Universidade Estadual de Londrina

PEDRO HENRIQUE BRAGA SIQUEIRA

REPRESENTAÇÃO E EXECUÇÃO DE CONSULTAS POR SIMILARIDADE EM SQL PADRÃO

Londrina
2018

PEDRO HENRIQUE BRAGA SIQUEIRA

REPRESENTAÇÃO E EXECUÇÃO DE CONSULTAS POR SIMILARIDADE EM SQL PADRÃO

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Daniel dos Santos Kaster

Londrina
2018

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Siqueira, Pedro Henrique Braga.
Representação e execução de consultas por similaridade em SQL Padrão / Pedro Henrique Braga Siqueira. - Londrina, 2018.
89 f. : il.

Orientador: Daniel dos Santos Kaster.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2018.
Inclui bibliografia.

1. Banco de dados - Gerência - Tese. 2. SQL (Linguagem de programação de computador) - Tese. I. Kaster, Daniel dos Santos. II. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. III. Título.

PEDRO HENRIQUE BRAGA SIQUEIRA

REPRESENTAÇÃO E EXECUÇÃO DE CONSULTAS POR SIMILARIDADE EM SQL PADRÃO

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Daniel dos Santos Kaster
Universidade Estadual de Londrina

Prof[a]. Dr[a]. Maria Camila Nardini Barioni
Universidade Federal de Uberlândia

Prof. Dr. Evandro Baccarin
Universidade Estadual de Londrina

Londrina, 14 de setembro de 2018.

A Deus.

A todos aqueles que me apoiaram durante o desenvolvimento desta dissertação.

A todos que se beneficiaram e se beneficiarão deste trabalho.

AGRADECIMENTOS

Agradeço a Deus pela minha vida e por todas as oportunidades que tive até o momento e que terei no futuro. Aos meus familiares e a todos meus amigos que de alguma forma me apoiaram até aqui. A todos os meus professores, desde os que me ensinaram nos primeiros dias do jardim de infância até o fim deste mestrado. Em especial ao meu orientador, Prof. Kaster, que me apoiou e orientou por todo este período. Sem ele os resultados deste trabalho não teriam sido tão bons. À equipe do grupo CROSS pelo apoio, em especial ao Paulo H. de Oliveira e ao Marcos Bedo pela ajuda na escrita dos artigos. À CAPES e à Fundação Araucária pelo auxílio financeiro.

"E tudo quanto fizerdes, fazei-o de todo o coração, como ao Senhor, e não aos homens"
(Bíblia – Colossenses 3:23)

SIQUEIRA, P. H. B.. **Representação e Execução de Consultas por Similaridade em SQL Padrão**. 2018. 89f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2018.

RESUMO

Esta dissertação aborda o armazenamento e a recuperação de dados complexos em SGBDRs (***S**istemas de **G**erenciamento de **B**anco de **D**ados **R**elacionais*), que depende de funções de distância para a avaliação da dissimilaridade dos dados. Neste sentido, um conjunto de ferramentas que armazenam dados complexos e utilizam operadores por similaridade na recuperação dos mesmos foram desenvolvidas em trabalhos anteriores. Entretanto, tanto a análise empírica de estratégias para armazenamento de dados complexos quanto a definição de uma representação adequada para operadores por similaridade ainda são questões em aberto na literatura. Este trabalho busca preencher essas lacunas através da classificação, implementação e avaliação de abordagens existentes para armazenamento de dados complexos de acordo com quatro abordagens suportadas pela linguagem SQL (***S**tructured **Q**uery **L**anguage*) padrão, a saber *relacional*,*objeto-relacional*, *binário* e *semi-estruturado*. Além disso, também é discutida uma representação abrangente para operadores por similaridade para recuperação de dados complexos que é consistente com o padrão SQL. É apresentada uma representação para funções de distância, que permite ao processador de consultas do SGBDR detectar e executar operadores por similaridade físicos. Foram avaliados a recuperação, inserção e armazenamento de dados complexos em diferentes cenários, considerando tabelas de entrada de diferentes tamanhos e dados complexos de diferentes dimensionalidades. Nestes cenários foram consideradas também três funções de distância que apresentam diferentes custos computacionais. Os resultados experimentais indicam que *(i)* as estruturas relacionais e objeto-relacionais superam em eficiência os outros dois concorrentes na maioria dos cenários, enquanto *(ii)* a estratégia objeto-relacional além de apresentar bom desempenho permite o uso de uma representação mais limpa.

Palavras-chave: Consultas por similaridade. Representação de dados complexos. Sistemas de Gerenciamento de Banco de Dados. SQL

SIQUEIRA, P. H. B.. **Representation and Execution of Similarity Queries in Standard SQL**. 2018. 89p. Master's Thesis (Master in Science in Computer Science) – State University of Londrina, Londrina, 2018.

ABSTRACT

This dissertation addresses complex data storage and retrieval in RDBMS (***R**elational **D**ata **B**ase **S**ystems*), concerning distance functions for the assessment of data dissimilarity. In this direction, a set of tools which store complex data and use similarity operators were developped by previous works. However, both the empirical analysis of strategies for complex data storage and the definition of a suitable representation for similarity query operators are still open issues in literature. In this work, we aim at fulfilling those gaps through the classification, implementation, and evaluation of existing approaches for complex data storage according to four approaches supported by standard SQL (***S**tructured **Q**uery **L**anguage*), namely *relational*, *object-relational*, *binary* and *semi-structured*. Moreover, we also discuss a comprehensive representation for similarity operators for complex data retrieval that is consistent with the SQL standard. Accordingly, a distance function representation is presented, which enables the RDBMS query processor to detect and execute physical similarity operators. We evaluated the retrieval, insertion and storage of complex data considering a set of scenarios, in which the input tables are of different sizes and dimensionalities. Additionally, these scenarios consider three distance functions with different computational costs. The experimental results indicate that *(i)* the relational and object-relational structures outperform the other two competitors in the majority of scenarios, whereas *(ii)* the object-relational strategy enables the use of a cleaner representation.

Keywords: Similarity queries. Complex data representation. Database Management Systems. SQL

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SGBD	*Sistema de Gerenciamento de Banco de Dados*
SGBDS	*Sistema de Gerenciamento de Banco de Dados por Similaridade*
SGBDR	*Sistema de Gerenciamento de Banco de Dados Relacional*
UDT	*User Defined Type*
UDF	*User Defined Function*
CBIR	*Content-Based Image Retrieval*
OR	*Objeto Relacional*
OCCI	*Oracle C++ Call Interface*
SQL	*Structured Query Language*
MAM	*Métodos de Acesso Métricos*
MPEG	*Moving Picture Experts Group*
MPQF	*MPEG Query Format*
TIFF	*Tagged Image File Format*
DICOM	*Digital Imaging and Comunications in Medicine*
ROI	*Region of Interest*
SVD	*Singular Value Decomposition*
PCA	*Principal Component Analisys*
LDA	*Linear Discriminant Analysis*
KPCA	*Kernel Principal Component Analisys*
FD	*Função de Distância*
FV	*Vetor de Características*
DTD	*Data Type Definition*

SUMÁRIO

1 INTRODUÇÃO

Os SGBDRs (**S**istemas de **G**erenciamento de **B**anco de **D**ados **R**elacionais) possibilitam o armazenamento e recuperação de um grande volume de dados de forma precisa, por meio de relacionamentos entre tabelas. Estes sistemas utilizam por padrão a linguagem SQL (***S**tructured **Q**uery **L**anguage*) e são o tipo de SGBD (**S**istema de **G**erenciamento de **B**anco de **D**ados) mais utilizado atualmente. Normalmente eles suportam os tipos de dados tradicionais (números, datas e cadeias de caracteres) os quais possuem relação de ordem total e também são chamados dados escalares. Contudo a demanda pelo armazenamento e recuperação de **dados complexos** (como são chamados dados não escalares) vem crescendo. Isso ocorre pois com o aumento na capacidade do armazenamento dos sistemas de informação, a demanda pelo processamento de dados complexos — dados multimídia, fenômenos naturais, séries temporais, séries de proteínas, etc. — é crescente. Estes últimos normalmente não possuem relação de ordem total, o que cria a demanda por novas técnicas que devem ser aplicadas para possibilitar o processamento dos dados complexos (Figura 1).

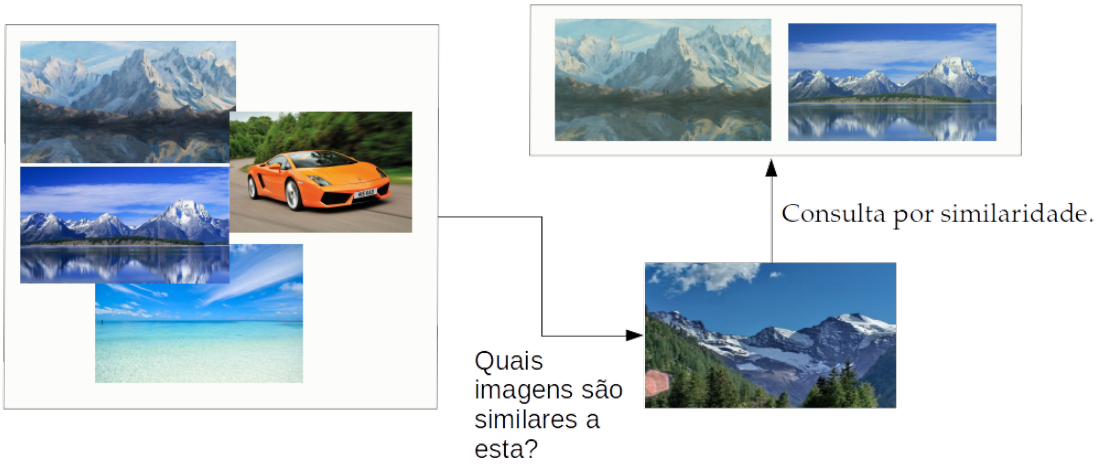

Figura 1 – Armazenamento e recuperação de dados complexos por similaridade. Ao inserir o conceito de similaridade em um SGBDR, é possível armazenar e recuperar dados complexos utilizando o mesmo. No exemplo dado, as imagens de montanhas armazenadas no SGBDR são recuperadas através de uma imagem de montanha, a qual é o centro da busca.

Diversas ferramentas foram desenvolvidas em trabalhos recentes [7, 2, 4, 8, 9, 10] no intuito de estender as funcionalidades dos SGBDRs, para que eles sejam capazes de armazenar e recuperar dados complexos, dando origem às ferramentas que este trabalho

chama de SGBDSs (**SGBDR**s por **S**imilaridade). Para representar e armazenar dados complexos, algumas destas ferramentas utilizam tipos de dados previstos na linguagem SQL padrão e outras a estendem para representar tipos de armazenamentos específicos para os dados complexos. Adicionalmente, não é possível aplicar os operadores de precedência entre dois elementos ($>$, $<$, $\leq$ e $\geq$), pois normalmente os domínios de dados complexos não atendem a relação de ordem total. Decorrente disto, usualmente, compara-se os pares de dados complexos por meio da similaridade entre eles. O cálculo da similaridade é feito através de **vetores de características**, que podem ser vistos como o resultado do mapeamento do conteúdo de um elemento em um domínio em particular (conforme ilustrado na Figura 2). As consultas por similaridade retornam os elementos mais similares de acordo com um elemento de referência, dada a medida efetuada por meio de uma **função de distância**. Embora trabalhos anteriores tenham desenvolvido alguns SGBDSs, não há uma discussão profunda na literatura a respeito da melhor representação para o modelo de armazenamento e das consultas por similaridade.

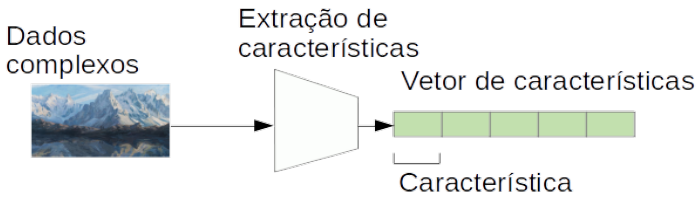

Figura 2 – Os dados complexos normalmente são sumarizados como vetores de características e os vetores de características são compostos por um conjunto de características. O processo que extrai as características de um dado complexo se chama **extração de características**.

Esta dissertação se insere neste contexto. Ela examina qual o uso mais adequado da linguagem SQL e modelo de dados básico suportado para se armazenar e recuperar dados complexos, dado qualquer domínio em que possam estar contidos. A abordagem adotada consiste em incluir extensões mínimas na SQL padrão e assim ser capaz de armazenar e recuperar dados complexos. A contribuição proposta é dividida em três pontos: *(i)* classificação dos vetores de características presentes na literatura, *(ii)* classificação e avaliação de um conjunto consistente de modelos de armazenamento de vetores de características em SGBDR.

1.1 Motivação

Trabalhos anteriores trazem diferentes propostas de inclusão de suporte a similaridade em SGBDRs. Estes trabalhos divergem na forma como armazenam vetores de características e como representam operadores por similaridade. Contudo, nenhuma destas propostas faz uma análise aprofundada do modelo mais adequado.

No que tange o armazenamento de vetores de características, os trabalhos existentes dividem-se com relação ao armazenamento de dados complexos em três categorias: *(i)* os que propõem um tipo de armazenamento próprio, *(ii)* os que estendem o catálogo de um SGBDR e *(iii)* os que representam os dados por meio de *User-Defined Types* (UDTs). Na primeira categoria, o `MESSIF` [7] armazena dados complexos como objetos semi-estruturados chamados *buckets*, os quais representam partições no espaço de busca. Na segunda categoria, o `SIREN` [2] consiste em um *middleware* que opera em uma camada acima de um SGBDR comercial. Na terceira categoria, estão as soluções `PostgreSQL-IE` [4], `FMI-SiR` [8], `SimDB` [9] e `MSQL` [10]. As duas primeiras armazenam dados complexos e vetores de características como atributos binários. No caso do `FMI-SiR`, os vetores de características são inseridos/atualizados através de funções externas ao SGBDR. Nas duas últimas soluções, cada característica extraída é armazenada em um atributo de uma tabela relacional que contém a representação do dado complexo. Entretanto, não há na literatura comparativo entre as diferentes abordagens para verificar qual é mais adequada ou eficiente.

Com relação à representação de operadores por similaridade, as soluções da literatura divergem também na forma de representá-los. Em geral, existem duas vertentes: *(i)* a que propõe extensões à linguagem SQL e *(ii)* a que utiliza recursos da SQL padrão. Por exemplo, as abordagens `SIREN`, `MESSIF` e `SimDB` adotam a primeira vertente. Tanto o `SIREN` quanto o `MESSIF` processam os predicados por similaridade separadamente e repassam os predicados que envolvem apenas dados tradicionais para o SGBDR, que se encontra em uma camada abaixo. Entretanto, devido a essa estratégia, o SGBDR não é capaz de reescrever os planos de execução das consultas, o que prejudica possíveis otimizações. O `SimDB` modifica diretamente o código-fonte do PostgreSQL para tentar cobrir essa lacuna, mas a solução não é extensível para outros SGBDRs nem cobre os critérios de consultas por similaridade de forma genérica. Já as abordagens `PostgreSQL-IE`, `FMI-SiR` e `MSQL` utilizam *User-Defined Functions* (UDFs) na representação dos operadores por similaridade através da SQL padrão. No caso do `PostgreSQL-IE`, o operador por similaridade é expresso na cláusula `FROM`, o que limita seu uso em representações de consultas complexas e não oferece suporte a otimização. O `FMI-SiR` implementa funções de distância usando o recurso *Oracle Data Cartridge*, que atua como uma interface para utilizar os métodos da biblioteca C++ externa Arboretum[1]. Adicionalmente, o `FMI-SiR` possibilita indexar os dados complexos por meio do comando `CREATE OPERATOR` do Oracle, além de permitir mesclar operadores por similaridade com operadores convencionais e manipular o plano de execução. A abordagem `MSQL` emprega uma estratégia parecida ao representar as funções de distância por meio de UDFs, que manipulam os vetores de características diretamente em colunas de tabelas.

