dela, para executar a primeira consulta o sistema necessita de duas especificações, sendo a primeira que encontram padrões na radiografia e que demonstrem a massa e a segunda que calcula a área da massa. Já a segunda consulta precisa de uma função que demonstre o sol e o mar em uma imagem e outra que reconhece a disposição espacial entre os objetos (para demonstrar se cada um deles está realmente no sul, norte, leste ou oeste) e assim determinar se a imagem representa um pôr do sol [Barioni et al. 2011].

Com isso, cada um das funções necessita possuir um operador correspondente. Ela consegue ser adaptada para sistemas que foquem mais em problemas específicos, porém, o número de operadores pode ocorrer de forma bastante grande, já que no caso de dados multimídia seria preciso fazer um ou mais operadores para cada domínio de imagens para criar uma semântica bem abrangente [Barioni et al. 2011].

A execução de buscas que tenham vários operadores acaba ficando bem complicada, o que dificulta consideravelmente a sua escrita por parte do usuário. Também é preciso muito mais tempo para ser gasto ao aprender sobre as diversas linguagens existentes que serão próprias para cada domínio de aplicações [Barioni et al. 2011].

No entanto, a abordagem dos operadores através da similaridade estabelece um grupo relativamente pequeno de operadores que podem ser usados em qualquer domínio de dados complexos. Esses operadores computam a similaridade entre um grupo de elementos de dados na consulta e os que estão armazenados no banco de dados, para analisar quais irão ajudar em certa condição de busca. Por meio desses operadores pode-se representar as consultas um e dois, onde a aplicação precisa avaliar o extrator de características e a função de distância mais correta para cada uma delas. O ponto positivo dessa abordagem é que por meio de um número bastante restrito de operadores de consulta pode-se representar pesquisas que envolvem um elevado número de domínios de dados complexos. É preciso somente o ajuste no cálculo de similaridade para resolver o problema em questão. Com as vantagens em se obter operadores por similaridade essa estratégia é bastante usada na literatura e será ponderada na próxima seção [Barioni et al. 2011].

2.3. Propostas de Inclusão de Operadores por Similaridade na Álgebra Relacional

Há três linhas principais que procuram acrescentar operadores por similaridade no modelo relacional. A primeira estabelece a estrutura algébrica com a ideia de lógica *fuzzy*. Essa costuma trazer a ideia de imprecisão [Penzo 2005].

Outra linha compreende o modelo relacional com o intuito de relações ordenadas (*ranked relations*), onde as tuplas e os atributos são classificados conforme um critério de *ranking* estabelecido pelo usuário. O conceito é ordenar as relações de entrada de maneira onde os operadores de consulta possam preservar essa ordem e voltem nos top-k elementos que ajudam a melhorar e assim a satisfazer o critério definido [Adali et al. 2004].

A terceira linha compreende a utilização de operadores por similaridade e foi a abordagem utilizada nesse artigo. Ele não altera as ideias do modelo relacional, porém, inclui diversos novos operadores a ele. Esses operadores conseguem ser divididos em

dois tipos de categorias, sendo essas categorias: as que usam um limiar de dissimilaridade para escolher os resultados que almeja e aqueles que retornam os elementos mais parecidos considerando o critério de similaridade estabelecido, delimitando-se no tamanho da resposta pelo número de elementos K [Adali et al. 2004]. As subseções abaixo demonstram um conjunto de operadores por similaridade que estão entre os mais essenciais já demonstrados.

2.3.1. Operadores de seleção e junção por similaridade

Segundo o autor [Oliveira 2010] há diversos tipos de operadores de consulta por similaridade, baseados na função da distância de um ou de vários centros. Os mais utilizados são os que usam as funções de distância de um centro. Eles, por serem mais conhecidos, são os seguintes: A procura por abrangência (*Range Query - Rq*), que ajuda a recuperar os elementos armazenados onde a dessemelhança de um elemento de consulta é inferior ou igual a certo limite de um centro de buscas, conforme pode ser observado na Figura 4 (a) e a busca pelos k vizinhos mais perto (*k-Nearest Neighbor query – k-NNq*) conforme a Figura 5 (a), que volta os K elementos mais próximos do elemento de consulta. [Oliveira 2010]

Em ambos os casos, os centros de busca são demonstrados pelas bolas pretas e a união resposta é dada pelos elementos que estão a um raio r dos mesmos.

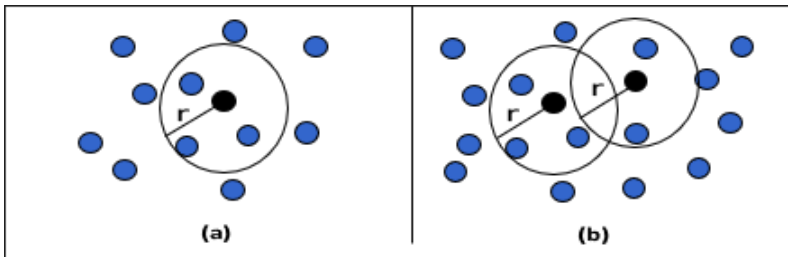

Figura 4. (a) Seleção por abrangência e (b) a união por abrangência.
Fonte: Própria.

Nos três exemplos, k = 3, os elementos da primeira relação de entrada são demonstrados pelas bolas pretas e os da segunda relação de entrada pelas bolas azuis e as setas mostram para os elementos que constituem o conjunto de resposta.

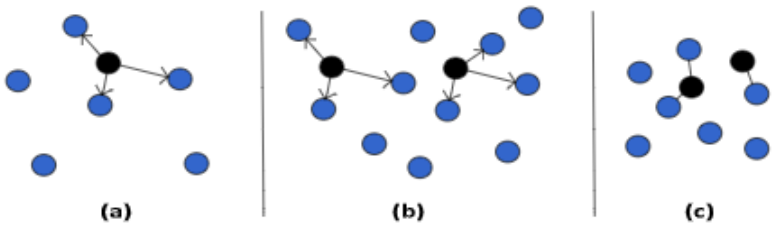

Figura 5. (a) Seleção k-NN, (b) União k-NN e (c) União k-CN.
Fonte: Própria.

Os operadores básicos trazidos acima são empregados para gerar consultas por similaridade. A união de similaridade são formas de mesclar dados complexos de dois conjuntos de dados de entrada conforme o critério de similaridade estabelecido. Alguns exemplos dela são os seguintes: a união por abrangência (*range join* – Figura 4(b)), a união dos k vizinhos mais próximos (*k - Nearest Neighbor join – onkNN-0* Figura 5(b)) e a união dos k pares mais próximos (*k-Closest Pair join – onkCN* – Figura 5(c)). O objetivo é conseguir todos os elementos armazenados que possuem certo elemento de referência, como, por exemplo, um dos k elementos mais parecidos [Oliveira 2010].

2.4. *Framework* SCRUM

O Scrum, um dos *frameworks* ágeis mais reconhecidos, sendo introduzido por Jeff Sutherland e Ken Schwaber em 1990 como uma abordagem para o desenvolvimento de *softwares* complexos. É baseado em princípios de colaboração, inspeção e adaptação. O Scrum oferece uma estrutura flexível e adaptável para lidar com desafios ligados à gestão de projetos, construindo ambientes dinâmicos de desenvolvimento. Para o funcionamento eficaz, o Scrum possui artefatos, como o "*Product Backlog*". Este é um componente crucial que representa uma lista priorizada de funcionalidades, requisitos e melhorias desejadas para o produto final [Schwaber and Sutherland 2020]. Segundo os autores, no scrum temos 3 papéis essenciais bem definidos, são eles:

- **Product Owner:** *Product Owner* ou "proprietário do produto"é o papel responsável por manter a relação entre a equipe de desenvolvimento e os *stakeholders*[2]. Seu papel também inclui definir as necessidades do cliente/produto, priorizar o *backlog* e garantir que o valor seja entregue.
- **Scrum Master:** O Scrum Master é responsável por guiar todo o processo do scrum, ele funciona como um facilitador retirando impedimentos de toda a equipe. Em resumo, sua função é garantir que a equipe siga as boas práticas do scrum.
- **Equipe de Desenvolvimento:** A equipe de desenvolvimento é composta por uma equipe multidisciplinar e autogerenciável. Ela converte os itens do *backlog* em incrementos de valor ao produto entregáveis durante cada *sprint*[3].