[1] `https://bitbucket.org/gbdi/arboretum`

Este trabalho visa aprofundar as opções e limitações de inclusão de similaridade em SGBDRs usando, sempre que possível, recursos da SQL padrão. Isto porque a SQL padrão oferece vantagens inatas em comparação às extensões, uma vez que é um padrão amplamente conhecido e utilizado pelos usuários. Além disso, os SGBDRs comerciais dão suporte à SQL sem a necessidade de quaisquer alterações no código-fonte das ferramentas, o que possibilita aproveitar o aprendizado obtido no desenvolvimento de recursos especializados para recuperação de dados tradicionais. Por fim, o uso de SGBDRs e da SQL padrão garante a integridade de transações, o que pode ser diretamente estendido aos dados complexos. Em resumo, há muitas vantagens em se utilizar o SQL padrão, como exemplificado a seguir.

- Já é um padrão amplamente aceito, utilizado e conhecido por diversos usuários.
  - Desenvolvedores não precisam aprender uma nova linguagem.
  - Evita a criação de diferentes linguagens pelas respectivas ferramentas.
  - Não é necessária a criação de um comitê, ou gastar energia em discussões sobre um novo padrão.
- É possível implementá-lo em plataformas comerciais sem nenhuma ou pouca alteração no código fonte das mesmas.
  - Possibilita a integração das buscas por similaridade junto a funcionalidades disponíveis nestes sistemas.
  - Faz uso de novas versões dos SGBDRs usados como base sem a necessidade de manutenção no sistema de busca por similaridade.
  - Sua implementação não implica em modificação dos sistemas legados que usam as respectivas plataformas.
  - Garante-se a integridade das transações referentes a operações que envolvem dados complexos [10].
- Aproveita-se o aprendizado obtido no desenvolvimento de recursos para suportar outros tipos de dados, como é o caso de dados espaciais.

Não obstante, as abordagens que empregam a SQL padrão: *(i)* consideram apenas tipos específicos de vetores de características e/ou operadores de consulta por similaridade; e *(ii)* não o fazem de forma transparente o suficiente para que seja possível ao processador de consultas identificar os operadores por similaridade físicos a serem executados, independentemente da forma de armazenamento do vetor de características. A proposta deste trabalho é abordar de forma abrangente como é possível oferecer, utilizando a SQL padrão, os recursos necessários para a execução de uma grande variedade

de consultas por similaridade, considerando diferentes tipos e combinações de vetores de características.

Este trabalho oferece formas de se representar um conjunto consistente de operadores por similaridade e analisa diferentes modelos de armazenamento de vetores de características existentes na SQL padrão. Para analisar os modelos de armazenamento é apresentado um conjuto substancial de experimentos que medem o espaço ocupado em disco pelos vetores de características armazenados, a inserção e a recuperação de vetores de características, sendo que a recuperação de vetores de características é avaliada através do desempenho do operador de busca por abrangência.

1.2 Objetivos e contribuições

Este trabalho preenche um conjunto de *gaps* existentes na literatura ao definir: *(i)* o melhor de um conjunto consistente de modelos de armazenamento de vetores de características possíveis em SGBDRs e *(ii)* uma linguagem que estende minimamente a SQL padrão de forma a permitir que o processador de consultas interprete o operador físico representado pelo usuário. As contribuições presentes do trabalho são sumarizadas conforme segue.

1. **Representação SQL para os dados complexos.** Categorização dos tipos de vetores de características existentes e das funções de distância, bem como uma abordagem para representar operadores de busca por similaridade e indexá-los no contexto de instruções SQL de alto nível, com pequenas modificações na linguagem.

2. **Avaliação empírica de um conjunto de modelos de armazenamento.** Implementação e avaliação de um conjunto de modelos de armazenamento representativos em um *framework* estendido baseado na SQL padrão. As avaliações são realizadas através do emprego do operador Rq incorporado em consultas SQL de alto nível, indicando os cenários em que cada modelo de armazenamento de dados específico pode ser adequado.

1.3 Organização do trabalho

O restante deste documento é dividido conforme segue. O Capítulo 2 traz os fundamentos da recuperação de dados por similaridade tendo como base trabalho relevantes desta área. O Capítulo 3 apresenta como o armazenamento e a recuperação de dados complexos são estruturados em SGBDSs desenvolvidos por trabalhos anteriores. O Capítulo 4 apresenta a proposta de categorização de vetores de características, representação de funções de distância e de operadores por similaridade e diferentes estratégias de representação e armazenamento de dados complexos, fazendo uso da SQL padrão. O Capítulo 5 descreve

a implementação realizada da proposta e das estratégias consideradas em um módulo acoplado ao SGBD Oracle, o *e*`FMI-SiR`, e uma avaliação experimental das estratégias. Por fim, o Capítulo 6 traz as conclusões da dissertação.

2 RECUPERAÇÃO DE DADOS COMPLEXOS POR SIMILARIDADE

A quantidade e a diversidade de informações armazenadas digitalmente vêm crescendo exponencialmente nos últimos anos. Essas informações são representadas em diversos formatos e tipos de dados, o que requer que as técnicas computacionais de armazenamento e recuperação de dados se tornem ainda mais flexíveis e eficientes. Os SGBDRs (***S**istema de **G**erenciamento de **B**ancos de **D**ados **R**elacionais*) comerciais existentes são capazes de tratar com eficiência os chamados dados tradicionais (como números ou pequenas cadeias de caracteres). Entretanto, outras categorias de dados requerem outras técnicas, usualmente muito distintas das empregadas por SGBDRs, para serem manipuladas.

Imagens, áudios, vídeos e séries temporais são alguns exemplos dessa crescente variedade de tipos de dados armazenados. A esses tipos de dados, não é possível aplicar operadores de comparação baseados na Relação de Ordem Total ($<$, $\leq$, $=$, $\neq$, $>$, $\geq$) com semântica adequada tal como no caso dos dados tradicionais, o que inicialmente inviabiliza seu tratamento em SGBDRs comerciais [11]. Uma alternativa semanticamente viável é comparar esses tipos de dados por meio do cálculo da distância entre eles seguindo a Teoria de Espaços Métricos [12], considerando o grau de suas respectivas dissimilaridades. Os dados comparados por similaridade são chamados de **dados complexos** neste texto.

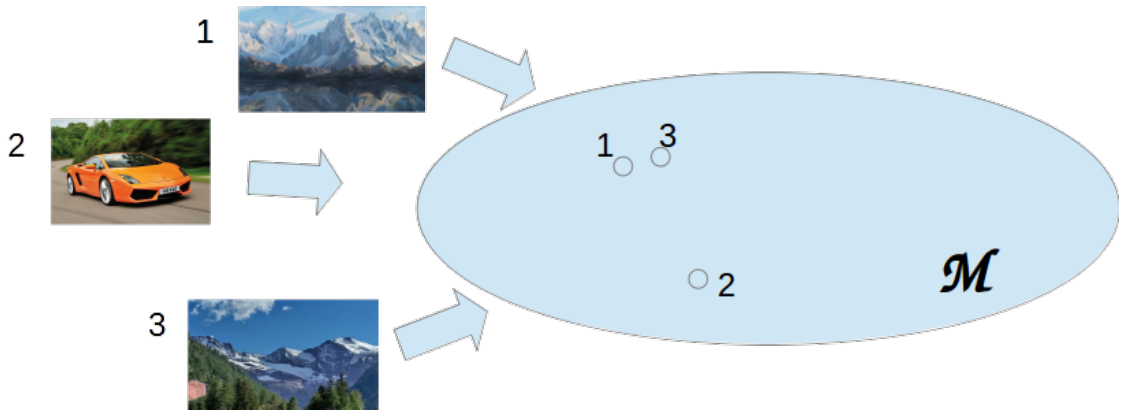

Figura 3 – Inserção de dados complexos em espaços métricos. Ao inserir dados complexos em um espaço métrico, idealmente os mesmos devem ficar tanto mais próximos no espaço quanto mais similares forem um do outro.

Para representar o conteúdo dos dados complexos, funções de extração de características particulares de cada domínio são comumente empregadas. Essas funções consistem

em mapear um dado complexo para um segundo domínio $\mathbb{S}$, cuja representação é uma estrutura mais simples do que o dado complexo original, a qual denominamos vetor de características[1]. Se uma função de distância δ for capaz de mensurar a distância entre dois objetos no domínio $\mathbb{S}$, $\delta : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}_+$, então dizemos que δ é capaz de medir a dissimilaridade entre dados complexos. O par composto pelo domínio dos vetores de características e pela função de distância caracteriza um Espaço Métrico $\mathcal{M} = \langle \mathbb{S}, \delta \rangle$, o que permite modelar o comportamento de diversos critérios de consultas por similaridade, tais como operações sobre conjuntos [13, 14] (Figura 3). SGBDRs podem empregar essa modelagem para representar operadores de consulta por similaridade [8], e o desempenho dessas buscas pode ser aprimorado por meio de otimizações no plano de execução e em algoritmos para a execução dos respectivos operadores em estruturas de indexação especializadas, conhecidas como Métodos de Acesso Métricos (MAMs) [12].

Ferramentas existentes para a manipulação de dados complexos em SGBDRs comerciais são, tipicamente, soluções híbridas que usam o sistema operacional para gerenciar os arquivos que contêm os dados e os vetores de características associados [5, 15, 14, 4, 8, 16, 10]. No contexto de uma solução geral para consultas por similaridade em SGBDRs, deve-se tratar o armazenamento de dados complexos e suas representações considerando as estruturas e os tipos de dados oferecidos pelo SGBDR. Além disso, é importante que tal solução não altere a forma como os operadores de busca da SQL padrão são tratados pelos SGBDRs, de forma que todo o processo de otimização de consultas não seja prejudicado pelo uso de implementações específicas de operadores por similaridade.

As seções a seguir introduzem os principais conceitos inerentes ao processo de armazenamento e recuperação de dados complexos por similaridade. São eles: vetores de características (Seção 2.1), funções de distância (Seção 2.2) e consultas por similaridade (Seção 2.3).

Os exemplos desta Seção e a maior parte dos exemplos desta dissertação utilizam imagens. Isto é feito em caráter ilustrativo, pois a contribuição deste trabalho engloba todos os domínios de dados complexos.

2.1 Vetores de Características

Vetores de características podem ser formados por um conjunto de medidas obtidas diretamente a partir de uma fonte de dados (Figura 2). Por exemplo, uma fonte de dados de amostras de solos pode ter amostras representadas por medidas que correspondem à concentração de um conjunto de nutrientes analisadas em laboratório. Em outros casos, a fonte de dados requer um processamento da informação para efetuar a extração de características.

[1] Apesar do termo *vetor*, não há restrição para que $\mathbb{S} = \mathbb{R}^n$.

O processo de obtenção dos vetores de características consiste na extração (Subseção 2.1.1), transformação e seleção das características (Subseção 2.1.2). Onde a extração do vetor de características é o processo de analise de um dado complexo. Nela é feita a extração de medidas e de outras informações que descrevem sob algum aspecto o conteúdo intrínseco do dado complexo. As transformação e seleção de características são processos que reduzem o conjunto de características a ser analisado com base no respectivo problema.

2.1.1 Extração de Características

O resultado da extração de características é o que se denomina vetor de caraterísticas (ou assinatura). Nesta seção será analisada a extração de características relevantes de dados complexos, adotando como exemplo a extração de características de imagens. Este processo envolve as seguintes fases: aquisição, seleção de regiões de interesse e limitação dos conjuntos de características utilizados.

Antes de se armazenar o vetor de características de uma imagem (ou um conjunto de imagens) em um SGBDS, se faz necessária a execução de diversos processos que devem ser feitos de forma criteriosa. Primeiramente é feita a aquisição da imagem, pela digitalização da mesma através de diferentes meios de captura. Após a aquisição, as imagens devem ser armazenadas em um formato específico. Por exemplo, na área médica as imagens costumam ser codificadas no padrão TIFF (***T**agged **I**mage **F**ile **F**ormat*) de 12 *bits* e encapsuladas em um arquivo no padrão DICOM (***D**igital **I**maging and **C**ommunications in **M**edicine*) [17, 18].

A fase de aquisição da imagem influencia todos os processos restantes, pois os objetos que são armazenados por meio de propriedades diferentes podem gerar falsos negativos e/ou falsos positivos no momento de sua recuperação [19]. Por exemplo, no caso das imagens médicas, as imagens podem ser obtidas de equipamentos diferentes e por este motivo, muitas vezes, se faz necessário um processamento posterior a digitalização das imagens para normalizar as propriedades das mesmas e a sua recuperação não seja prejudicada.

As imagens que se encontram em formato digital passam por uma fase de processamento que pode ser totalmente automatizado ou pode ser supervisionado[2] por um usuário especialista [20]. Nesta fase usualmente são selecionadas regiões de interesse da imagem, ROI (do inglês ***R**egions **O**f **I**nterest*). Para isto existem técnicas na literatura que consistem na aplicação de algoritmos de segmentação da imagem[3]. Após esta divisão, o

[2] Os **processos automatizados** executam algoritmos que não necessitam de *professor* para avaliar os resultados e os **processos supervisionados** precisam de um *professor*, um usuário especialista que avalia as respostas do algoritmo de acordo com as respectivas entradas.

[3] Técnicas de segmentação de imagem consistem em dividir uma imagem em múltiplas regiões, ou seja em conjuntos de pixels, que correspondem às áreas de interesse da aplicação.

cálculo das propriedades locais é realizado. Ao calcular as propriedades locais da imagem, é suficiente que o conjunto de características seja limitado, utilizando apenas aquelas que são do interesse do usuário [21] - processo que é aprofundado na Subseção 2.1.2. Por exemplo, no processamento de imagens, a maior parte dos algoritmos que fazem este tipo de processamento podem ser divididos em três grupos [22]: cor [23], forma [24] e textura [25].

Encontrar a melhor forma de se representar as propriedades das imagens na forma de um vetor de características não é fácil. Cada caso demanda que se encontre a melhor técnica de se fazê-lo, algumas vezes através de análises exaustivas e outras com apoio de algoritmos de inteligência artificial [26].

2.1.2 Transformação e Seleção de Características

Os processos de transformação e seleção de características têm como objetivo reduzir a dimensionalidade dos vetores de características visando a solução mais eficiente de um dado problema. Estes têm como objetivo reduzir a complexidade do tratamento dos vetores de características e reduzir o custo computacional do mesmo.