O processo do Scrum é dividido em reuniões onde há uma interação entre todos os papéis envolvidos para gerenciar cada etapa de desenvolvimento do produto. As reuniões são:

- **Sprint Planning:** Essa reunião demarca o começo de cada *sprint*. Nesse momento, será discutido os items do *backlog* que serão priorizados na próxima *sprint* de desenvolvimento. É também traçado um plano para alcançar o objetivo definido.
- **Daily Scrum:** É uma reunião diária cujo objetivo é passar uma atualização do progresso obtido pela equipe. Nessa reunião também é relatado as dificuldades e impedimentos, bem como o plano de ação para o dia.
- **Sprint Review:** Ao final de cada *sprint* é realizado a *sprint review*. O objetivo dessa reunião é compartilhar *feedbacks* entre todos os membros envolvidos. Esse *feedback* auxilia na adaptação do próximo *backlog*.

[2]*Stakeholders* são todos os indivíduos, grupos ou entidades que têm interesse e/ou são afetados de alguma forma por uma organização, projeto ou decisão.

[3]Uma *sprint* se refere a um período de tempo definido durante o qual uma equipe trabalha em um conjunto específico de tarefas. Geralmente, são curtas.

- **Sprint Retrospective:** Essa reunião demarca o final de cada ciclo de desenvolvimento. Nela é realizado uma reflexão entre todos os membros envolvidos com o objetivo em identificar melhorias no processo.

O Scrum permite uma abordagem ágil para a gestão de projetos, proporcionando a colaboração, a flexibilidade e a entrega de valor contínua. Seus princípios, papéis e reuniões trabalham em conjunto para garantir a adaptação eficaz em ambientes de desenvolvimento dinâmicos. Ao utilizar o Scrum, as empresas podem melhorar a qualidade do seu produto final, a satisfação do cliente e a eficiência do processo de desenvolvimento [Schwaber and Sutherland 2020].

3. Trabalhos Correlatos

A análise e recuperação de dados complexos baseados em similaridade têm sido objeto de estudo em diversas pesquisas ao longo do tempo. Esta seção tem como objetivo apresentar uma revisão mais detalhada dos trabalhos relacionados, que oferecem uma base sólida para o desenvolvimento do artigo em questão. Foi analisado estudos sobre a recuperação de dados por similaridade, a utilização de operadores por abrangência e dos k vizinhos mais próximos, bem como a aplicação de *Hints* em bancos de dados Oracle.

3.1. SimDB

[Silva et al. 2010] propuseram o SimDB, um sistema de gerenciamento de bancos de dados com suporte nativo para consulta por similaridade. O SimDB estende as funcionalidades dos SGBDs (Sistemas Gerenciadores de Bancos de Dados) tradicionais, permitindo consultas que levem em consideração a similaridade entre os dados. O SimDB utiliza uma abordagem baseada em operadores de similaridade e algoritmos de agrupamento para recuperar dados complexos. A pesquisa de Silva et al demonstrou como o SimDB pode ser usado para melhorar a eficiência e a precisão das consultas por similaridade, além de fornecer um ambiente de prototipagem para o desenvolvimento de novas abordagens de similaridade.

3.2. SIREN

[Barioni et al. 2005] propuseram o SIREN, um método para consulta de objetos complexos por similaridade utilizando SQL. Os autores introduziram uma série de novas cláusulas e funções SQL, permitindo a recuperação de dados baseada em medidas de similaridade. Os autores demonstram como o SIREN pode ser aplicado em diversos cenários, incluindo a recuperação de informações multimídia e a análise de séries temporais. Além disso, os autores também abordaram o uso de operadores de abrangência e dos k vizinhos mais próximos na recuperação de dados por similaridade. O SIREN representa uma importante contribuição para a integração de consultas por similaridade em SGBDs relacionais, permitindo a utilização de uma linguagem amplamente conhecida como SQL para tratar problemas complexos de recuperação de dados.

3.3. Consultas por similaridade usando SGBDs relacionais

[Siqueira et al. 2018] exploraram como as consultas por similaridade podem ser realizadas usando SGBDs relacionais padrão e a linguagem SQL. Os autores apresentaram uma série de abordagens e técnicas que permitem a execução de consultas por similaridade

usando recursos existentes em SGBDs relacionais, sem a necessidade de extensões ou modificações específicas. Entre as técnicas apresentadas estão o uso de funções de agregação, operadores de janela e funções analíticas. Siqueira et al mostrou que é possível alcançar resultados satisfatórios em termos de eficiência e precisão, mesmo utilizando SGBDs relacionais convencionais.

3.4. Equivalência de sentenças com operadores de similaridade

No artigo de [Silva et al. 2013], os autores abordam a equivalência de sentenças que utilizam operadores por similaridade. Eles investigam o conceito de avaliação de consultas por similaridade, a transformação dessas consultas e a otimização do processamento em um ambiente de banco de dados.

O estudo de [Silva et al. 2013] analisa a equivalência de sentenças com operadores de similaridade, explorando as possíveis transformações e reescritas de consultas por similaridade. Os autores apresentam uma taxonomia de consultas por similaridade, classificando-as em três categorias: (1) consultas baseadas em similaridade de objetos, (2) consultas baseadas em similaridade de atributos e (3) consultas baseadas em similaridade de valores. Essa taxonomia ajuda a entender e identificar as possíveis transformações e otimizações que podem ser aplicadas a consultas por similaridade.

[Silva et al. 2013] também exploram regras de equivalência e transformações de consultas por similaridade, incluindo a reescrita de consultas e a aplicação de técnicas de otimização de consultas. Eles apresentam uma série de regras que podem ser usadas para transformar consultas por similaridade, a fim de melhorar a eficiência do processamento. Algumas dessas regras incluem a eliminação de redundâncias, a aplicação de técnicas de seleção e junção, e a utilização de estruturas de indexação para otimizar a recuperação de dados por similaridade.

Em resumo, os trabalhos relacionados apresentam uma variedade de abordagens e técnicas para a recuperação de dados complexos baseados em similaridade. A integração dessas abordagens em bancos de dados relacionais, como o Oracle, pode melhorar significativamente o desempenho e a precisão das consultas. O desenvolvimento de métodos e *frameworks*, como o SimDB e SIREN, demonstra o avanço no campo de pesquisa de recuperação de dados por similaridade e a crescente importância de tais técnicas para lidar com desafios em cenários de análise de dados complexos. Este artigo se concentra na implementação de operadores de similaridade utilizando SQL padrão, sem a utilização de nenhum tipo de extensão. Ele utiliza apenas ferramentas já presentes no banco de dados Oracle e adota uma abordagem exploratória para analisar a possibilidade de otimização dessas consultas em um banco de dados relacional levando em consideração os estudos apresentados nessa seção.

4. Metodologia

Foi realizado um estudo de caso baseado em um trabalho de pesquisa sobre dados complexos, que consistiu na implementação de consultas por similaridade em bancos de dados relacionais. A escolha do banco de dados Oracle foi devido ao seu ótimo desempenho, escalabilidade e por possuir recursos avançados para a linguagem SQL. As consultas foram desenvolvidas utilizando a linguagem SQL e PL/SQL, empregando os operadores de abrangência e os K-Vizinhos mais próximos. Foi feita também uma investigação para entender o funcionamento do otimizador da Oracle, que realiza a otimização das consultas

em tempo de execução com base no custo de processamento. Essa análise serviu para demonstrar que além de ser possível implementar consultas por similaridade no modelo relacional, bancos de dados robustos como o Oracle também são capazes de otimizar essas consultas e gerar planos de execução alternativos. Para isso, foram utilizados *Hints*, ferramenta já existente no banco de dados Oracle. Os *Hints* são comentários que podem ser adicionados junto a instruções SQL, fornecendo sugestões ao otimizador do banco de dados Oracle. Essas sugestões fornecidas ao otimizador permitem manipular o plano de execução da *query* para forçar um caminho de processamento alternativo. O plano de execução das consultas foi analisado e alterado utilizando *Hints*, de modo a verificar se é possível alterar o plano de execução do otimizador.

Este projeto foi gerenciado por meio de elementos da metodologia SCRUM. Foram definidos os requisitos/objetivos necessários para implementar as consultas por similaridade, que fizeram parte do *product backlog*. Estes requisitos tiveram tempo de implementação previstos fixos que foram divididas em *sprints* de no máximo 15 dias. Desta forma foi possível utilizar o modelo de desenvolvimento ágil para gerenciar o tempo de implementação do projeto. O objetivo é seguir o fluxograma da Figura 6, que resume o processo que será explorado na seção de Desenvolvimento do Trabalho.