A dimensionalidade é uma das propriedades dos dados multidimensionais de maior importância devido à sua influência nos mesmos. Por um lado, caso a dimensionalidade seja muito baixa, as informações armazenadas podem ser inexpressivas. Por outro, a complexidade de se tratar um conjunto de dados complexos se eleva quanto maior for sua dimensionalidade. Adicionalmente, dados com muitas dimensões tendem a ter ruídos e também informações redundantes que geram processamento desnecessário. Em alguns casos a complexidade é tão grande que é chamada de **maldição da (alta) dimensionalidade**. A maldição da dimensionalidade é tratada através do conceito de **dimensão de imersão** e **dimensão intrínseca**, além de técnicas de **redução de dimensionalidade** [27].

A dimensão de imersão é a dimensionalidade do espaço onde os dados complexos do objeto em questão estão inseridos. Já a dimensão intrínseca é a dimensão dos dados independentemente da dimensionalidade do espaço no qual estão inseridos. Por exemplo, uma esfera que possui três dimensões (dimensão intrínseca) pode ter as suas informações armazenadas em um domínio de cinco dimensões (dimensão de imersão). A dimensão intrínseca é teoricamente o número mínimo de dimensões necessárias para representar o dado complexo sem perda de informação. É possível encontrá-la através da análise de correlação por meio do uso da teoria dos fractais [27].

A **redução da dimensionalidade** é imprescindível em alguns casos devido à maldição da dimensionalidade. Ela consiste em métodos de transformação e de seleção de características, os quais devem ser escolhidos de acordo com o domínio utilizado na aplicação.

A **transformação de características** consiste em transformar características em novas características ou fazer combinações de características e os algoritmos de **seleção de características** escolhem um subconjunto das características originais. Normalmente a seleção de características reduz a necessidade de prospecção de dados e os mesmos mantêm os significados físicos originais, enquanto que os algoritmos de transformação podem prover dados que têm uma melhor discriminação, embora normalmente não possuam uma correlação clara com as características originais.

Existem muitas técnicas de **transformação de características**. Abaixo serão apresentadas algumas das mais populares: SVD (***S**ingular **V**alue **D**ecomposition*), PCA (***P**rincipal **C**omponent **A**nalisys*) e LDA (***L**inear **D**iscriminant **A**nalysis*).

A técnica SVD fatora a matriz que representa o vetor de característica. Dada uma matriz que representa o vetor de características A de dimensões $m \times n$, fatoriza-se o mesmo através da fómula $A = U \sum V^T$, sendo U uma matriz unitária $m \times m$, $\sum$ é uma matriz diagonal mxn e V^T uma matriz unitária $n \times n$, em um processo chamado decomposição em valores singulares de A. As matrizes U, V e $\sum$ podem ser vista como *conceitos* escondidos em A. Sendo que os valores de $\sum$ são chamados de valores singulares. Depois desta fatoração, os menores valores singulares podem ser excluídos, reduzindo-se o número de dimensões da matriz que representa o respectivo vetor de características [28].

As técnicas de transformação de características mais utilizados são a PCA e a LDA, ambos similares à SVD. A PCA [29] é bastante utilizada porque é possível calculá-la com algoritmos de complexidade polinomial. Nela a maior parte das informações são condensadas em um número menor de dimensões, nas quais a variação é mais elevada. Este método é flexível [30], sendo que vem sendo usado em diferentes pesquisas de aplicações como técnicas de identificação de Alzheimer [31] e na avaliação da contaminação de metais pesados em superfícies [32].

A LDA é uma generalização do discriminante linear de Fisher (no Inglês *Fischer's linear discriminant*) e consiste em encontrar a combinação linear das características para separá-las em classes. Ela vem sendo usado em diferentes aplicações. Exemplos são sistemas de processamento de imagens que fazem reconhecimento facial [33, 34].

A principal diferença entre a PCA e a LDA é que a segunda utiliza rótulos de classes, o que significa que é um método supervisionado, enquanto que a PCA é um método não supervisionado. Os métodos descritos até aqui são métodos aplicáveis em domínios lineares, contudo existem aplicações que devem resolver o problema em um domínio não-linear. É possível, nestes casos, utilizar funções *kernel*, que transformam modelos não-lineares em modelos lineares mapeando os dados em um domínio de maior dimensionalidade onde os mesmos possuem um padrão linear. Assim é possível utilizar, por exemplo, o método PCA em um domínio não linear com o uso de função *kernel* – método chamado de KPCA (***K**ernel **P**rincipal **C**omponent **A**nalisys*), o qual é usado em

diferentes aplicações, por exemplo, em sistemas de reconhecimento de face [35].

Os métodos de **seleção de características** podem ser divididos em três famílias diferentes: filtro, encapsulamento (*wrapper*) e métodos embarcados [36]. Nos métodos da primeira família, a seleção é feita através das características intrínsecas dos dados, sem a aplicação de um algoritmo de aprendizado de máquina. Já na segunda família, os dados são "encapsulados" em torno de um método de aprendizado de máquina. O *wrapping* testa os subconjuntos possíveis de dados para que o algoritmo de aprendizado teste todos eles de acordo com um classificador e escolhe o melhor. Por fim, nos métodos embarcados o algoritmo aprende quais as características são mais relevantes enquanto o modelo é criado.

Embora as famílias de métodos *wrapper* e de métodos embarcados possam ser mais precisos em relação aos filtros, devido ao fato de usarem classificadores, elas normalmente não servem para outras classificações. Em outras palavras, os filtros são soluções mais genéricas embora menos precisas. Outro problema dos métodos da família *wrapper* é que eles têm custos muitas vezes proibitivos para dados de alta dimensionalidade, pois utilizam métodos de aprendizado de máquina [36].

2.2 Funções de Distância

O cálculo de similaridade (ou dissimilaridade) entre vetores de características é usualmente realizado utilizando-se funções de distância. Elas retornam um valor real não negativo. Quanto menor este valor mais similares são os objetos analisados e o valor zero indica identidade.

Existem diversas funções de distância catalogadas na literatura [37], definidas para espaços multidimensionais e/ou para dados adimensionais (que não possuem dimensão definida). Algumas funções de distância são particularmente interessantes pois permitem a definição de espaços métricos.

Um espaço métrico é formalmente definido como $M = \langle \mathbb{S}, \delta \rangle$, tal que $\mathbb{S}$ é um domínio de dados e δ é uma função de distância, tal que $\delta : \mathbb{S} \times \mathbb{S} \mapsto \mathbb{R}_+$ que satisfaz as seguintes propriedades (**postulados do espaço métrico**): identidade ($\delta(s_1, s_2) = 0 \iff s_1 = s_2$), simetria ($\delta(s_1, s_2) = \delta(s_2, s_1)$), não negatividade ($0 < \delta(s_1, s_2) < \infty \iff s_1 \neq s_2$) e desigualdade triangular ($\delta(s_1, s_3) \leq \delta(s_1, s_2) + \delta(s_2, s_3)$). Exemplos de métricas são as funções da Família Minkowski (L_p) com $p \geq 1$. Estas funções de distância operam com dados numéricos dimensionais e incluem a distância Manhattan (L_1), a distância Euclidiana (L_2) e a distância Chebyshev (L_∞) [37]. Uma métrica que opera sobre cadeias de caracteres, que é um tipo de dado adimensional, é a distância Levenshtein, ou distância de edição (L_{edit})] [38].

Diferentes funções de distância têm diferentes propriedades. Por exemplo, as funções da Família Minkowski são mais suscetíveis a maldição da dimensionalidade com o

aumento do p, ou seja, a função Euclidiana é mais sensível ao aumento da dimensionalidade do que a função Manhattan - conforme demonstram as análises experimentais de [39]. Outro exemplo, a Mahalanobis é mais custosa e possui formato elíptico. Ela é mais adequada para encontrar agrupamentos do que as funções L_p [40].

Existem situações em que é interessante modificar o espaço métrico definido por uma função de distância através da **ponderação de características** [41]. Esta consiste em atribuir pesos a determinadas características para aumentar ou diminuir a sua importância no cálculo de similaridade. O problema da atribuição de pesos às características pode ser resolvido de forma trivial em um processo de tentativa e erro exaustivo. Ainda, conforme ilustrado em [41, 42], é possível que o SGBDS use a ponderação de características para ajustar o cálculo das distâncias de acordo com a percepção do usuário, que informa ao sistema quais respostas de uma consulta são relevantes e quais não são.

Além disso, em alguns casos, é necessário utilizar uma coleção de descritores para uma imagem. Por exemplo, de forma a simular melhor a percepção humana, a qual considera simultaneamente diversos aspectos de uma imagem, derivados dos padrões de cor, textura e forma [43]. A distância entre dois vetores de características neste caso é calculada por meio de uma função agregadora de distâncias, que agrega os valores de distância entre cada aspecto do dado complexo. Se a função agregadora de distâncias é uma métrica, ela é denominada uma métrica produto, pois opera sobre o produto cartesiano dos componentes dos vetores de caraterísticas compostos [44, 45, 46]. Um exemplo de métrica produto é a combinação linear entre as distâncias parciais: $\Delta(s_1, s_2) = \sum_i^n w_i * \delta_i(s_{1_i}, s_{2_i})$, onde w_i é o peso da distância δ_i entre os componentes s_{1_i}, s_{2_i} dos vetores de características compostos s_1 e s_2 (a classificação dos tipos de vetores de características é apresentada na Seção 4.1).

2.3 Consultas por Similaridade

O processamento de dados complexos requer abstrações não triviais sobre o seu conteúdo. Isso leva ao problema da representação das consultas por similaridade. Duas alternativas podem ser adotadas: a abordagem dos operadores baseada em conteúdo ou a abordagem dos operadores por similaridade. Ambas serão exemplificadas através das seguintes consultas: *(i)* retorne os exames radiológicos que apresentam massa de tamanho anormal e *(ii)* retorne as imagens que contenham o pôr do sol no mar [47].

A abordagem dos operadores baseados em conteúdo define operadores específicos para cada abstração em particular. Nela, para executar a consulta i o sistema precisa de duas funções específicas, a primeira que encontram padrões na radiografia e que identifiquem a massa e a segunda que calcula a área da massa e a consulta ii requer uma função que reconheça o sol e o mar em uma imagem e outra que reconhece a disposição

espacial entre os dois objetos (para reconhecer se cada um deles está no sul, norte, leste ou oeste). Nesta abordagem cada uma das funções deve ter um operador correspondente. Ela pode ser adequada para sistemas que focam em problemas pontuais, contudo o número de operadores pode ser excessivamente grande, pois no caso de dados multimídia seria necessário criar um ou mais operadores para cada domínio de imagens para desenvolver uma semântica suficientemente abrangente e o mesmo é verdade para outros domínios de dados complexos. Neste sentido, a execução de buscas que envolvam diversos operadores fica complexa, o que dificulta a sua escrita por parte do usuário. Além disso, é necessário muito mais tempo a ser gasto no aprendizado das diferentes linguagens, que serão específicas para cada domínio de aplicações.

Por outro lado, a abordagem dos operadores por similaridade define um pequeno grupo de operadores capazes de serem utilizados em qualquer domínio de dados complexos. Um conjunto consistente destes operadores é apresentado na Subseção 2.3.2. Tais operadores computam a similaridade entre um grupo de elementos dado na consulta e os que estão armazenados no banco de dados, para verificar quais satisfazem a uma determinada condição de busca. Através destes operadores é possível representar as consultas *i* e *ii*, sendo que a aplicação deve avaliar o extrator de características e a FD mais adequados para cada uma delas. Esta abordagem possui duas vantagens com relação à anterior. A primeira vantagem da abordagem dos operadores por similaridade é o fato de ser mais fácil fornecer um exemplo do que descrevê-lo. A segunda vantagem é que com um número restrito de operadores de consulta é possível representar pesquisas envolvendo um grande número de domínios de dados complexos. É necessário apenas o ajuste do cálculo de similaridade para o problema em questão.

Dadas as vantagens em se adotar operadores por similaridade (em detrimento de operadores baseados em conteúdo), esta estratégia é amplamente utilizada na literatura e será considerada no restante deste texto.

2.3.1 Propostas de Inclusão de Operadores por Similaridade na Álgebra Relacional

Existem três linhas principais que buscam incluir operadores por similaridade no modelo relacional. A primeira define a estrutura algébrica com o conceito de lógica *fuzzy*. Esta traz o conceito de imprecisão, seguida por diversos trabalhos [48, 49, 50, 51].

Outra linha estende o modelo relacional com o conceito de relações ordenadas (*ranked relations*), na qual as tuplas e os atributos são classificados de acordo com um critério de *ranking* definido pelo usuário. A idéia é ordenar as relações de entrada de forma que os operadores de consulta preservem esta ordem e retornem os *top-k* elementos que melhor satisfazem o critério estabelecido. Exemplos de trabalhos que seguem essa linha são [52, 53, 54].

A última delas é a utilizada neste trabalho. Ela não modifica os conceitos do modelo relacional, mas inclui novos operadores ao mesmo. Estes operadores podem ser divididos em duas categorias: *(i)* os que utilizam um limiar de dissimilaridade ξ para selecionar os resultados desejados e *(ii)* aqueles que retornam os elementos mais similares considerando-se o critério de similaridade fornecido, delimita-se o tamanho da resposta por um número de elementos k.

2.3.2 Operadores por Similaridade

Esta seção apresenta um conjunto de operadores por similaridade que estão entre os mais importantes apresentados até o momento na literatura.

2.3.2.1 Operadores de seleção e junção por similaridade

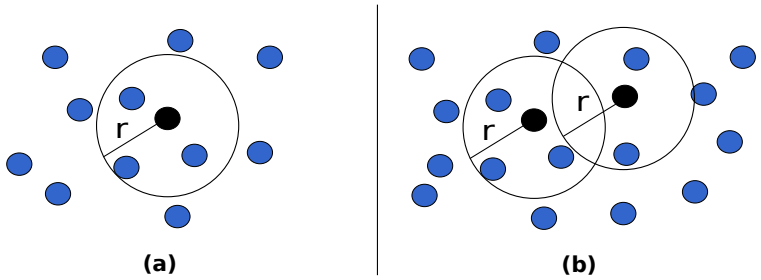

Figura 4 – (a) Seleção por abrangência e (b) junção por abrangência. Em ambos os exemplos, os centros de busca são representados pelas bolas pretas e o conjunto resposta é dado pelos elementos que se encontram a um raio r dos mesmos.

Existem vários tipos de operadores de consulta de similaridade, baseados em função de distância de um ou múltiplos centros. Os mais usados são os que utilizam funções de distância de um centro. Entre eles, os mais populares são: a **busca por abrangência** (*range query* - Rq), que recupera os elementos armazenados cuja dessemelhança de um elemento de consulta é menor ou igual a um determinado limite de um centro de busca (Figura 4(a)) e a **busca pelos k de vizinho mais próximos** (k-*Nearest Neighbor query* – k-NNq) (Figura 5(a)), que retorna os k elementos mais próximos ao elemento de consulta.

Os operadores básicos apresentados acima são empregados para gerar outros tipos de consultas por similaridade. Junções de similaridade são operações que mesclam dados complexos de dois conjuntos de dados de entrada de acordo com um determinado critério de similaridade. Exemplos delas são: a **junção por abrangência** (*range join* – $\bowtie_{\mathrm{Rq}}$ – Figura 4(b)), **junção dos k vizinhos mais próximos** (k - *Nearest Neighbor join* – $\bowtie_{k\mathrm{NN}}$ -0 Figura 5(b)) e **junção dos k pares mais próximos** (k-*Closest Pair join* – $\bowtie_{k\mathrm{CN}}$ – Figura 5(c)). Por fim, outro operador importante é o **k-NN reverso** (Rk-NNq), cujo

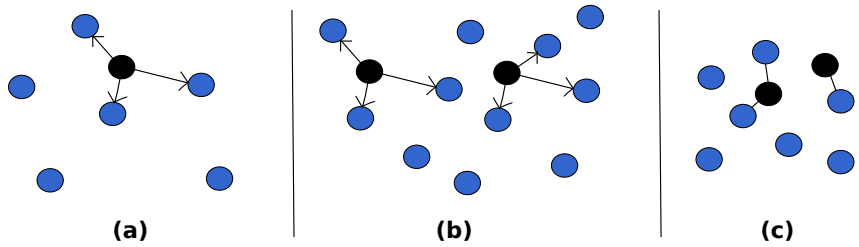

Figura 5 – (a) Seleção k-NN, (b) junção k-NN e (c) junção k-CN. Nos três exemplos, $k = 3$, os elementos da primeira relação de entrada são representados pelas bolas pretas e os da segunda relação de entrada pelas bolas cinza e as setas apontam para os elementos que compõem o conjunto resposta.

objetivo é obter todos os elementos armazenados que têm um dado elemento de referência como um dos k elementos mais similares [55, 56].

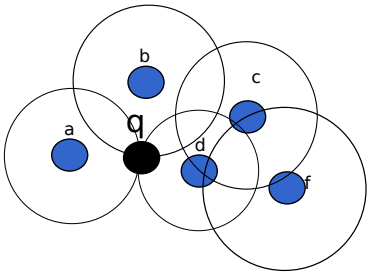

Figura 6 – Exemplo de seleção Rk-NNq, onde o centro de busca é o elemento q e os elementos retornados são os a, b e d, para $k = 1$.

Em particular, é conhecido que os operadores k-NNq e $\bowtie_{k\mathrm{NN}}$ não obedecem algumas regras de transformação de plano de execução válidas para os operadores tradicionais e para os operadores Rq e $\bowtie_{\mathrm{Rq}}$. Para aqueles dois operadores não são válidas as propriedades comutativas e distibutivas, além da maioria das propriedades de combinação/separação e associativas [57]. Este fato deve ser levado em consideração no desenvolvimento dos SGBDSs.

Os operadores citados têm sido utilizados em diversas aplicações: as buscas por abrangência e pelos k vizinhos mais próximos são aplicados em proteção da privacidade [58], aplicações médicas [59, 18], entre outros; o k-CN é utilizado no design de redes ópticas [60] e em visão computacional [61]; o Rk-NNq foi aplicado em soluções para a biologia molecular [62] e suporte a decisões [63]; os operadores ARq e k-ANNq são aplicados em buscas envolvendo dados georeferenciados [64].

A seguir são apresentadas definições formais dos principais operadores de consulta por similaridade, usando espaços métricos para representar operadores por similaridade.

Dado um espaço métrico $\langle \mathbb{S}, \delta \rangle$, onde $\mathbb{S}$ é um domínio de dados e δ uma função de

distância que satisfaz o postulado do espaço métrico, o operador de seleção por abrangência sobre um conjunto de dados $S \subseteq \mathbb{S}$ pode ser representado como $\mathrm{Rq}(s_q, \xi)$, sendo $s_q \in \mathbb{S}$ o elemento que representa o centro da consulta e ξ o raio que delimita sua abrangência considerando a distância δ, retornando todos os elementos $s_i \in S$ tais que $\delta(s_q, s_i) \leq \xi$. Para exemplificar o uso deste operador, um usuário de um sistemas de busca por imagens pode consultar as imagens mais próximas de uma foto dos Alpes através da sentença: *Encontre as imagens que se encontram a um limiar de dissimilaridade igual a 20 unidades desta imagem dos Alpes, fornecida como centro de consulta. Utilize a função Manhattan.*

A representação do operador k-NNq, considerando-se um espaço métrico $\langle \mathbb{S}, \delta \rangle$, pode ser dada por $k\text{-NNq}(s_q, k)$, onde $s_q \in \mathbb{S}$ é o elemento usado como referência e k o número de elementos mais próximos desta referência no conjunto de dados $S \subseteq \mathbb{S}$, que formam o conjunto solução da consulta. Este operador pode ser utilizado em uma rede social, onde um usuário pode requerer a resposta da sentença: *Retorne as cinco pessoas cujas fotos de capa são as mais parecidas com a foto da minha amiga Rebeca Abravanel. Utilize a função Mahalanobis.*

A junção por abrangência é um operador binário que requer dois conjuntos $R, S \in \mathbb{S}$, onde $\langle \mathbb{S}, \delta \rangle$ é o espaço métrico por ele definido, dado um limiar de abrangência ξ. O conjunto resposta da junção por abrangência $R \bowtie_{\mathrm{Rq}} S$ é composto pela concatenação dos pares dos elementos $\langle r, s \rangle$, $r \in R$ e $s \in S$, tais que $\delta(r, s) \leq \xi$. Por exemplo, o usuário de um sistema de imagens médicas pode pedir o resultado da sentença: *Faça a junção das imagens de estômago tiradas no dia de hoje com as imagens de estomago com úlcera. Retorne todas as imagens de hoje que se encontram a um raio igual a 30 unidades de alguma imagem de estômago com úlcera. Utilize a função Manhattan.*

O operador de junção dos k vizinhos mais próximos envolve dois conjuntos $R, S \in \mathbb{S}$, onde $\langle \mathbb{S}, \delta \rangle$ é um espaço métrico. Este operador pode ser denotado como $R \bowtie_{k\mathrm{NN}} S$ e retorna a concatenação dos k pares dos elementos $\langle r, s \rangle$, $r \in R$ e $s \in S$, tais que s é um dos k-vizinhos mais próximos de r. Para exemplificar, este operador pode ser utilizado em uma rede social para responder a seguinte sentença de um usuário: *retorne as três pessoas mais parecidas de cada um dos meus amigos, considerando um conjunto de propriedades dos perfis dos usuários. Utilize a função Euclidiana.*

Na junção dos k pares mais próximos, dado um espaço métrico $\langle \mathbb{S}, \delta \rangle$ e dois conjuntos $R, S \in \mathbb{S}$, a junção $R \bowtie_{k\mathrm{CN}} S$ tem como conjunto solução os k pares de elementos $\langle r, s \rangle$, $r \in R$ e $s \in S$, que são os mais próximos entre si dentre todos os pares formados por um elemento de cada conjunto. Um médico poderia utilizar este operador em um sistema ao solicitar, por exemplo: *faça uma junção das imagens de fígado dos pacientes do dia 27/11/2016 com as imagens de fígado com insuficiência hepática aguda e retorne os dez pares mais próximos entre si, utilizando a distância Manhattan.*

Por fim, o conjunto resposta do operador k-NN reverso é dado formalmente por [65]:

$RkNNq(q,k) = A = \{s_i \in S, q \in kNN(s_i,k) \wedge \forall s_j \in S - A : q \notin kNN(s_j,k)\}$. Este operador pode ser utilizado em um aplicativo policial onde o usuário requisita a resposta da sentença: *Retorne as fotos cadastradas que têm a foto do suspeito x como uma das duas fotos mais similares.*

2.3.2.2 Operadores de seleção por similaridade baseada em agregação

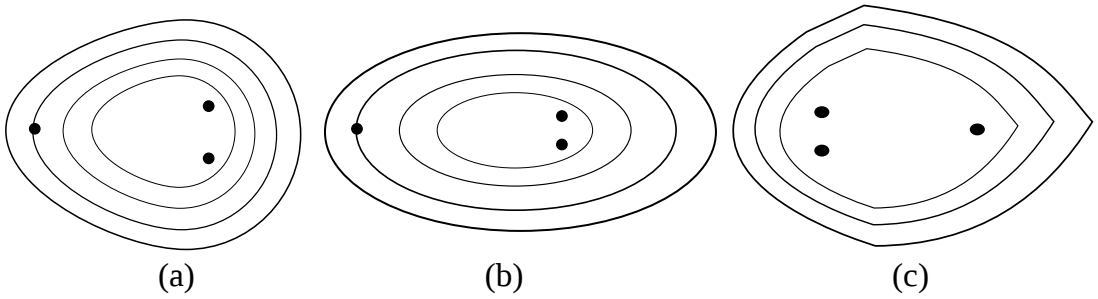

Figura 7 – Efeito do *grip factor* em um espaço Euclidiano bidimensional considerando um conjunto de centros Q=$\{q_1,q_2,q_3\}$, para (a) g=1, (b) g=2 e (c) g=∞ (Retirada de [1]).

Os operadores por similaridade agregada utilizam consultas por múltiplos centros. As funções de múltiplos centros são chamadas na literatura de funções de distância agregadas d_g, e são uma generalização das funções de distância de um centro. Dado um padrão de agregação (no inglês *grip factor*) g, d_g pode ser representada algebricamente como $d_g(s,Q) = \sqrt[g]{\sum_{s_q \in Q} \delta(s,s_q)^g * w_q}$, sendo δ uma função de distância de um centro.

A Figura 7 apresenta o efeito do *grip factor* em um espaço Euclidiano bidimensional composto pelos três centros do conjunto Q. Cada linha apresenta o espaço geométrico onde d_g tem o mesmo valor. As linhas representam um raio de abrangência distinto, o que permite a definição de consultas por abrangência e limitadas pelos k vizinhos mais próximos [1].

A função d_g é utilizada em dois operadores na literatura: busca por abrangência agregada (ARq) e busca pelos k-vizinhos mais próximos agregada (k-ANNq), exemplificada na Figura 8. Ambos os operadores permitem a criação de índices que são válidos quaisquer sejam o *grip factor* e o número de centros de busca [64]. Em [14] são destacadas três situações especiais para o operador k-ANNq: *(i)* Quando $g = 1$ minimiza-se a soma das distâncias e é chamada SUM, *(ii)* para $g = 2$ minimiza-se a média das distâncias para todos os centro e é chamada ALL e *(iii)* para $g = \infty$ miniza-se a distância máxima e é chamada MAX. Este operador pode ser utilizado por uma empresa de logistica que investiga a melhor cidade para instalar o seu armazem através da seguinte sentença: *retorne a cidade cuja soma das distâncias entre ela e (i) Belo Horizonte, (ii) São Paulo e (iii) Curitiba seja mínima. Utilize a distância Euclidiana no cálculo.*

As consultas por similaridade baseadas em agregação são uma extensão das consultas por abrangência e dos k vizinhos mais próximos. Diferentemente dos demais operadores, a consulta por similaridade agregada trabalha com um conjunto $Q \subseteq \mathbb{S}$ de elementos de busca, onde $\mathbb{S}$ é um domínio de dados. Pode-se agrupar este tipo de operador em três grupos diferentes: retorna os elementos mais próximos ou que estejam no raio de abrangência a/de algum elemento de consulta; retorna os elementos mais próximos ou que estejam no raio de abrangência a/de todos elementos de consulta; retorna os elementos mais próximos ou que estejam no raio de abrangência segundo uma função que agrega todos os elementos de consulta.

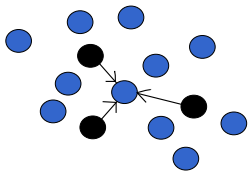

Figura 8 – Consulta k-ANNq com $k = 1$. Nesta consulta, o conjunto resposta é composto pelo elemento que se encontra mais próximo considerando-se mutuamente os três centros, representados pelas bolas pretas.

Existem propostas recentes que têm trabalhado em uma extensão a estas consultas, que são as consultas por similaridade agregada flexíveis, desenvolvidas por [66]. Dado um conjunto de entrada $S \subseteq \mathbb{S}$, é retornada a agregação das distância entre um de seus elementos $s \in S$ e um subconjunto do conjunto Q. Ela vem sendo útil para muitas aplicações e explorado em pesquisas atuais. Por exemplo, em buscas espaciais [67, 68].

2.3.2.3 Operadores de agrupamento por similaridade

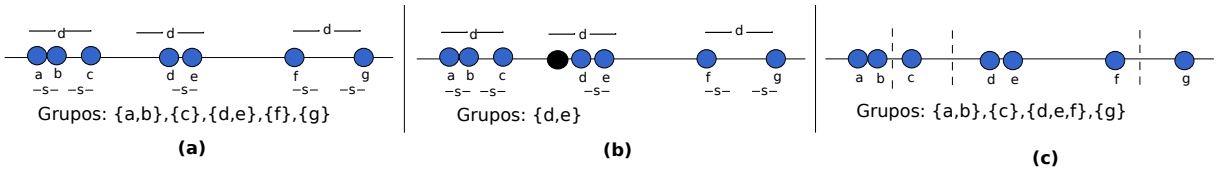

Figura 9 – Representações do operador de agrupamento por similaridade. Nesta ilustração, o diametro máximo é representado pelo simbolo d e a separação máxima de um elemento de seu grupo é representada pelo simbolo s. As sentenças dos agrupamentos são: (a) SGB-U – GROUP BY cophir MAXIMUM_ELEMENT_SEPARATION 3 MAXIMUM_GROUP_DIAMETER 6 (b) SGB-A – GROUP BY cophir AROUND {21} MAXIMUM_ELEMENT_SEPARATION 3 MAXIMUM_GROUP_DIAMETER 6 (c) SGB-D – GROUP BY cophir DELIMITED BY (SELECT Value FROM Thresholds) - os thresholds são representados pelas linhas tracejadas [9].

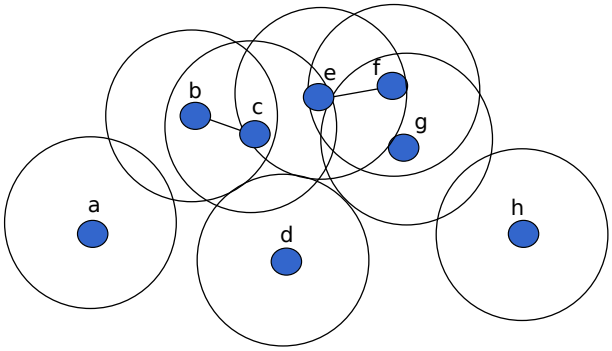

Figura 10 – O SGB-All agrupa os elementos que estão a um dado raio de todos os outros elementos do grupo. No exemplo apresentado o conjunto de entrada é dividido em seis grupos formados pelos elementos: (a), (b, c), (d), (e, f), (g) e (h). Neste tipo de agrupamento podem haver sobreposições (no inglês *overlaps*) entre os grupos, sendo que neste exemplo as sobreposições são resolvidas adicionando-se cada elemento em um grupo qualquer, através do *on-overlap join-any*.