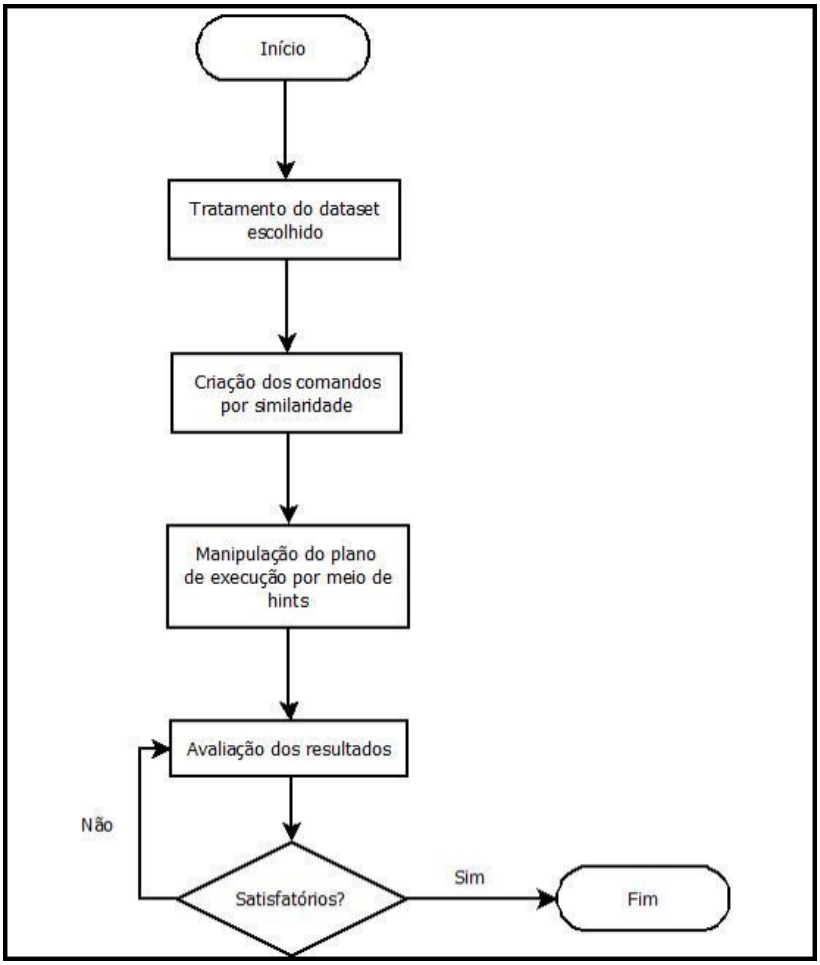

Figura 6. Fluxograma de implementação do projeto.
Fonte: Própria.

5. Desenvolvimento do trabalho

5.1. Metodologia SCRUM

A metodologia SCRUM é uma abordagem iterativa e incremental para o desenvolvimento de *software* que se baseia em ciclos curtos de trabalho chamados *Sprints*. Cada *Sprint* tem como objetivo entregar um incremento de valor ao produto final. Durante o projeto, as atividades foram organizadas em um *backlog*, que é uma lista priorizada de tarefas a serem realizadas. As tarefas foram selecionadas e alocadas para cada *Sprint* de acordo com sua prioridade e complexidade. Na Figura 7 é apresentado o *backlog* contendo os requisitos para implementação das tarefas.

TASK ID	TASK NAME	SPRINT	START	FINISH	PRIORITY	STATUS	STORY POINTS
1	Identificação de um dataset da área médica adequado	Sprint 1	01/01/2023	10/01/2023	Alta	Concluído	3
2	Tratamento dos dados para inserção no banco de dados	Sprint 2	16/01/2023	20/01/2023	Alta	Concluído	2
3	Criação das tabelas no banco de dados Oracle com a coluna fv do tipo varray	Sprint 3	01/02/2023	10/02/2023	Alta	Concluído	5
4	Inserção dos dados tratados no banco de dados (Comandos de insert)	Sprint 4	11/02/2022	20/02/2022	Alta	Concluído	5
5	Implementação da função PL/SQL manhattan_distance_varray (Função de distância)	Sprint 5	05/03/2023	20/03/2023	Alta	Concluído	21
6	Desenvolvimento das consultas SQL (Range query)	Sprint 6	01/04/2023	15/04/2023	Média	Concluído	13
7	Desenvolvimento das consultas SQL (K-NN)	Sprint 7	01/05/2023	15/05/2023	Média	Concluído	13
8	Manipulação do plano de execução utilizando Hints	Sprint 8	05/06/2023	20/06/2023	Baixa	Concluído	5
9	Verificação dos planos de execução e análise dos resultados	Sprint 9	10/06/2023	25/06/2023	Baixa	Concluído	8

Figura 7. Product Backlog.
Fonte: Própria.

5.2. Identificação e tratamento do *dataset*

O primeiro passo no desenvolvimento do artigo foi identificar um *dataset* de imagens médicas que já tivesse passado pelo processo de extração e seleção de características, o qual pudesse ser utilizado para implementar as consultas por similaridade. O *dataset* selecionado foi retirado do *site* kaggle [4], uma plataforma para aprendizado de ciência de dados que contém inúmeros *datasets* que podem ser utilizados pela comunidade de programação em diversos tipos de soluções. O *dataset* pertence a um repositório de *Machine learning* chamado UC Irvine [UCI Machine Learning Repository 2016]. Após encontrar um *dataset* adequado, os dados foram tratados e preparados para serem inseridos no banco de dados Oracle. Esse processo envolveu a remoção de dados inconsistentes utilizando o Microsoft Excel e a conversão de formatos para criar os comandos *insert* na linguagem SQL, de forma a possibilitar o armazenamento e o processamento adequados.

[4]https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data

5.3. Criação das tabelas e inserção dos dados

As tabelas foram criadas no banco de dados Oracle com a coluna fv sendo um varray, que representa um vetor de características de imagem médica. O uso do tipo varray permite armazenar *arrays* de tamanho variável, facilitando a representação de dados complexos, como vetores de características. Com os dados tratados e as tabelas criadas, o próximo passo foi inserir os dados no banco de dados. Esse processo envolveu a criação de *scripts* SQL para inserção dos dados, garantindo a integridade e a consistência dos dados armazenados. O código utilizado para a criação da tabela e inserção dos dados é apresentado respectivamente nas Figuras 8 e 9 a seguir.