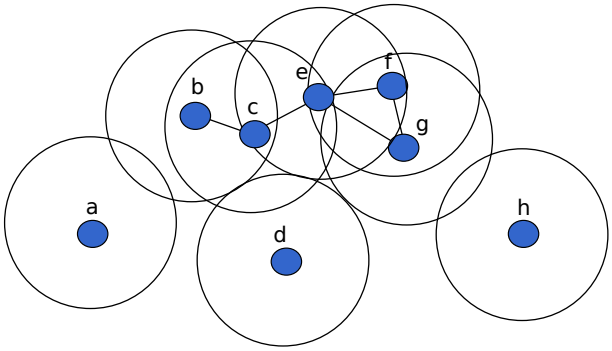

Figura 11 – O SGB-Any agrupa os elementos que estão a um dado raio de qualquer outro elemento do grupo. Neste exemplo os grupos formados são: (a), (d), (b, c, e, f, g) e (h).

Os operadores de agrupamento usados na cláusula GROUP BY da linguagem SQL são uma forma prática e fácil de se construir consultas com funções de agregação agrupadas de acordo com um critério indicado. Isso faz com que os mesmos sejam uma ferramenta poderosa nas mãos do usuário, tanto no que tange o processamento de dados tradicionais quanto no que tange dados complexos. Dois trabalhos relevantes propuseram um conjunto de operadores de agrupamento por similaridade, eles são apresentados a seguir.

O primeiro destes trabalhos definiu três operadores, que podem ser aplicados em dados unidimensionais [69]. O *Unsupervised Similarity Group-by* (SGB-U) agrupa os dados de acordo com a separação máxima entre dois elementos e o diâmetro máximo do grupo. Já no *Supervised Similarity Group Around* (SGB-A), o usuário define os pontos que serão os centros dos grupos e opcionalmente faz uso dos mesmos dois parâmetros que definem o

SGB-U. Finalmente, o *Supervised SGB using Delimiter* (SGB-D) faz o agrupamento com base em um conjunto de limiares (*thresholds*) fornecidos pelo usuário Figura 9.

O segundo trabalho definiu dois novos operadores que fazem agrupamento de dados multidimensionais: o *SGB-All* e o *SGB-Any* [70]. No SGB-All (Figura 10) os elementos são agrupados de acordo com a sua distância a todos os outros elementos do grupo. Todos os elementos de um grupo devem estar a um raio de todos os outros elementos deste grupo. O referido estudo apresenta uma semântica estendida que indica como deve ser a sobreposição dos diferentes grupos: *(i) on-overlap join-any* para que os objetos sejam adicionados a um grupo arbitrário, *(ii) on-overlap eliminate* em que os elementos que estão em uma sobreposição não são considerados e *(iii) on-overlap form-new-group* onde as sobreposições se transformam em novos grupos. No SGB-Any (Figura 11), caso um objeto esteja a uma distância ϵ de um dos elementos de algum grupo, ele entrará neste grupo. O único parâmetro necessário é o limiar de distância ϵ que um elemento precisa estar de outro para ser considerado do grupo.

2.4 Índices para Consultas por Similaridade

Buscas por similaridade normalmente envolvem um grande volume de dados e por conta disso o processamento deve ser feito de forma eficiente. A utilização de índices traz ganhos significativos para a execução de operadores por similaridade, conforme avaliado em [71, 72, 73] respectivamente para os índices Slim-tree e eD-index. Existem inúmeros trabalhos na literatura que propõem índices para consultas por similaridade. Um tipo de índice amplamente usado para aumentar a eficiência de consultas por similaridade é o MAM (**M**étodo de **A**cesso **M**étrico), cujo trabalho pioneiro foi a BK-tree [74]. Um MAM relaciona os elementos que nele são inseridos apenas por suas relações de similaridade, assim ele é capaz de indexar qualquer tipo de dado exigindo apenas a representação adequada da métrica. Usualmente, divide-se o conjunto de dados em subconjuntos e escolhe-se elementos que representam os respectivos subconjuntos. Quando um objeto é inserido no espaço métrico indexado pelo MAM, suas distâncias em relação aos representantes do respectivo subconjunto são calculadas e armazenadas.

Os MAMs são amplamente pesquisados e sua aplicação em diversos domínios de aplicações é promissora [75, 76, 77]. Sendo que existem três pontos que distinguem os diferentes MAMs: a forma como se escolhe os representantes dos subconjuntos, como os objetos são dispostos em relação aos representantes e se podem sofrer inserções ou remoções sem serem descaracterizados (se são estáticos ou dinâmicos).

Existem muitos trabalhos que têm como objetivo desenvolver MAMs para consultas por similaridade. A M-tree é um MAM dinâmico e suas extensões são provavelmente os MAMs mais difundidos [78, 79]. Ela é uma organização hierárquica de objetos s_i em

um conjunto $S \subseteq \mathbb{S}$ tal que $\langle \mathbb{S}, \delta \rangle$ é um espaço métrico. É uma árvore balanceada, cujos nós têm uma capacidade fixa e ocupam um espaço definido por um limiar e, além disso, ela é uma árvore dinâmica e paginada. Exemplos de índices que utilizam a M-tree como base são a Slim-tree [80], usada no SIREN, no SimbA e no `FMI-SiR`. A Slim-tree [81] é um índice dinâmico e o primeiro capaz de ajustar o próprio *fat-factor*[4] para otimizar o desempenho das buscas do índice. Por conta disso, o mesmo se ajusta após uma inserção ou remoção dos dados inseridos de forma mais adequada.

A M-tree foi contraposta pelo D-Index em [82], cujo objetivo é reduzir os custos de CPU e de E/S em relação aos custos apresentados pelos índices antecessores. Sendo o *eD-index* [73] uma versão do D-Index que tem o algoritmo de particionamento aprimorado e é usado no SimDB.

[4] O *fat-factor* é responsável por mensurar o grau de sobreposição dos nós do índice, é um parâmetro que afeta diretamente o desempenho do índice.

3 ABORDAGENS EXISTENTES DE SUPORTE A CONSULTAS POR SIMILARIDADE EM SGBDR

Na literatura existem diversos trabalhos que adicionam novas funcionalidades a SGBDRs comerciais para que estes passem a suportar o armazenamento e a recuperação de dados complexos. Este capítulo avalia as propostas desenvolvidas nos trabalhos anteriores de acordo com os seguintes pontos:

- modelo de armazenamento;
- representação dos operadores por similaridade;
- criação de estruturas de indexação;
- geração de planos de execução alternativos.

Este capítulo se aprofunda na análise do armazenamento e da recuperação de dados complexos de cada um dos seguintes SGBDSs: `SIREN` (Seção 3.1), `PostgreSQL-IE` (Seção 3.2), `MESSIF` (Seção 3.3), `SimDB` (Seção 3.4), `MSQL` (Seção 3.5) e `FMI-SiR` (Seção 3.6). Adicionalmente, a Seção 3.7 descreve como a XML se insere na recuperação e armazenamento de dados complexos. Por fim, a Seção 3.8 apresenta um sumário das abordagens existentes e contextualiza a proposta deste trabalho.

Os exemplos apresentados utilizam as seguintes tabelas:

```
--As tabelas armazenam dados do tipo genérico FEATURE_VECTOR que
--representa diferentes tipos ao longo do texto, a depender do modelo
--de armazenamento utilizado em cada caso.

--Tabela que armazena os vetores de características das imagens
--da base CoPhIR, menos aquelas que contêm dados de Londres.
CREATE TABLE cophir (
  id INTEGER PRIMARY KEY,
  image IMAGE_TYPE,
  fv FEATURE_VECTOR
);

--Tabela que armazena os vetores de características de Londres
--presentes na base CoPhIR
CREATE TABLE london (
  id INTEGER PRIMARY KEY,
```

```
    image IMAGE_TYPE,
    fv FEATURE_VECTOR
);
```

3.1 O Sistema SIREN

O SIREN [2] é um *midleware* que utiliza tipos de dados previstos no padrão SQL/MM e uma gramática SQL estendida para armazenar e recuperar dados complexos. A arquitetura deste sistema é ilustrada na Figura 12 e o funcionamento do mesmo é explorado nesta seção.

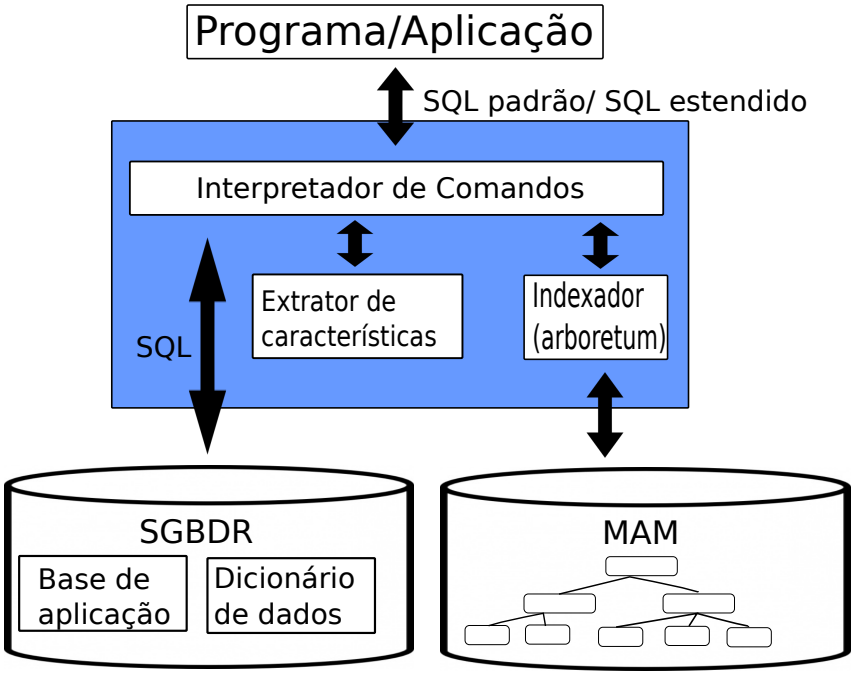

Figura 12 – Arquitetura do SIREN [2]

O principal componente do SIREN é o interpretador de comandos, o qual intercepta todos os comandos enviados pela aplicação. É ele quem faz as análises sintáticas e semânticas necessárias para o processamento dos comandos enviados. Ao receber uma sentença em SQL padrão, o SIREN retransmite a mesma para o SGBDR. Quando a sentença possui algum comando estendido que representa um operador por similaridade, o SIREN processa os operadores por similaridade e requisita em seguida ao SGBDR o processamento dos demais operadores.

O SIREN adiciona uma coleção de tabelas ao SGBDR que são usadas como dicionário dos dados métricos armazenado no sistema. Estas tabelas representam tanto a estrutura dos dados métricos como também dos extratores de características conforme ilustrado na Figura 13.

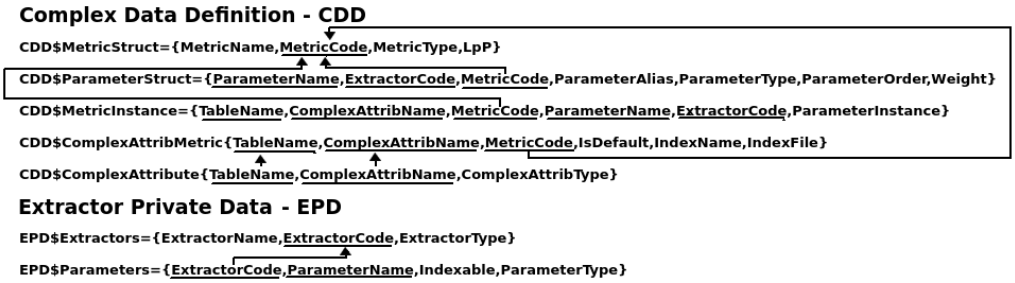

Figura 13 – Dicionário de dados do SIREN (retirada de [3]).

Os tipos de dados utilizados no armazenamento de dados complexos são: PARTICULATE e MONOLITHIC, criados por meio de extensões nos comandos DDL do SGBDR utilizado. Cada dado MONOLITHIC é armazenado em uma coluna do tipo binário e cada dado PARTICULATE é armazenado em um conjunto de atributos do tipo INTEGER de uma tabela criada pelo SIREN para o respectivo vetor de características. Dados MONOLITHIC têm seu domínio especializado nos dados STILLIMAGE e AUDIO, além de ser possível outras especializações. As métricas são associadas aos dados complexos através dos comandos CREATE METRIC, ALTER METRIC e DROP METRIC. Eles são responsáveis por criar uma métrica associada a um tipo de dado, a uma função de distância e a um extrator de características. Abaixo são ilustrados os comandos utilizados para a criação de métricas para os tipos PARTICULATE e MONOLITHIC, respectivamente.

```
--Cria métrica para dado PARTICULATE.
CREATE METRIC <nome_métrica> USING <função_distância> FOR PARTICULATE
(<nome parâmetro> <tipo parâmetro> [<peso>]
[, <nome parâmetro> <tipo parâmetro> [<peso>], ...]);

--Cria métrica para dado MONOLITHIC.
CREATE METRIC <nome_métrica> USING <função_distância> FOR STILLIMAGE
(<nome_extrator> (<nome_parâmetro> AS <alias_parâmetro> [<peso>], ...)
[, <nome_extrator> (<nome_parâmetro> AS <alias_parâmetro> [<peso>], ...),
...]);
```

A partir deste ponto a métrica pode ser associada ao dado complexo através de restrições. Desta restrição é possível criar um índice para ser utilizado nas buscas por similaridade, por meio do comando CREATE INDEX. São fornecidos exemplos destas duas formas de associação, respectivamente, nos códigos abaixo.

```
--Cria métrica para dado MONOLITHIC.
```

```
CREATE METRIC MyHistogram USING MANHATTAN FOR STILLIMAGE
(HISTOGRAMEXT (Histogram AS Histo));

--Associa a métrica criada a coluna fv.
CREATE TABLE cophir (
Id INTEGER PRIMARY KEY,
fv STILLIMAGE,
METRIC (fv) USING (MyHistogram));

--Cria índice através da métrica previamente criada.
CREATE INDEX cophir_ix ON cophir (fv)
USING MyHistogram;
```

A DML do SIREN estende o SQL padrão para representar onze operadores por similaridade: Rq, kNN, $\bowtie_{\mathrm{Rq}}$, $\bowtie_{k\mathrm{NN}}$, $\bowtie_{k\mathrm{CN}}$ e seis operadores que fazem seleção por similaridade de grupo ARq e k-ANNq, cada um através dos padrões `SUM`, `MAX` e `ALL`.