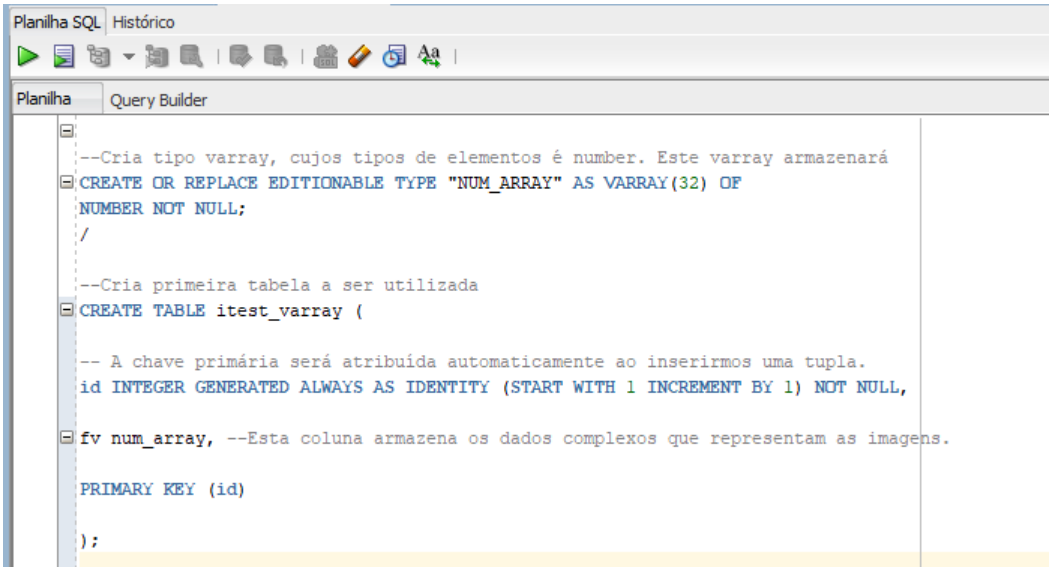

Figura 8. Criação da tabela e variável Varray.
Fonte: Própria.

```
-- Insere os dados na tabela itest_varray.

INSERT INTO itest_varray (fv) values ( NUM_ARRAY(17.99,10.38,122.8,1001,0.1184,0.2776,0.3001,0.1471,0.241
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(20.57,17.77,132.9,1326,0.08474,0.07864,0.0869,0.07017,0.
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(19.69,21.25,130,1203,0.1096,0.1599,0.1974,0.1279,0.2069,
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(11.42,20.38,77.58,386.1,0.1425,0.2839,0.2414,0.1052,0.25
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(20.29,14.34,135.1,1297,0.1003,0.1328,0.198,0.1043,0.1809
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(12.45,15.7,82.57,477.1,0.1278,0.17,0.1578,0.08089,0.2087
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(18.25,19.98,119.6,1040,0.09463,0.109,0.1127,0.074,0.1794
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(13.71,20.83,90.2,577.9,0.1189,0.1645,0.09366,0.05985,0.2
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(13,21.82,87.5,519.8,0.1273,0 .1932,0.1859,0.09353,0.235,
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(12.46,24.04,83.97,475.9,0.1186,0.2396,0.2273,0.08543,0.2
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(16.02,23.24,102.7,797.8,0.08206,0.06669,0.03299,0.03323,
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(15.78,17.89,103.6,781,0.0971,0.1292,0.09954,0.06606,0.18
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(19.17,24.8,132.4,1123,0.0974,0.2458,0.2065,0.1118,0.2397
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(15.85,23.95,103.7,782.7,0.08401,0.1002,0.09938,0.05364,0
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(13.73,22.61,93.6,578.3,0.1131,0.2293,0.2128,0.08025,0.20
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(14.54,27.54,96.73,658.8,0.1139,0.1595,0.1639,0.07364,0.2
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(14.68,20.13,94.74,684.5,0.09867,0.072,0.07395,0.05259,0.
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(16.13,20.68,108.1,798.8,0.117,0.2022,0.1722,0.1028,0.216
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(19.81,22.15,130,1260,0.09831,0.1027,0.1479,0.09498,0.158
INSERT INTO itest_varray (fv) values ( NUM_ARRAY(13.54,14.36,87.46,566.3,0.09779,0.08129,0.06664,0.04781,
```

Figura 9. Armazenando os dados em Varrays.
Fonte: Própria.

As informações inseridas no vetor correspondem as características inerentes dos dados necessárias para representar determinada imagem dentro do banco de dados. O *dataset* retirado do *site* Kaggle representa um conjunto de dados de câncer de mama da Universidade de Wisconsin [Wolberg and Street 1995]. Foram inseridos cerca de 13.000 vetores de características, sendo cada vetor contendo 32 características do câncer. Os dados inseridos contém informações n-dimensionais sobre o câncer, contendo informações como, por exemplo, raio médio, textura média, perímetro médio, compacidade média, simetria média, dentre outros dados que representam o câncer de forma espacial.

5.4. Implementação da função PL/SQL *Manhattan Distance Varray*

Com os dados inseridos no banco de dados, a função PL/SQL *manhattan distance varray* foi criada para calcular a distância Manhattan entre dois vetores de características. Como apresentado anteriormente no referencial teórico, para se calcular a similaridade/dissimilaridade entre dois vetores de características é necessário utilizar uma função de distância. O bloco de código apresentado na Figura 10 representa a função em PL/SQL chamada manhattan_distance_varray, que calcula a distância Manhattan entre dois *arrays* numéricos (Varrays). A distância Manhattan é uma métrica de distância entre dois pontos em um espaço bidimensional ou multidimensional, medida pela soma das diferenças absolutas entre suas coordenadas. A implementação dessa função foi de extrema importância para o correto funcionamento das consultas por similaridade, uma vez que ela calcula a similaridade dos objetos armazenados e os ordena com base nisso.

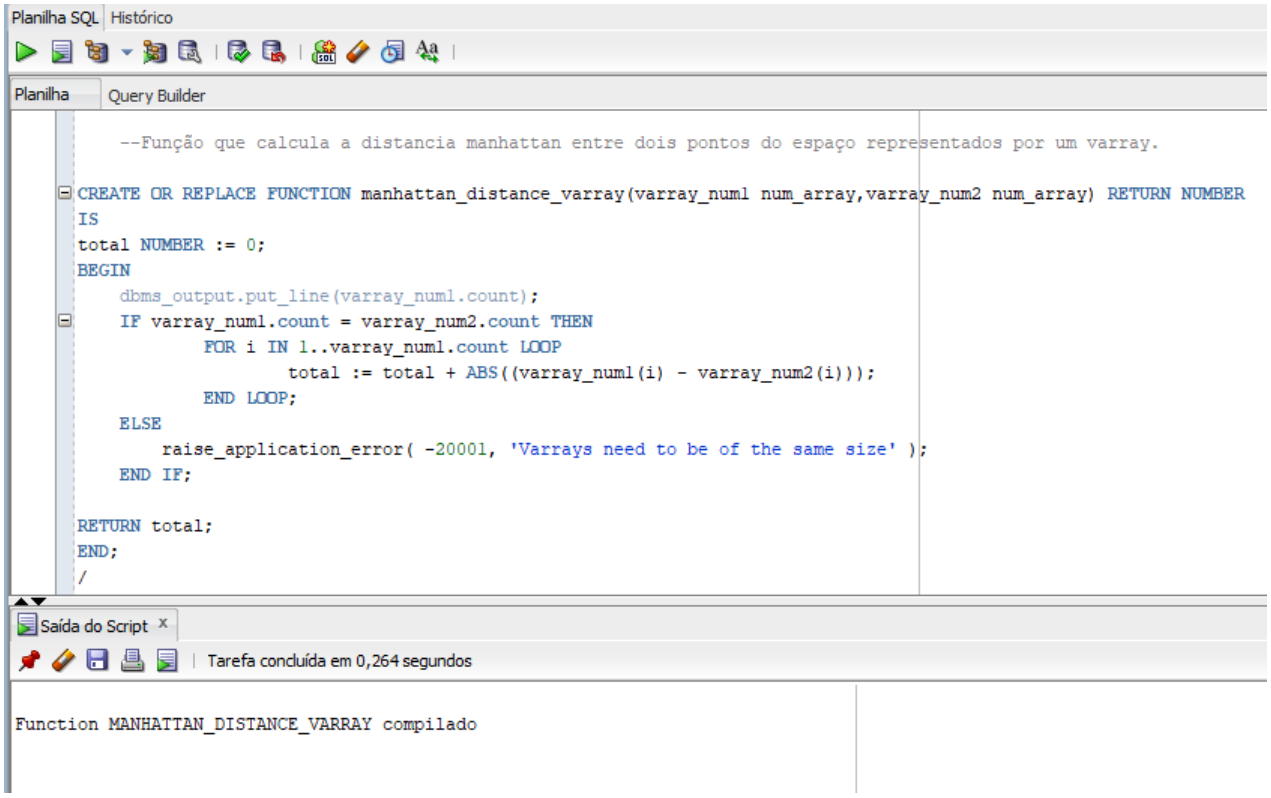

Figura 10. Função Manhattan.
Fonte: Própria.

A função **manhattan_distance_varray** é definida com dois parâmetros de entrada: **varray_num1** e **varray_num2**, ambos do tipo **num_array** (*Varray* definido na Figura 8). A declaração **IS** marca o início do bloco de código da função. É declarada uma variável **total** do tipo **NUMBER**, inicializada com o valor zero. Essa variável será usada para armazenar a soma das diferenças absolutas entre os elementos dos dois Varrays.

A estrutura de controle **IF** verifica se os Varrays **varray_num1** e **varray_num2** têm o mesmo número de elementos (tamanho). Se eles tiverem o mesmo tamanho, o código entra no bloco **THEN**. Dentro do bloco **THEN**, um **loop FOR** é usado para iterar sobre os elementos dos *Varrays*. O loop percorre de 1 até o número de elementos nos *Varrays*, usando **varray_num1.count** como limite.

Para cada iteração, a diferença absoluta entre os elementos correspondentes dos dois Varrays é calculada usando **ABS(varray_num1(i) - varray_num2(i))**, e esse valor é adicionado à variável **total**.

Se os Varrays não tiverem o mesmo tamanho, o código entra no bloco **ELSE**. Dentro do bloco **ELSE**, duas linhas de código utilizam **dbms_output.put_line** para imprimir o número de elementos nos Varrays **varray_num1** e **varray_num2**. Isso é útil para depuração. Em seguida, é utilizado o procedimento **raise_application_error** para lançar um erro personalizado com o código de erro **-20001** e a mensagem *"Varrays need to be of the same size"*. Isso é feito para indicar que os Varrays precisam ter o mesmo tamanho para calcular a distância Manhattan.

O bloco **END IF** fecha a estrutura de controle **IF**. O código retorna o valor da variável **total**, que é a distância Manhattan calculada. O bloco **END** marca o final da função. Resumindo, essa função calcula a distância Manhattan entre dois Varrays numéricos, somando as diferenças absolutas entre seus elementos correspondentes. Se os Varrays não tiverem o mesmo tamanho, um erro é gerado indicando que eles precisam ter o mesmo tamanho para realizar o cálculo.

5.5. Desenvolvimento das consultas por similaridade

O operador de abrangência pode ser executado por meio da consulta fornecida como exemplo na Figura 11 (onde o centro da consulta é o elemento com o ID 1 e o raio da abrangência do operador é 100). Nessa consulta, estamos realizando uma seleção por abrangência na tabela Itest_Varray, passando como parâmetro para a função de distância uma abrangência de 100.

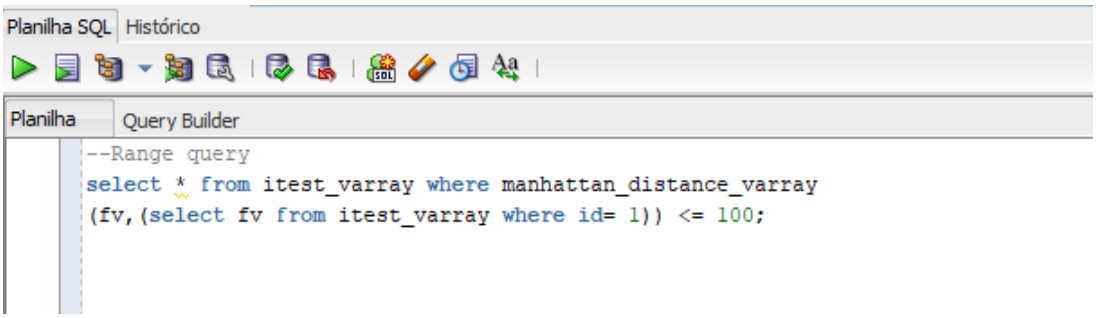

Figura 11. *Range Select* (Seleção por Abrangência).
Fonte: Própria.

Select id from itest_varray: aqui temos um comando que seleciona o campo **id** da tabela **itest_varray**. Onde **manhattan_distance_varray(fv, (select fv from itest_varray where id=1)) <= 100**: neste ponto, a cláusula **WHERE** filtra os resultados com base na condição **manhattan_distance_varray(fv, (select fv from itest_varray where id=1))**, que calcula a distância Manhattan entre o valor do campo fv na linha atual e o valor do campo fv na linha da tabela **itest_varray** onde o **id** é igual a 1. O operador **<= 100** verifica se a distância Manhattan calculada é menor ou igual a 100. Portanto, este código seleciona os IDs das linhas na tabela itest_varray onde a distância Manhattan entre o campo **fv** dessas linhas e o campo **fv** da linha com **id** igual a 1 é menor ou igual a 100. Isso recupera as linhas que estão relativamente próximas, de acordo com a métrica de distância Manhattan.

Já o operador KNN pode ser executado com o comando apresentado na Figura 12. Essa consulta representa uma seleção KNN que retorna os dez elementos mais próximos da tabela itest_varray do elemento itest_varray cujo id é igual a 5.

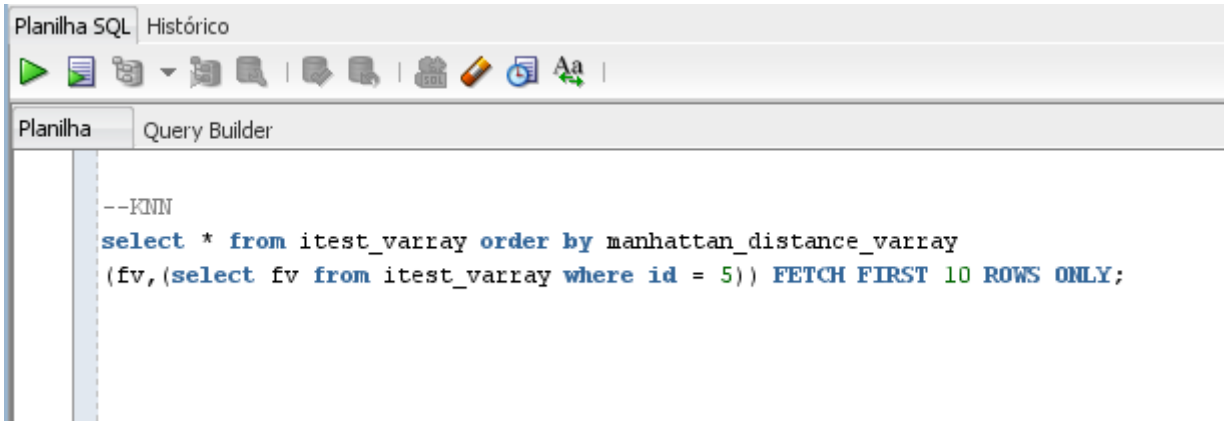

Figura 12. K-NN Select (Seleção K-NN).
Fonte: Própria.

Essa consulta SQL está selecionando dados de uma tabela chamada **itest_varray** e ordenando os resultados com base na função **manhattan_distance_varray**. A seguir, uma explicação do que cada parte do código está fazendo: **select * from itest_varray**: Esse comando seleciona todas as colunas da tabela itest_varray. **Order by manhattan_distance_varray (fv, (select fv from itest_varray where id = 5))**: aqui, a cláusula order by está ordenando os resultados com base no valor retornado pela função **manhattan_distance_varray**. **Fv** é um campo da tabela itest_varray. **(Select fv from itest_varray where id = 5)** é uma subconsulta que seleciona o valor do campo fv da linha na tabela itest_varray onde o id é igual a 5. A função manhattan_distance_varray é chamada com esses dois argumentos (fv e o valor retornado pela subconsulta), e o resultado dessa função é usado para ordenar os resultados. **FETCH FIRST 10 ROWS ONLY**: esta parte limita o número de resultados retornados para apenas 10 linhas. Em resumo, o código está selecionando todas as colunas da tabela itest_varray, ordenando os resultados com base na distância Manhattan calculada pela função manhattan_distance_varray entre o valor do campo fv na linha atual e o valor do campo fv da linha com id igual a 5. Em seguida, ele limita o resultado a apenas as 10 primeiras linhas ordenadas.

Podemos notar que ambos possuem quase a mesma estrutura, exceto pela forma em que a subconsulta será filtrada. No KNN, diferente do *Range Query*, onde a subconsulta é filtrada pela cláusula WHERE, na qual passamos um parâmetro de abrangência para nossa função de distância buscar os dados, no KNN utilizamos a cláusula ORDER BY. Isso ocorre justamente porque, no KNN, como não está sendo passado nenhum parâmetro de distância, a função deve calcular todas as distâncias para depois ordenar e através do operador retornar os K elementos mais próximos do meu elemento de consulta utilizando a instrução FETCH FIRST ROWS ONLY.

Em seguida, foi desenvolvido um *Range Query* complexo. Uma consulta SQL complexa envolve instruções que incluem múltiplas tabelas e/ou condições complexas para extrair informações específicas de um banco de dados. Isso requer operações avançadas como junções, subconsultas, agrupamentos, funções de agregação e cláusulas condicionais complexas. Nela iremos analisar o plano de execução e utilizar os *hints* da Oracle afim de manipular o plano de execução. Essa query mescla dados de dois conjuntos de dados e passa para a função Manhattan dois raios de abrangência. A tabela min_itest representa o segundo conjunto de dados, ela foi criada contendo 199 vetores de características da tabela itest_varray. A Figura 13 exibe o código da *query*.