A Rq é representada através de dois comandos: `NEAR` e `RANGE`. O comando `NEAR` indica o centro de busca e o comando `RANGE` indica o raio da busca. Exemplo de uma Rq no `SIREN`:

```
--Rq no SIREN
SELECT id FROM cophir WHERE fv
NEAR 'exemplo.jpg' RANGE 1.5;
```

Os comandos **NEAR**, e **STOP AFTER** indicam respectivamente o centro de busca e o número de elementos do conjunto resposta a ser retornado pelo operador kNN. Abaixo é dado um exemplo de uso do kNN no `SIREN`:

```
--kNNq no SIREN
SELECT id FROM cophir WHERE fv
NEAR 'exemplo.jpg' STOP AFTER 10;
```

Analogamente é possível representar os operadores $\bowtie_{\mathrm{Rq}}$ e $\bowtie_{k\mathrm{NN}}$, respectivamente ilustrados abaixo:

```
--Junção por abrangência no SIREN
select cfv.id FROM cophir cfv, london lfv
WHERE cfv.fv NEAR lfv.fv RANGE 0.9;

--Junção por abrangência no SIREN
select cfv.id FROM cophir cfv, london lfv
WHERE cfv.fv NEAR lfv.fv STOP BY 10;
```

As consultas por múltiplos centros são feitas de forma análoga a Rq e kNN, através do uso dos métodos `SUM`, `MAX` e `ALL` fornecidos pela ferramenta. O exemplo abaixo ilustra uma k-ANNq representada no `SIREN`:

```
SELECT id FROM cophir WHERE fv NEAR SUM
(SELECT fv FROM london WHERE id = 1 OR id = 2 OR id = 3) STOP AFTER 1;
```

3.2 O Módulo `PostgreSQL-IE`

O `PostgreSQL-IE` [4] utiliza a SQL padrão para armazenar e recuperar dados complexos — mais especificamente imagens para sistemas CBIR (***C**ontent-**B**ased **I**mage **R**etrieval*). Ele foi desenvolvido no PostgreSQL e define UDTs e UDFs para o armazenamento de dados complexos e implementa funções de distância através de UDFs.

O `PostgreSQL-IE` suporta o armazenamento de imagens. Esta ferramenta possui um catálogo estendido para armazenar os metadados das imagens armazenadas como dados complexos (Figura 14). Dados complexos que são representados na ferramenta através dos tipos STILLIMAGE e PARTICULATE, provenientes da padrão SQL/MM.

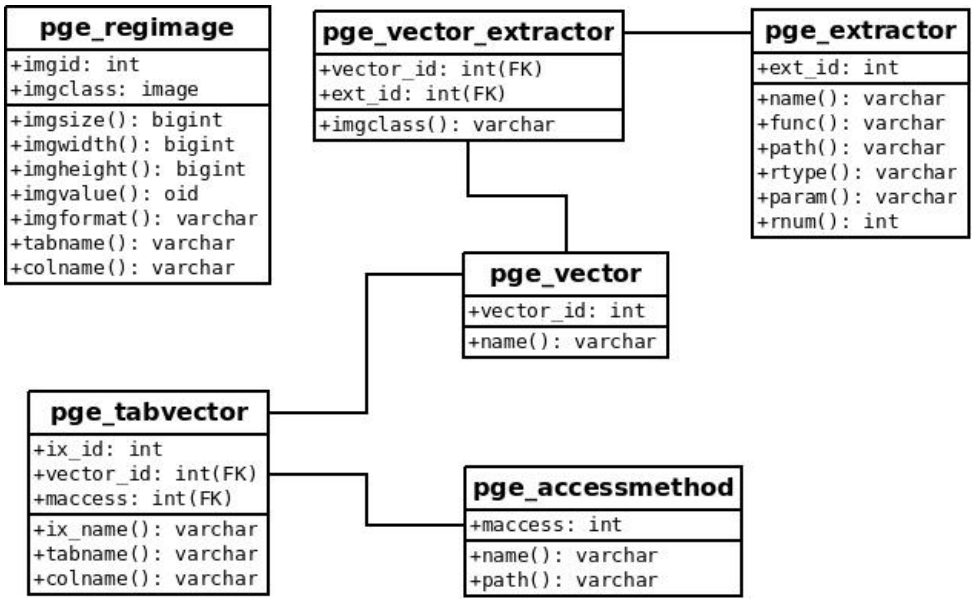

Figura 14 – Esquema de tabelas responsáveis por armazenar metadados no módulo `PostgreSQL-IE` (retirada de [4]).

A extração e inserção de dados complexos são feitos com UDF fornecidas pelo sistema `PostgreSQL-IE`. O usuário pode criar as UDFs de extração através de uma função, exemplificada a seguir, denominada *CREATE_EXTRACTOR*:

```
--Cria um novo extrator.
 SELECT Create_Extractor ('Fourier','getFourierDescriptor','float','ExtShape');
```

Os dados complexos são armazenados em um UDT, chamado PGImage, que contém (entre outras informações) um id para a tabela *pge_regimage* e o dado complexo armazenado em uma coluna binária. A tabela *pge_regimage* faz parte de um esquema que estende os metadados do PostgreSQL, cujo objetivo é armazenar informações das imagens armazenadas no sistema. A função *Define_Feature_Vector* define os vetores de características e os associa a um extrator — esta informação é armazenada nos metadados criados pelo SGBDR. Por fim, as imagens são inseridas por meio de uma UDF responsável por carregar todas as informações na tabela em que o dado complexo é armazenado e nos respectivos metadados.

A recuperação de dados complexos é feita por meio de UDFs que são executadas na clausula FROM da consulta SQL. Através das funções $RANGE$ e KNN, o `PostgreSQL-IE` representa e executa a seleção por abrangência e pelos k-vizinhos mais próximos, sendo que estas funções são executadas na cláusula FROM da consulta SQL. As referidas funções especificam os parâmetros do índice a ser utilizado pelo operador, além dos parâmetros do operador em questão. Um exemplo de utilização do KNN é ilustrado abaixo (o uso da função $RANGE$ é análogo):

```
--Busca kNN no PostgreSQL-IE
 SELECT cfv.id, Score_IE('sc_1', cfv.fv) score
 FROM cophir cfv
 WHERE cfv.id IN (SELECT * FROM KNN('sc_1',5,'UM_INDICE', cfv.fv))
 ORDER BY score;
```

Neste exemplo, o 'sc1' conecta o operador por similaridade a função $Score\_IE$ para que o resultado do operador em questão seja ordenado.

3.3 O *Framework* `MESSIF`

O `MESSIF` [5] é um *framework* que possibilita o armazenamento e a recuperação de imagens, textos e também fornece funções de distância para análise de proteínas. Seu objetivo é facilitar a criação de sistemas CBIR, pois permite o armazenamento centralizado ou distribuído dos dados complexos e a recuperação otimizada dos mesmos. Neste *framework*, a recuperação de dados complexos pode ser feita através da linguagem SQL estendida *SimSeQL* [7] e as consultas são otimizadas por meio de indexação e estatísticas. Por possuir uma arquitetura aberta e processamento de dados métricos genérico, o `MESSIF` possibilita que novos módulos sejam desenvolvidos e acoplados à sua arquitetura (a qual é modular e preparada para a adição de novos módulos).

O armazenamento de dados complexos no `MESSIF` é feito através de *buckets*, cada um dos quais representa uma partição do espaço métrico. Estes *buckets* podem ser arma-

zenados em diferentes *peers* de um sistema computacional distribuído, conforme ilustrado na Figura 15. Em uma rede distribuída, o `MESSIF` é capaz de indexar os dados armazenados na rede através de uma indexação por camadas e o usuário pode escolher a técnica de indexação a ser utilizada pelo sistema. Nas redes distribuídas, os *peers* se comunicam através de mensagens TCP e UDP e a sua estrutura é *peer-to-peer*, podendo cada *peer* acessar os recursos e comunicar-se diretamente com qualquer outro *peer* através de um endereço único que cada um deles possui.

O `MESSIF` é capaz de fazer buscas apenas nos *buckets* que possuem dados de interesse utilizando pivôs para representar cada *bucket*. Para isso o sistema possui uma estrutura de árvore de navegação e cada *bucket* é capaz de procesar o operador separadamente.

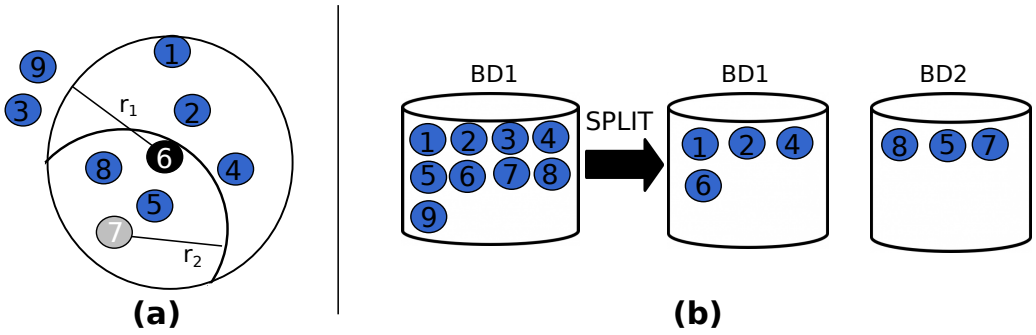

Figura 15 – (a) Partições em forma de esfera do espaço métrico e (b) o armazenamento das partições divididas em um conjunto de banco de dados distribuídos em rede [5].

Além de serem otimizadas por índices, as buscas por dados complexos no `MESSIF` possuem otimizações baseadas em estatísticas coletadas automaticamente pelo sistema. Adicionalmente, a implementação deste sistema permite ao usuário criar estatísticas específicas para novos operadores que sejam adicionada s a estrutura modular do sistema.

A linguagem *SimSeQL* é capaz de representar vários operadores por similaridade, contudo, propõe extensões à linguagem SQL. A sintaxe do operador *SELECT* é apresentada a seguir:

```
SELECT [TOP n | ALL] {attribute | ds.distance | ds.rank | f(params)} [, ...]
FROM
{dataset | SIMSEARCH [:obj [, ...]]
IN data source AS ds [, data source2 [, ...]]
BY {attribute [DISTANCE FUNCTION função de distância(params)]
| função de distância(params)} [METHOD method(params)]
WHERE /* Restrição sobre os valores dos atributos */
ORDER BY {attribute | ds.distance [, ...]}
```

3.4 O Sistema SimDB

O SimDB e suas variações (DBSimJoin [6] e o i-SimJoin [72]), os quais são chamados resumidamente neste trabalho de SimDB, é uma ferramenta que armazena vetores de características provenientes de qualquer domínio de dados complexos.

Estes vetores de características são armazenados em tabelas do modelo relacional de uma versão modificada do PostgreSQL, a qual reconhece uma linguagem SQL estendida que reconhece operadores por similaridade através de novos comandos. As informações armazenadas no SimDB podem ser indexadas pelo eD-Index.

O SimDB foi implementado através da modificação das árvore sintática e de consultas do PostgreSQL conforme ilustrado pela Figura 16. Este sistema é adequada para demonstrar que implementar em SGBDRs uma linguagem estendida gera efeitos colaterais, neste caso em específico a ferramenta deixa de ser capaz de processar os operadores tradicionais. Este tipo de solução não é adequado portanto para sistemas comerciais que requerem a execução de operadores por similaridade e operadores tradicionais simultaneamente.

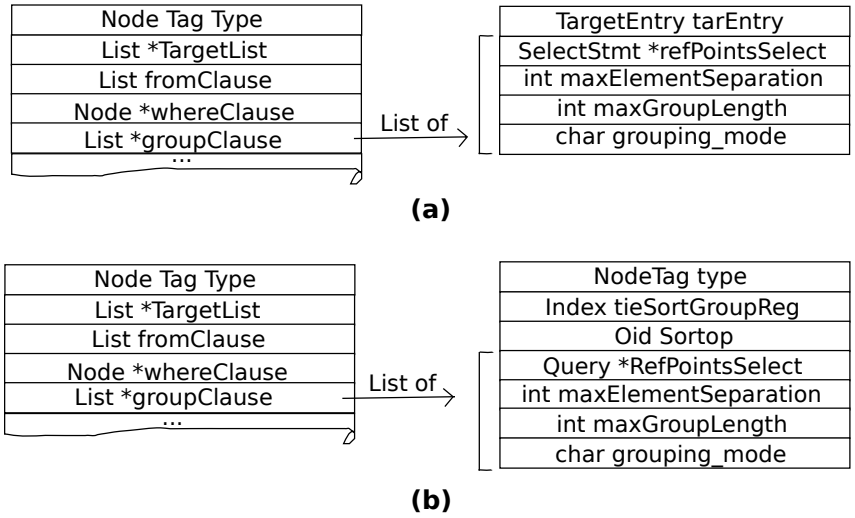

Figura 16 – (a) Modificações na árvore sintática e (b) modificações na árvore de consultas do PostgreSQL feitas no desenvolvimento do SimDB (extraída de [6]).

A linguagem do SimDB representa um conjunto de operadores por similaridade físicos. Além disso, esta ferramenta possibilita a execução de planos de execução alternativos em alguns casos. A representação da seleção por abrangência no DBSimJoin requer que o usuário informe todas as colunas que armazenam cada uma das características, conforme o seguinte exemplo:

```
SELECT * FROM cophir S
WHERE EucDist([S.s1 ... S.s9],[0.02 ... 0.02])<=1.5;
```

A junção por abrangência no SimDB é exemplificada no exemplo que segue. Esta ferramenta exige a extensão da linguagem SQL, pois utiliza as palavras WITHIN e OF, ambas não previstas no padrão.

```
SELECT cfv.id FROM cophir cfv, london lfv
WHERE cfv.fv WITHIN 1.5 OF lfv.fv;
```

O SimDB prevê também o uso de operadores de agrupamento por similaridade representados por uma gramática estendida. Este trabalho propôs representações para três operadores por agrupamento unidimensionais: SGB-U, SGB-A, e o SGB-D. As extensões na SQL padrão são feitas na cláusula GPOUP BY e as representações dos três operadores são respectivamente ilustrados a seguir. No exemplo são considerados dados de temperatura, por serem unidimensionais.

```
SELECT ...
GROUP BY cophir
MAXIMUM_ELEMENT_SEPARATION 2 MAXIMUM_GROUP_DIAMETER 6;

SELECT ...
GROUP BY cophir
AROUND {30,50} MAXIMUM_ELEMENT_SEPARATION 2 MAXIMUM_GROUP_DIAMETER 20;

SELECT ...
GROUP BY cophir
DELIMITED BY (SELECT Value FROM Thresholds);
```

3.5 O Módulo MSQL

O módulo MSQL [10] é capaz de armazenar e recuperar dados complexos utilizando a SQL padrão. Para isso, este SGBDS fornece um conjunto de UDFs que tornam o SGBDR capaz de recuperar dados complexos por meio dos operadores Rq e k-NN. Os vetores de características são inseridos em uma tabela do modelo relacional R e um índice construído com uma árvore B^+ inserido na coluna I que é adicionada à tabela R.

O usuário do banco de dados deve inserir os vetores de características extraídos dos dados complexos sendo trabalhados em uma tabela do modelo relacional. Estas informações podem ser indexadas através dos passos que seguem abaixo, sendo a PTable uma tabela previamente criada com os *pivots* da árvore B+ a ser criada. Devem ser criados *triggers* na tabela cophirfv para que o índice se mantenha atualizado quando itens forem adicionados, removidos ou atualizados.

```
CREATE TYPE iPair as (key INTEGER, value DOUBLE);

ALTER TABLE cophir ADD I iPair;

UPDATE cophir R
set R.I = (SELECT (S.PID, DIST(S[A], R[A]))::iPair as I1 FROM PTable S
ORDER BY I1.value LIMIT 1);

CREATE INDEX I-INDEX ON R USING BTREE (I ASC NULL LAST);
```

Uma consulta por abrangência no `MSQL` é representada conforme exemplificado abaixo.

```
SELECT R.a1, R.a2, ..., R.an
FROM cophir R
WHERE LOCATE(R.I, @ranges) AND DIST(R[A], q[A], threshold);
```

A função de distância é representada neste caso pela função `DIST`, que recebe a tabela R, um centro de busca q e o raio de abrangência da consulta, representado pelo parâmetro `threshold` . Já a função `LOCATE` tem o objetivo de filtrar os valores de acordo o índice e os intervalos presentes em `@ranges` podem ser gerados por um procedimento predefinido no sistema.

3.6 O Módulo `FMI-SiR`

O `FMI-SiR` [8] é um módulo que utiliza o SQL padrão e foi implementado no Oracle. Nele os vetores de características são armazenados em colunas binárias das tabelas relacionais e a recuperação dos mesmos é feita através da implementação dos operadores por similaridade em UDF.

O armazenamento de vetores de características no `FMI-SiR` é feito através de UDF fornecidas pelo módulo. Estas UDFs serializam os vetores de características em colunas BLOB do SGBDR. As UDFs do `FMI-SiR` são implementadas utilizando-se o C/C++.

O único comando que a implementação do `FMI-SiR` no Oracle utiliza que não é previsto no padrão SQL é o `CREATE OPERATOR`. Ele é utilizado para a definição de um operador, o qual é necessário para a indexação das informações armazenadas e faz parte do *Oracle Data Cartridge*, o qual permite a implementação de índices e outros recursos definidos pelo usuário. O usuário pode inserir vetores de características e indexá-los no `FMI-SiR` através dos comandos a seguir.

```
--Cria tipo com funções a serem utilizadas na criação e manutenção do índice.
```

```
CREATE OR REPLACE TYPE index_im_type AS OBJECT (
scanctx RAW(4),
STATIC FUNCTION ODCIIndexCreate(),
STATIC FUNCTION ODCIIndexDrop(),
STATIC FUNCTION ODCIIndexInsert(),
STATIC FUNCTION ODCIIndexDelete(),
STATIC FUNCTION ODCIIndexUpdate(),
STATIC FUNCTION ODCIIndexStart(),
MEMBER FUNCTION ODCIIndexFetch(),
MEMBER FUNCTION ODCIIndexClose());

--Cria operador a partir da UDF que implementa a função de distância
CREATE OPERATOR Manhattan_dist BINDING (BLOB, BLOB)
RETURN FLOAT USING Manhattan_distance;

--Cria o tipo de índice a partir do operador e das funções usadas
--para a manutenção do índice.
CREATE INDEXTYPE Slim_Manhattan FOR
Manhattan_dist(BLOB, BLOB)
USING index_im_type;

--Cria o índice
CREATE INDEX cophir_Slim_Manhattan_idx ON cophir(fv)
INDEXTYPE IS Slim_Manhattan PARAMETERS ('8192');
```

A recuperação de dados complexos com o `FMI-SiR` é feita através de UDFs e operadores, utilizando o SQL padrão. A seguir é ilustrado como uma seleção e uma junção por abrangência são representadas, respectivamente, utilizando o operador `Manhattan_dist` é representado.

```
--Seleção por abrangência indexada
SELECT cfv.id FROM cophir cfv
WHERE Manhattan_dist(cfv.fv,
(SELECT fv FROM london c WHERE c.id=988)) <= 24;

--Junção por abrangência indexada
SELECT cfv.id FROM cophir cfv, london lfv
WHERE Manhattan_dist(cfv.fv, lfv.fv) <= 24;
```

3.7 Uso da XML para o Armazenamento e Recuperação de Dados Complexos

Além das propostas baseadas em SQL para representação do armazenamento e recuperação de dados complexos, há na literatura propostas de representação utilizando a XML (*e**X**tensible **M**arkup **L**anguage*) [7]. Adicionalmente, mecanismos de manipulação de XML foram embutidos na SQL padrão, sendo que estes podem ser utilizados em modelos de armazenamento e recuperação de dados complexos.

A XML utiliza elementos e atributos indicados nos documentos através de marcas. Esta linguagem se destaca no mercado por sua flexibilidade, extensibilidade e interoperabilidade, por isso ela é amplamente utilizada.

Dois motivos fazem da XML uma linguagem extensível. Primeiramente, o desenvolvedor pode criar suas próprias DTDs e/ou *schemas* extensíveis, os quais podem ser usadas em diversas aplicações. Em segundo, a própria linguagem XML pode ser estendida para outras linguagens. A linguagem XML é interoperável. Já que é possível utilizá-la em um grande número de plataformas. Os interpretadores de documentos baseados nesta linguagem têm relativamente baixo custo, pois sua estrutura é consistente. De fato, devido a esta interoperabilidade o XML complementa o Java em diversas aplicações, por exemplo. A linguagem XML é semi-estruturada. Para isso ela provém duas formas de se definir como os seus documentos são formados: o XML *schema* e o DTD (**D**ocument **T**ype **D**efinition). Destes, o primeiro é mais utilizado, por isso será explorado mais a fundo neste texto.

Ao definir o XML *schema* para um atributo, todos os documentos XML que são inserido no mesmo necessitam ser validados de acordo com as regras do mesmo. Para criar estas regras, a linguagem aceita alguns tipos predefinidos de dados como texto (*string*), lógico (`boolean`), número real (`float` e `double`), dentre outros. Além disso, é possível criar um refinamento para os tipos existentes através do tipo simples (`simple type`) e estruturar elementos filho e atributos de um elementos através do tipo complexo (`complex type`).

Após armazenar os dados XML no Banco de Dados o usuário pode recuperá-los utilizando as linguagens XPath ou XQuery. A XPath permite que o usuário selecione os dados especificando o caminho em que os dados se encontram nos documentos da mesma forma que se especifica o caminho de um arquivos nos sistemas operacionais. Já a XQuery utiliza a sintax FLWOR, juntamente com construções XPath, a qual permite a utilização de associação de valores a variáveis e a interação das mesmas em valores intermediários.

Alguns SGBDs possuem meios nativos para armazenar documentos XML em tipos de dados definidos utilizando recursos objeto-relacionais [83]. O Oracle, por exemplo, fornece o tipo XMLType (tipo previsto no SQL padrão com o nome de XML), que suporta

armazenamento de documentos XML na forma binária (em atributos BLOB), relacional ou em atributos CLOB. Este tipo permite que o usuário execute operações relacionais em dados armazenados como documentos XML, além de possuir métodos para extrair, criar e indexar dados XML, e também funcionalidades disponibilizadas por APIs PL/SQL e Java. Enquanto o armazenamento dos documentos XML em atributos CLOB é feito na íntegra (valores e rótulos), o que faz com que este modelo seja mais utilizado por um número limitado de aplicações que precisam acessar os documentos XML com todos os seus rótulos, o armazenamento de documentos XML em atributos binários guarda somente o conteúdo do mesmo — i.e., os rótulos não são incluídos no armazenamento. Além disso, são feitas outras otimizações para que as informações sejam guardadas de forma mais compacta — espaços em branco desnecessários são removidos, alguns tipos de dados são convertidos, entre outros [84]. Adicionalmente, o Oracle é capaz de armazenar documentos XML no modelo relacional. Este modelo de armazenamento pode proporcionar um melhor desempenho na recuperação de dados em relação às outras formas de armazenamento para alguns tipos de busca (e.g., filtros em caminhos específicos dos documentos), e por isso é mais eficiente para aplicações que fazem uso deste tipo de buscas.

Uma vez que os dados estão inseridos no modelo relacional, os mesmos podem ser recuperados utilizando o mesmo modelo com o auxílio de sentenças XQuery. Para isso usa-se a função $XMLCast(XMLQuery())$, que é responsável por converter a busca por dados em documentos XML para uma busca por dados no esquema relacional. Além disso, é possível utilizar objetos DOM em funções PL/SQL ou em funções externas ao SGBDR para construir a hierarquia do documento XML e navegar pelos seus nós.

3.8 Discussão

A Tabela 1 sintetiza as características das principais propostas de SGBDS encontradas na literatura e apresentadas neste capítulo. É possível verificar que os trabalhos anteriores divergem em dois aspectos principais, que são o foco deste trabalho: *(i)* algumas ferramentas utilizam a SQL padrão e outras a estendem e *(ii)* os trabalhos anteriores divergem quanto ao tipo de armazenamento que deve ser usado para as informações do dado complexo. As ferramentas que utilizam a SQL padrão não são capazes de otimizar o plano de execução das consultas, pois as funções de distância não formam blocos de busca que possam ser reconhecidos como operadores por similaridade. Embora algumas das propostas que estendem a SQL sejam capazes de otimizar o plano de execução das consultas, utilizam a representação da SQL estendida, o que não é o ideal, conforme pontos já apresentados. Já com relação ao tipo de armazenamento, não há nenhum trabalho que faz uma análise aprofundada de diferentes opções, indicando suas vantagens e limitações. Este trabalho visa abordar esses dois aspectos, utilizando construções da SQL padrão para representar operadores por similaridade e explorando quatro diferentes modelos de

Tabela 1 – Comparação entre sistemas que armazenam e recuperam dados complexos

	Extensão do SQL	Otimização de consultas	Armazenamento de dados complexos	Operadores por similaridade
`SIREN` [2]	Grande	Não	Binário	Compilador próprio
`PostgreSQL-IE` [4]	Não se aplica	Não	Binário	UDF/UDT
`MESSIF` [7]	Grande	Não	Semiestruturado	Compilador próprio
`SimDB` [9]	Grande	Baseada em regras	Relacional	Código-fonte do SGBDR modificado
`MSQL` [10]	Pequena	Baseada em regras	Relacional	UDF/UDT
`FMI-SiR` [8]	Pequena	Não	Binário	UDF e operadores

dados suportados no padrão SQL. As soluções propostas são apresentadas no próximo capítulo.

4 PROPOSTA DE INCLUSÃO DE SIMILARIDADE EM SGBDR USANDO A SQL PADRÃO

Conforme apresentado nos capítulos anteriores, trabalhos passados desenvolveram diversos SGBDSs capazes de armazenar e recuperar dados complexos. Contudo, não há na literatura análise dos tipos de vetores de características e de funções de distância existentes na mesma ou do modelo de armazenamento mais adequado para se armazenar dados complexos em SGBDRs.

Este capítulo apresenta uma proposta baseada em SQL padrão para inclusão de recursos para armazenamento de dados complexos e recuperação por similaridade em SGBDR, com objetivo de preencher os *gaps* expostos ao *(i)* categorizar os vetores de características e as funções de distância existentes na literatura (Seção 4.1), *(ii)* ilustrar a representação de um conjunto consistente de operadores por similaridade através da SQL padrão (Seção 4.2), *(iii)* apresentar uma taxonomia dos modelos de armazenamento de dados complexos possíveis utilizando a SQL padrão (Seção 4.3) e *(iv)* analisar a implementação da taxonomia proposta em uma discussão das vantagens e limitações de seus modelos de armazenamento (Seção 4.4). Grande parte das contribuições apresentadas neste trabalho foram publicadas em [85, 86].

4.1 Categorização de Vetores de Características e de Funções de Distância

Um dado complexo pode ter inúmeros atributos que o descrevem de alguma forma. Este trabalho define o tipo genérico *vetor de características* (*feature vector* — FV) a todo atributo, ou conjunto de atributos, que é comparado utilizando-se uma função de distância. As operações definidas para cada tipo distinto de FV é conjunto de funções distância aplicáveis a ele.

Embora um FV seja um tipo de dado abrangente, uma vez que correspondem a instâncias em qualquer domínio $\mathbb{S}$, a grande maioria dos FVs encontrados na literatura podem ser classificados em três dimensões: *(i) dimensionais* ou *adimensionais*; *(ii) simples* ou *compostos*; e *(iii) homogêneos* ou *heterogêneos*.

Um *FV Dimensional* tem um número fixo de características numéricas. Este é o tipo de FV mais comum para dados complexos. Já os FVs adimensionais são conjuntos de características numéricas em que o número de elementos não é fixo, denominados *FVs Numéricos Adimensionais*, ou conjuntos de características não numéricas. FVs Numéricos Adimensionais são usados, por exemplo, para representar características extraídas de regiões de interesse de imagens, cujo número pode variar de imagem para imagem. FVs

adimensionais não numéricos podem ser divididos em *FVs de Caracteres*, usados, por exemplo, para representar sequências genéticas, e *FVs de Textos*, que permite representar, por exemplo, *tags* descritivas associadas a uma imagem, no estilo *bag of words*, ou características correspondentes a atributos categóricos.

Um *FV Simples* tem características pertencentes ao mesmo domínio de dados tradicionais. Os metadados necessários a um FV Simples são o número de características e o tipo de dados das características. A Figura 17(a) ilustra o caso em que o tipo de dados é de tamanho fixo e é armazenado de alguma forma no catálogo do banco de dados, mas o número de características é associado ao FV, o que permite representar vetores de diferentes tamanhos. Esse tipo de estrutura permite armazenar FVs Dimensionais, FVs de Caracteres e FVs Numéricos Adimensionais. Note-se que o número de características poderia ser armazenado no catálogo do banco para os dois primeiros casos, mas FVs numéricos adimensionais exigem que o número de características esteja no vetor, por isso essa estrutura é a que engloba essas três possibilidades. Já no FV da Figura Figura 17(b) o tipo de dados das características também está embutido. Isso mantém o FV autocontido, entretanto, esse dado é repetido para todos os valores de atributo desse tipo em uma tabela, o que ocasiona desperdício de espaço. Por fim, a Figura Figura 17(c) ilustra o caso em que o tipo de dados das características é de tamanho variante, como os FVs de Texto, onde a estratégia de representação usada contém um cabeçalho com um conjunto de entradas no estilo [`tamanho,referência para o conteúdo`] seguido do conteúdo de cada característica.

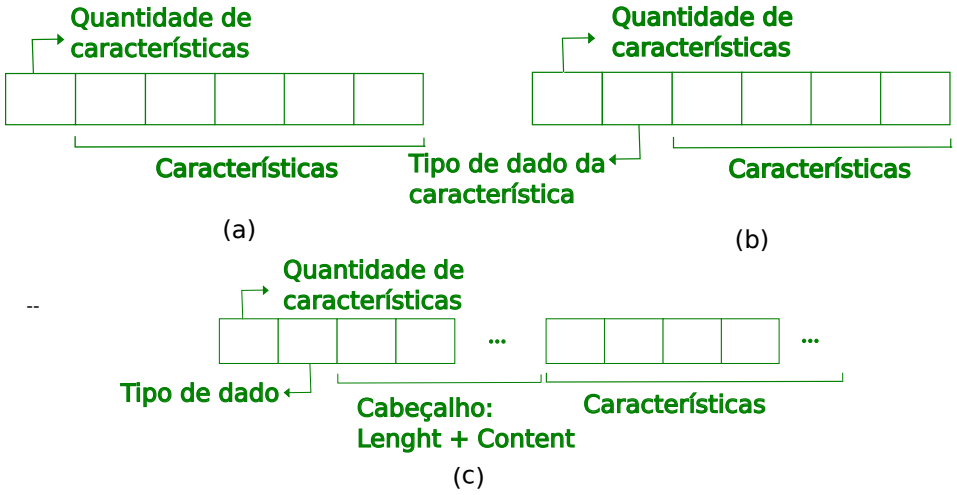

Figura 17 – Vetor de características homogêneo. (a) Características com tipo de dados pré-definido. (b) Características podem ser de diversos tipos de dados. (c) Vetor de características adimensional.