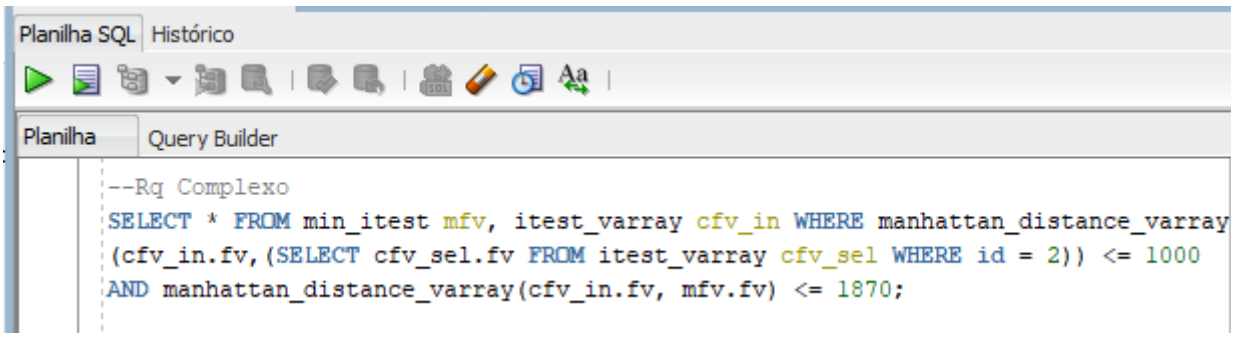

Figura 13. *Range Query* complexo.
Fonte: Própria.

Podemos observar que, nesta consulta, estamos realizando duas operações diferentes. Primeiramente, executamos uma seleção por abrangência na tabela itest_varray e, posteriormente, efetuamos uma junção por abrangência entre as tabelas itest_varray e min_itest. Segue uma explicação mais detalhada do código:

SELECT * FROM min_itest mfv, itest_varray cfv_in: este trecho seleciona todas as colunas das tabelas **min_itest** e **itest_varray**. Os apelidos **mfv** e **cfv_in** são usados para referenciar essas tabelas nas condições subsequentes.

WHERE manhattan_distance_varray(cfv_in.fv, (SELECT cfv_sel.fv FROM itest_varray cfv_sel WHERE id=2)) <= 1000: aqui, a cláusula **WHERE** filtra os resultados com base nas seguintes condições: **manhattan_distance_varray(cfv_in.fv, (SELECT cfv_sel.fv FROM itest_varray cfv_sel WHERE id=2))** calcula a distância Manhattan entre o valor do campo **fv** na linha atual da tabela itest_varray e o valor do campo **fv** da linha na tabela itest_varray onde o **id** é igual a 2. **<= 1000** verifica se a distância Manhattan calculada é menor ou igual a 1000.

AND manhattan_distance_varray(cfv_in.fv, mfv.fv) <= 1870: este critério adicional na cláusula **WHERE** verifica outra condição: **manhattan_distance_varray(cfv_in.fv, mfv.fv)** calcula a distância Manhattan entre o valor do campo **fv** na linha atual da tabela itest_varray e o valor do campo **fv** na linha atual da tabela min_itest. **<= 1870** verifica se a distância Manhattan calculada é menor ou igual a 1870.

Portanto, a consulta está selecionando as linhas que atendem simultaneamente às duas condições: uma relacionada à distância entre o campo fv da tabela **cfv_in** (tabela itest_varray) e um valor específico calculado a partir da tabela itest_varray onde o id é 2, e outra relacionada à distância entre o campo fv da tabela **cfv_in** e o campo fv da tabela **mfv** (tabela min_itest). Em outras palavras, ela está buscando registros que estejam dentro de determinadas distâncias de ambos os valores de referência.

Essas consultas demonstram a viabilidade de implementar desde consultas mais básicas até consultas mais complexas, utilizando o operador de abrangência por meio do SQL padrão, sem necessidade de extensões na linguagem ou de bibliotecas adicionais. Isso é particularmente útil para aplicações que requerem busca por similaridade, onde a identificação de objetos semelhantes com base em suas características é um elemento fundamental.

6. Ambiente de desenvolvimento dos testes

Para a execução e testes das consultas, foi utilizado um computador *Desktop* com a seguinte configuração:

- Processador Intel Core I5 9400F 2.9ghz
- 16 GB de RAM DDR4 3200mhz
- SSD 512GB XPG S41 TUF, M.2 NVMe
- Windows 10 Pro
- Sistema operacional 64 bits
- Oracle Database 21c Express Edition Release 21.0.0.0 [5]

7. Manipulação do plano de execução utilizando *hints*

O plano de execução é uma descrição detalhada e estruturada do processo pelo qual um banco de dados processa uma consulta SQL para obter e manipular os dados solicitados [Prado 2012]. É uma ferramenta crucial utilizada pelos sistemas de gerenciamento de banco de dados, como o Oracle, com o objetivo de otimizar e executar consultas de maneira eficiente.

Ao enviar uma consulta SQL para o banco de dados, o otimizador analisa a consulta, leva em conta as estatísticas das tabelas e índices envolvidos e, em seguida, cria um plano de execução para determinar a melhor estratégia de acesso aos dados. O plano de execução é, basicamente, uma lista de etapas e operações que o banco de dados seguirá para executar a consulta e produzir os resultados desejados [Prado 2012]. Para realizar a manipulação do plano de execução da *query* complexa, foi utilizado o *hint* NO_PUSH _SUBQ. A Figura 14 mostra a implementação do *hint* no código da consulta.

[5] Oracle Database 21c Express Edition Release 21.0.0.0 é uma versão gratuita e de entrada da Oracle Database, projetada para desenvolvedores, estudantes e pequenas aplicações.

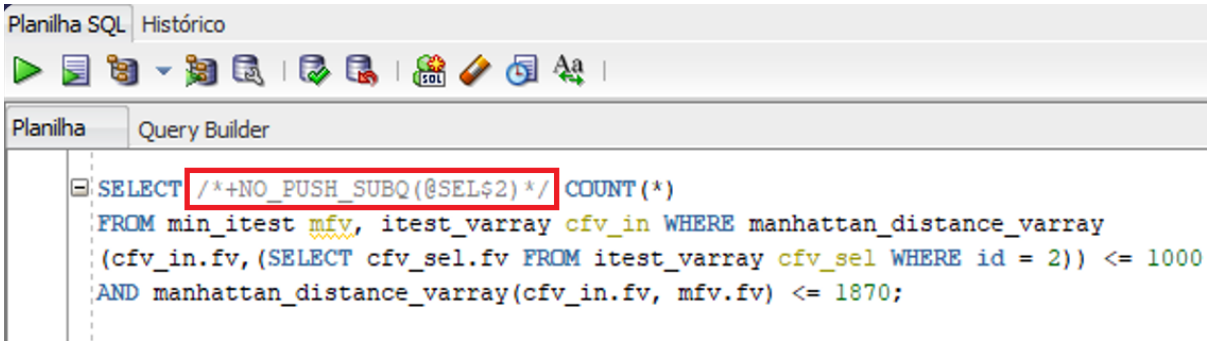

Figura 14. Consulta com a dica NO_PUSH_SUBQ.
Fonte: Própria.

Segundo a documentação da Oracle, o *hint* NO_PUSH _SUBQ informa ao otimizador que ele não deve mesclar a subconsulta com a consulta principal e, em vez disso, avaliar as subconsultas não mescladas como a última etapa do plano de execução. Ele indica que a subconsulta deve ser executada separadamente, e seus resultados usados na consulta principal. A utilização desse *hint* pode ser útil em casos onde a subconsulta é mais eficiente quando executada separadamente. No entanto, é importante considerar o impacto no desempenho da consulta e testar em diferentes cenários antes de usar esse hint [Oracle 2023] .

Para este estudo, a utilização do *hint* escolhido será unicamente para gerar um plano de execução diferente daquele escolhido pelo otimizador. A otimização das consultas será realizada pela ferramenta CBO (*Cost-Based Optimizer*), um otimizador baseado em custo já presente no banco de dados Oracle. Na seção final do artigo, será feita uma análise entre o plano de execução da consulta otimizada pelo CBO, comparada à consulta manipulada pelo *Hint* que gerou o plano de execução alternativo. Dessa forma, iremos visualizar quais métodos de acesso o otimizador utilizou e o que o levou a utilizar tais métodos, demonstrando assim que o otimizador da Oracle enxerga diferentes planos de execução e sempre irá escolher o de menor custo.

8. Verificação dos planos de execução e análise dos resultados

Como última etapa do desenvolvimento, o plano de execução do *Range query* complexo foi extraído e analisado para confirmar que a alteração desejada ocorreu conforme planejado. Para extrair o plano de execução da consulta, foi utilizado o comando SQL *Explain Plan For*. A Figura 15 a seguir demonstra sua utilização. O comando é executado no início de cada consulta e, após o compilamento, o plano de execução é "explicado", sendo possível acessá-lo através da procedure display da *package* dbms_xplan.

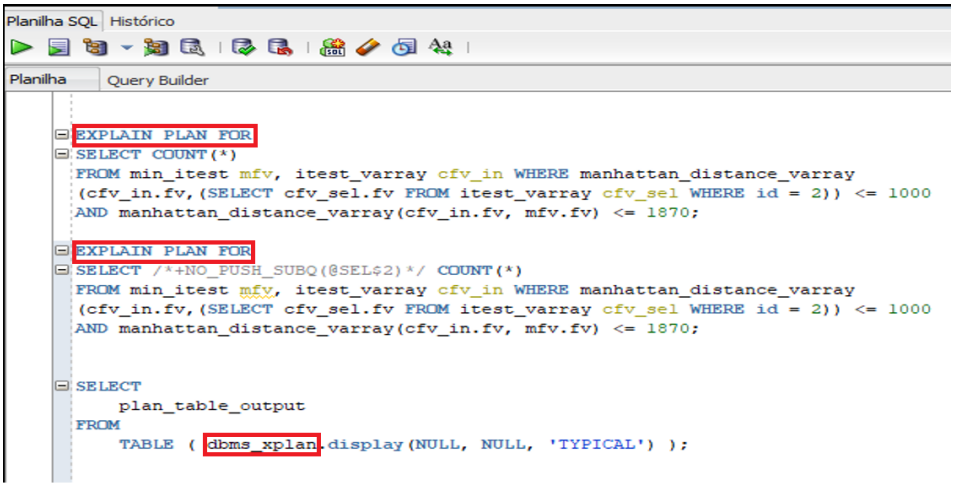

Figura 15. Extração do plano de execução.
Fonte: Própria.

A análise dos planos de execução permitiu visualizar o desempenho da consulta complexa e compreender o impacto dos *hints* da Oracle na manipulação do plano de execução. A comparação dos planos de execução das consultas 1 e 2 exibidos na Figura 15 mostrou que o uso do *hint* NO_PUSH _SUBQ na Consulta 2 levou o otimizador a escolher um plano de execução diferente, o que poderia resultar em um melhor desempenho da consulta, dependendo do contexto e das características dos dados envolvidos. Através do Plano de Explicação do SQL Developer, foi constatado que a *subquery* faz a seleção por similaridade. Os planos de execução obtidos são apresentados, respectivamente, nas Figuras 16 e 17 a seguir:

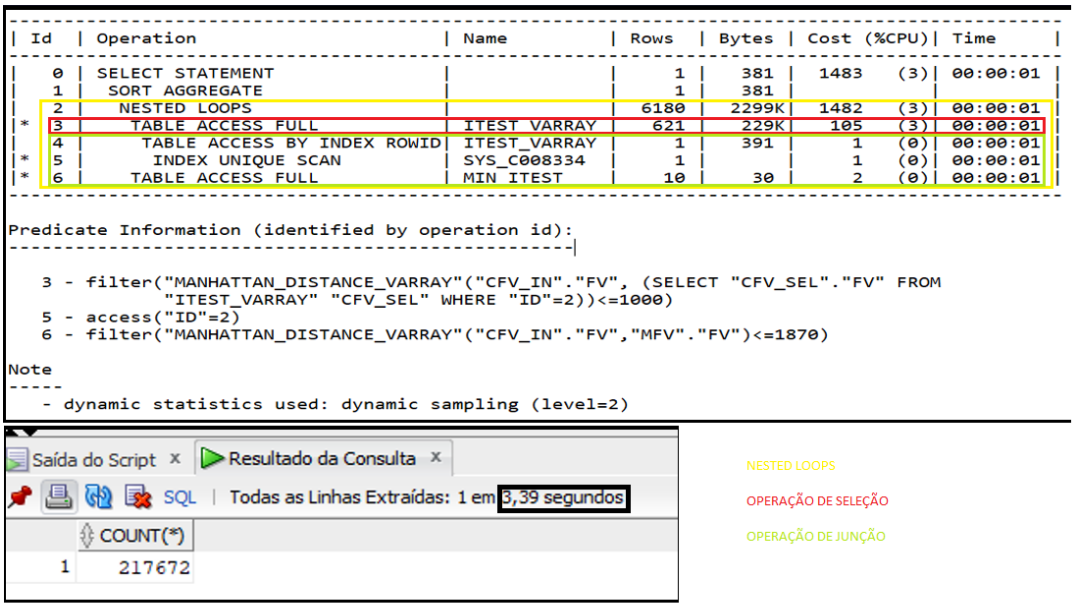

Figura 16. Consulta otimizada pelo CBO.
Fonte: Própria.

No plano de execução do Oracle Database, o "*Select Statement*", também conhecido como declaração SELECT, é um componente da estrutura que representa a operação principal da consulta SQL. É o ponto de partida da execução da consulta que descreve como os dados serão recuperados e ajustados para atender à solicitação da consulta. Para uma melhor compreensão do plano de execução, os métodos de acesso foram destacados.

O método de acesso *Nested Loops* (destaque amarelo) é uma técnica utilizada pelo otimizador para combinar duas tabelas utilizando *loops* aninhados. A ideia básica do *Nested Loops* é comparar cada linha da tabela externa com todas as linhas da tabela interna para encontrar as correspondências necessárias para a junção. Junções entre tabelas geralmente são mais custosas, e é por esse motivo que o otimizador utiliza métodos específicos, como o *Nested Loops*, para o processamento de *Joins*. Dentro do *Nested Loops*, teremos os acessos às tabelas da consulta através do método *TABLE ACCESS FULL*.

Podemos notar que a operação de seleção na tabela ITEST_VARRAY (Destaque vermelho) foi realizada primeiro, e por último foi realizada a operação de junção entre as tabelas ITEST_VARRAY e MIN_ITEST (Destaque verde), o que, do ponto de vista lógico, é o melhor plano de execução possível. Primeiro, é realizada uma operação menos custosa que filtra alguns elementos e, depois, uma operação mais custosa com menos elementos. Como resultado, temos o retorno de 217672 registros em 3,39 segundos. Um fato interessante que também podemos observar é que o otimizador indexou os registros através do ROWID. O ROWID é um identificador único da tabela, e é a maneira mais rápida de encontrar um registro.

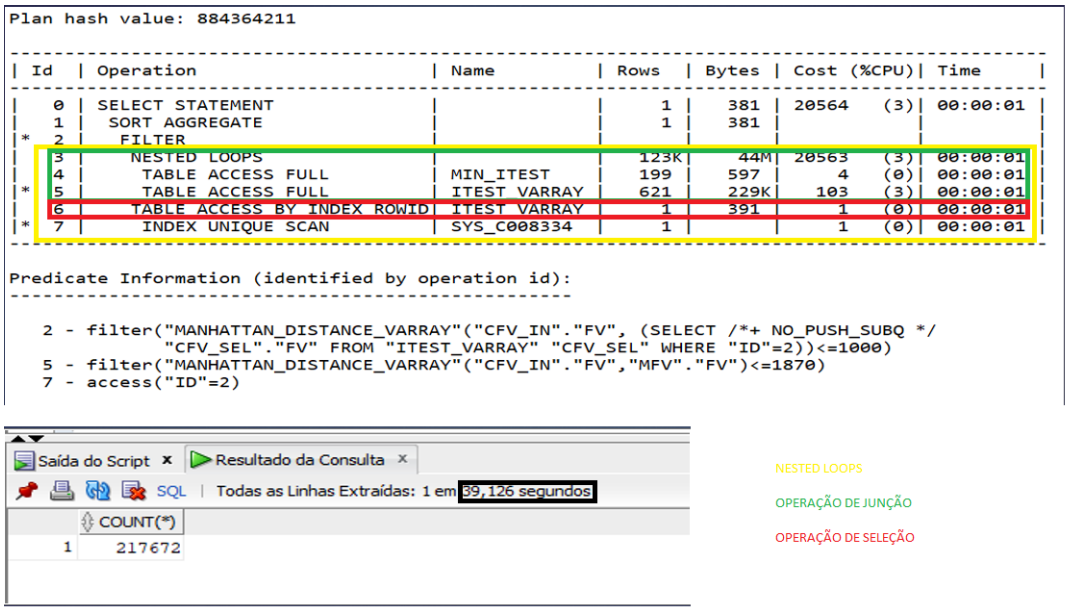

Figura 17. Consulta manipulada: Hint NO PUSH SUBQ.
Fonte: Própria.

Observamos no plano de execução da consulta manipulada que o *hint* utilizado gerou um plano de execução equivalente, apenas invertendo a ordem dos fatores. O *hint* fez com que a *subquery* não pudesse ser inserida no filtro dentro do *Nested Loops*, obrigando

o SGBD a realizar a operação de seleção por abrangência após a junção por abrangência. Isso resultou em uma execução que demandou muito mais recursos de processamento pelo otimizador, pois a operação de junção (mais custosa) foi realizada primeiro, filtrando grande parte dos elementos da consulta. A consulta otimizada pelo CBO foi executada em 3,96 segundos enquanto a consulta manipulada pelo hint foi executada em 39 segundos. Com isso, podemos concluir que o plano de execução escolhido pelo CBO é o mais otimizado para o cenário atual.

Neste sentido, fica demonstrado que muitas das operações necessárias para implementar um Banco de Dados por Similaridade já se encontram em Bancos de Dados Relacionais tradicionais. É possível utilizar operadores por similaridade, mesmo que esses operadores não estejam previstos na linguagem. O otimizador da Oracle consegue enxergar diferentes planos de execução para a consulta, escolhendo sempre o de menor custo.

9. Conclusão

Este artigo apresentou o desenvolvimento de consultas por similaridade utilizando SQL padrão em um banco de dados relacional Oracle, com foco em operadores de busca por similaridade. Podemos concluir que é possível, mesmo que não previsto na linguagem dos bancos de dados relacionais, desenvolver operadores por similaridade para solucionar cenários de análise de dados complexos. Podemos afirmar que o banco de dados Oracle, além de suportar esses operadores, é capaz de otimizá-los e também gerar diferentes planos de execução alternativos por meio da utilização de *hints*.

Mostrado que o Banco de Dados gera os diferentes Planos de Execução através de *hints*, temos que esses diferentes Planos de Execução são enxergados pelo otimizador do Oracle e que este decide o melhor Plano de Execução possível com base no custo. Conclui-se, então, que o banco de dados Oracle demonstra ser uma opção com boa robustez e flexibilidade, podendo ser utilizado até mesmo em cenários de dados complexos, não necessitando de nenhuma extensão para isso, utilizando apenas seus próprios recursos internos e realizando pequenas adaptações.

Ao longo deste projeto, foi adotado com sucesso a metodologia Scrum como abordagem principal para gerenciar o trabalho. A estrutura iterativa e colaborativa proporcionada pelo Scrum permitiu uma gestão mais eficiente das tarefas, garantindo uma maior flexibilidade na adaptação a mudanças. A organização dos requisitos no *product backlog* e a divisão do projeto em *sprints* ajudou a manter um ritmo constante de trabalho e a cumprir os prazos estabelecidos. No geral, a aplicação do Scrum foi fundamental para o sucesso deste projeto, permitindo entregar resultados consistentes e alinhados com a expectativa.

Para o desenvolvimento deste trabalho e para destacar as competências e habilidades adquiridas ao longo do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, foram aplicados conceitos associados às disciplinas de Banco de Dados, Engenharia de Software, Estruturas de Dados, Inteligência Artificial, Gestão de Projetos, Metodologia Ágil, Metodologia de Pesquisa Científica e Projeto de Sistemas, visando o planejamento e o desenvolvimento teórico-técnico. Outros conhecimentos necessários para o desenvolvimento do trabalho foram obtidos por meio de pesquisa bibliográfica e estudo técnico, utilizando a ferramenta apresentada.

Como proposta para trabalhos futuros, é possível apresentar um estudo sobre a viabilidade de implementar outros operadores por similaridade presentes na literatura utilizando o SQL padrão em outros SGBDs, como o PostgreSQL ou o SQL Server. Além disso, é possível analisar o comportamento do plano de execução desses SGBDs, explorar potenciais ferramentas de otimização e conduzir testes com conjuntos de dados mais extensos.

Referências

Adali, S., Bufi, C., and Sapino, M.-L. (2004). Ranked relations: Query languages and query processing methods for multimedia. *Multimedia Tools and Applications*, 24:197–214.

Barioni, M. C. N., dos Santos Kaster, D., Razente, H. L., Traina, A. J., and Júnior, C. T. (2011). Querying multimedia data by similarity in relational dbms. In *Advanced database query systems: techniques, applications and technologies*, pages 323–359. IGI Global.

Barioni, M. C. N., Razente, H. L., Traina Jr, C., and Traina, A. J. (2005). Querying complex objects by similarity in sql. In *SBBD*, volume 5, pages 130–144.

Chen, L., Gao, Y., Li, X., Jensen, C. S., and Chen, G. (2015). Efficient metric indexing for similarity search. In *2015 IEEE 31st International Conference on Data Engineering*, pages 591–602. IEEE.

Cuellar, J. D. C. (2021). *Diferenciação de medidas em Espaços Métricos*. PhD thesis, Universidade de São Paulo.

Joshi, A. and Haspel, N. (2020). A novel data instance reduction technique using linear feature reduction. *Journal of Artificial Intelligence and Systems*, 2(1):191–206.

Kaster, D. d. S. (2012). *Tratamento de condições especiais para busca por similaridade em bancos de dados complexos*. PhD thesis, Universidade de São Paulo.

Levitin, A. (2011). *Introduction to the Design and Analysis of Algorithms*. Pearson, 3 edition.

Liu, Y., Zhang, D., Lu, G., and Ma, W.-Y. (2007). A survey of content-based image retrieval with high-level semantics. *Pattern recognition*, 40(1):262–282.

Oliveira, W. D. d. (2010). *Operação de busca exata aos K-vizinhos mais próximos reversos em espaços métricos*. PhD thesis, Universidade de São Paulo.

Oracle (2023). *Oracle Database SQL Language Reference, 12c Release 2 (12.2)*. Oracle Corporation. Disponível em: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/sqlrf/Comments.htmlGUID-7DC64CF1-465E-4307-B5CF-62E0757FCECA.

Penzo, W. (2005). Rewriting rules to permeate complex similarity and fuzzy queries within a relational database system. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):255–270.

Pola, I. R. V. (2010). *Explorando conceitos da teoria de espaços métricos em consultas por similaridade sobre dados complexos*. PhD thesis, Universidade de São Paulo.

Prado, F. (2012). Tuning de sql em bancos de dados oracle - revista sql magazine 97. Disponível em: http://www.devmedia.com.br/tuning-de-sql-em-bancos-de-dadosoracle-revista-sql-magazine-97/23810.

Queiroz, S. S. F. and Pinto, K. L. N. (2014). Extração de características e reconhecimento de padrões e objetos. *VETOR-Revista de Ciências Exatas e Engenharias*, 24(2):2–13.

Schwaber, K. and Sutherland, J. (2020). Guia do scrum: Um guia definitivo para o scrum: As regras do jogo. Disponível em: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-PortugueseBR-3.0.pdf.

Silva, Y. N., Aly, A. M., Aref, W. G., and Larson, P.-A. (2010). Simdb: a similarity-aware database system. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1243–1246.

Silva, Y. N., Aref, W. G., Larson, P.-A., Pearson, S. S., and Ali, M. H. (2013). Similarity queries: their conceptual evaluation, transformations, and processing. *The VLDB Journal*, 22:395–420.

Siqueira, P. H., Oliveira, P. H., Bedo, M. V., and Kaster, D. S. (2018). Standard sql approaches for similarity searching. In *2018 XLIV Latin American Computer Conference (CLEI)*, pages 472–481. IEEE.

UCI Machine Learning Repository (2016). Breast cancer wisconsin data set. Disponível em: https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data.

Wang, S., Nassar, M., Atallah, M., and Malluhi, Q. (2013). Secure and private outsourcing of shape-based feature extraction. In *Information and Communications Security: 15th International Conference, ICICS 2013, Beijing, China, November 20-22, 2013. Proceedings 15*, pages 90–99. Springer.

Wolberg, William, M. O. S. N. and Street, W. (1995). Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5DW2B.

Documento Digitalizado Restrito

Artigo de TCC do aluno Leonardo Luca Ribeiro

Assunto: Artigo de TCC do aluno Leonardo Luca Ribeiro
Assinado por: Daiane Tomazeti
Tipo do Documento: Dissertação
Situação: Finalizado
Nível de Acesso: Restrito
Hipótese Legal: Informação Pessoal - dados pessoais e dados pessoais sensíveis (Art. 31 da Lei nº 12.527/2011)
Tipo do Conferência: Documento Digital

Documento assinado eletronicamente por:

- **Daiane Mastrangelo Tomazeti**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 26/09/2023 14:21:41.

Este documento foi armazenado no SUAP em 26/09/2023. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1447861
Código de Autenticação: bdcb0f4047