FVs Compostos consistem em um conjunto de FVs Simples, em que cada FV Simples também é chamado de *componente*. FVs Compostos podem representar simultaneamente diversos aspectos de um dado complexo, tais como, para uma imagem, FVs

Simples baseados em cor, textura, formas, rótulos, regiões de interesse e anotações. Nossa análise é que os FVs compostos podem ser subdivididos entre predefinidos e dinâmicos. *FVs Compostos Predefinidos* (Figura 18(a)) têm a estrutura definida a priori, enquanto *FVs Compostos Dinâmicos* (Figura Figura 18(b)) possibilitam que componentes sejam adicionados e/ou removidos de acordo com a necessidade, definindo a estrutura a posteriori, de acordo com o uso.

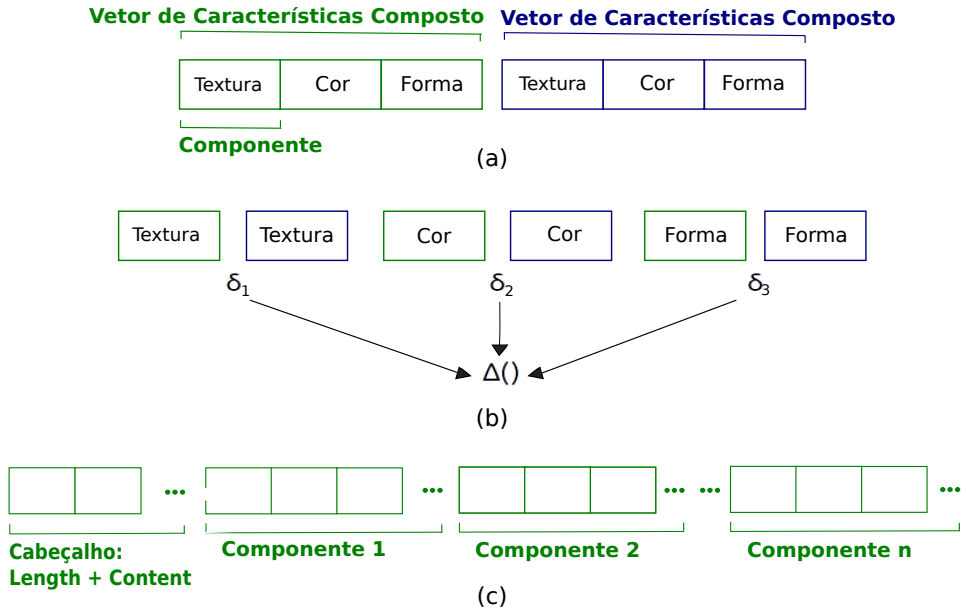

Figura 18 – (a) FVs Compostos Predefinidos. (b) Aplicação de Métrica Produto em um par de FVs Compostos Predefinidos. (c) FV Composto Dinâmico.

Por fim, os *FVs Homogêneos* constituem uma categoria de FVs simples cujas características são representadas por dados pertencentes a um mesmo domínio (Figura Figura 17) e os *FVs Heterogêneos* possuem características que pertencem a diferentes domínios (e.g., números, textos/palavras e atributos categóricos). Um FV Heterogêneo pode ser considerado um caso particular de FV Composto, cujos componentes são FVs Homogêneos de forma a representar todos os tipos diferentes de características do vetor. Note-se que um FV Composto Homogêneo pode ser convertido em um FV Simples Homogêneo, sendo que, nesse caso, não haveria mais distinção entre os subgrupos de características.

Considerando as funções de distância (também referenciada como FD neste estudo), a classificação proposta é: *(i)* simples ou composta; e *(ii)* de um ou múltiplos centros de consulta.

FDs Simples podem ser aplicadas a FVs Simples Homogêneos. Exemplos de FDs Simples são as funções da família Minkowski, que podem ser aplicadas a FVs (Simples) Dimensionais cujas características estejam no domínio $\mathbb{R}$, a FD de Edição (Levenshtein ou *Edit*), aplicada a FVs Adimennsionais de Caracteres, e a FD de Jaccard, aplicável a qualquer domínio de conjuntos, tipicamente FVs de Textos [12, 87]. *FDs Compostas*

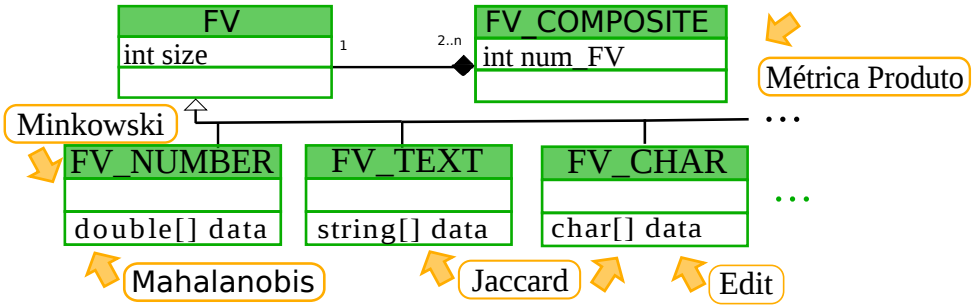

Figura 19 – Hierarquia de classes de vetores de características proposta e exemplos de funções de distâncias definidas sobre os tipos.

são aplicadas a FVs compostos (o que engloba os FVs Heterogêneos). FDs Compostas também são conhecidas na literatura como FDs de múltiplos descritores [44]. Como mostra a Figura 18(b), uma FD Composta agrega em um único valor os resultados de FDs Simples calculadas sobre os componentes de um FV Composto, sendo que a forma mais comum de agregação é a utilização de uma Métrica Produto. Um exemplo típico de Métrica Produto é a combinação linear dos resultados das FDs Simples.

A proposta desse trabalho é consolidar todas essas variações de FVs nos tipos de dados indicados na hierarquia de classes da Figura 19. Na figura, FV_NUMBER permite representar FVs Dimensionais e FVs Numéricos Adimensionais, ao passo que FV_TEXT e FV_CHAR permitem representar, respectivamente, FVs de Textos e FVs de Caracteres, e o tipo FV_Composed é para FVs Compostos. A figura também indica exemplos de FDs associadas a cada tipo de dados. Quanto a implementação das FDs, elas podem ser implementadas de acordo com a hierarquia proposta na Figura 20. Desta forma, o usuário pode implementar as FDs sem se preocupar em escrever o código das verificações necessárias à implementação de uma FD feita para FVs dimensionais, por exemplo.

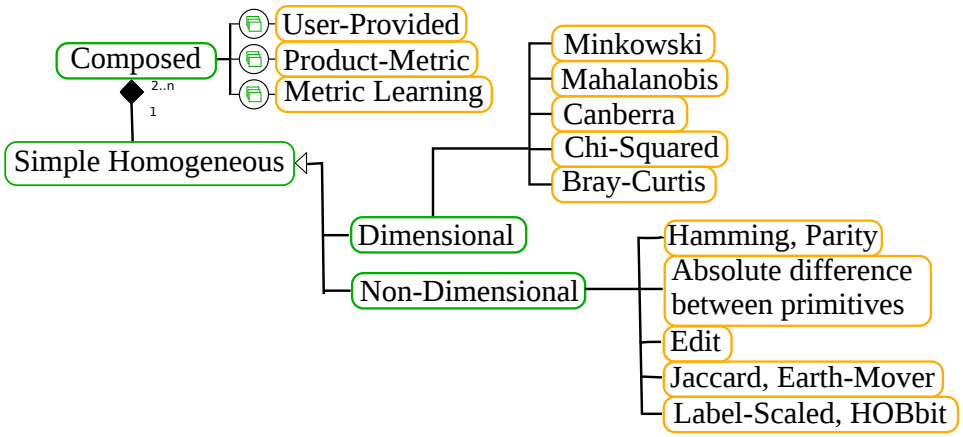

Figura 20 – Hierarquia proposta de funções de distância.

4.2 Representação de Consultas por Similaridade

No que tange à representação de consultas por similaridade, nota-se que os trabalhos existentes: *(i)* abordam um conjunto reduzido de operadores; ou *(ii)* propõem extensões significativas à SQL de forma a compreender uma gama maior de operadores. A proposta deste trabalho é que é possível representar a maioria dos operadores por similaridade em SQL padrão apenas definindo funções de distância de forma adequada.

Esta seção apresenta uma proposta de representação de um conjunto consistente de operadores por similaridade de forma a possibilitar o processamento do respectivo operador físico dos mesmos. Para isso, sugerimos a extensão do comando DDL de criação de função de distância `CREATE FUNCTION` com uma palavra opcional que diferencie as funções de distância das demais funções do SGBD. Propõe-se que essa palavra seja `DISTANCE`. A diferenciação de funções de distância possibilitaria ao processador de consultas identificar um operador por similaridade e mapeá-lo para o respectivo operador físico. Consequentemente, seria possível a execução de algoritmos eficientes e que o SGBDR escolhesse o plano de execução mais adequado. Além disso, é proposto que seja possível associar índices a operadores por similaridade, por exemplo, usando o comando `CREATE OPERATOR` do SGBDR Oracle.

As seções a seguir detalham a representação proposta para os diferentes operadores por similaridade considerados.

4.2.1 Representação de Funções de Distância

A forma trivial de representar uma função de distância é por meio de uma função SQL que recebe como parâmetros dois FVs de um mesmo tipo (e.g., `FV_Type`) e retorna a distância entre eles.

```
--modelo de função de distância do eFMI-SiR
distance_<nome_da_FD>(FV_Type, FV_Type);
```

O exemplo a seguir mostra como representar utilizando uma função desse tipo (`Manhattan_distance`) uma seleção por abrangência sobre a tabela `cophir`, tendo como elemento de consulta o elemento que tem `id=2`, e uma junção por abrangência entre as tabelas `cophir` e `london`.

```
-- Seleção por abrangência
SELECT id
FROM cophir WHERE Manhattan_distance(fv,
 (SELECT fv FROM cophir WHERE id=2)) <= 10;
```

```
-- Junção por abrangência
SELECT c.id
FROM cophir c, london l
WHERE manhattan_distance(c.fv, l.fv) <= 10;
```

Embora essa representação seja executável, o SGBDR só poderá realizar adequadamente um processo de otimização caso consiga mapear univocamente essa sintaxe para o operador físico por similaridade de seleção por abrangência. Caso esse mapeamento seja factível, as regras específicas de reescrita de consulta aplicáveis ao operador físico por similaridade utilizado podem ser aplicadas, bem como a escolha de algoritmos e índices específicos a esse operador. Para possibilitar isso, nossa proposta é funções de distância sejam identificadas de forma diferenciada de outras funções. Neste tralho, propomos estender o comando DDL `CREATE FUNCTION` e seus comandos complementares (`ALTER`, `DROP`, etc.) com a palavra reservada `DISTANCE` para indicar que se trata de uma função de distância. Desta forma, ao detectar uma função de distância em uma consulta, o processador de consultas pode fazer sem ambiguidade o processo de tradução para o operador por similaridade correspondente.

Para que algoritmos e índices específicos sejam utilizados para agilizar a execução de operadores por similaridade, novamente é preciso identificar uma correspondência unívoca. Essa correspondência depende: *(i)* do domínio do vetor de características, *(ii)* da função de distância e *(iii)* dos parâmetros da função de distância, pois diferenças que afetam a distribuição de distâncias no espaço de busca inviabilizam o uso de índices. Por exemplo, se existe um índice para um atributo de um determinado tipo de FV usando a função de distância Euclidiana, esse índice não pode ser utilizado em consultas envolvendo outras distâncias ou mesmo em consultas que utilizam a distância Euclidiana para o mesmo atributo, mas com diferenças nos pesos das características.

O domínio do vetor de características pode ser definido por meio do tipo de FV correspondente ao atributo e por restrições de integridade eventualmente associadas[1]. As funções de distância podem ser escolhidas a partir de um conjunto básico oferecido pelo SGBDR ou definidas pelo usuário. Já os parâmetros de funções de distância precisam ser associados a cada instanciação diferente de função de distância. A parametrização de uma função de distância pode ser feita a priori ou na invocação da função. Como o número de parâmetros pode ser grande, fazê-lo na invocação resulta em uma sintaxe carregada e impede a indexação em consultas. Portanto, nossa proposta é parametrizar a priori, associando um rótulo a cada parametrização, ou seja, o usuário deverá criar uma

[1] Restrições de integridade não são objeto desse trabalho, mas, por exemplo, a um atributo do tipo FV_NUMBER de uma tabela poderia ser adicionada uma restrição de integridade que verifique o tamanho (`size`) do FV, garantindo que todos os FVs armazenados tenham a mesma dimensionalidade. Outra alternativa é as restrições serem verificadas em tempo de execução da respectiva função de distância.

função de distância com a parametrização desejada. A criação de índices sobre atributos de tipos FV também deve indicar a parametrização utilizada para que o processador de consulta possa identificar sem ambiguidade os índices elegíveis para otimização, bem como a estrutura de dados utilizada, quando for o caso. Com isso, no exemplo a seguir, o otimizador poderia traduzir a instrução SQL em uma Junção por Abrangência que retorna os pares de imagens da tabela `cophir` e da tabela `london` (imagens de Londres) cuja distância entre os seus FVs (de tipos compatíveis) é limitada a `10` unidades, utilizando a função de distância `my_weighted_manhattan` e, eventualmente, utilizar o índice `cophir_my_weighted_manhattan_ix`, cuja estrutura é uma Slim-tree (neste exemplo, o FV é representado por um VArray).

```
-- Criação de uma distância parametrizada
CREATE DISTANCE FUNCTION my_weighted_manhattan
  (fv1 FV_Type, fv2 FV_Type)
  RETURN NUMBER
AS
  total NUMBER := 0;
  weights FV_Type := FV_Type(1, 2, 2, ...);
BEGIN
  --multiplica os valores contidos no VArray pelo respectivo peso
  IF fv1.count = fv2.count THEN
    FOR i IN 1..fv1.count LOOP
      total := total + (weights(i) * ABS(fv(i) - fv2(i)));
    END LOOP;
  ELSE
    raise_application_error(-20001, 'FVs must be of the same size');
  END IF;
  RETURN total;
END;
/

...;

-- Criação de um índice por similaridade
CREATE INDEX cophir_my_weighted_manhattan_ix
 ON cophir(fv)
 USING slimtree my_weighted_manhattan;

-- Junção por abrangência
SELECT l.image AS limg, c.image AS cimg
FROM cophir c, london l
```

```
WHERE my_weighted_manhattan(c.fv, l.fv) <= 10;
```

Analogamente, operadores que envolvam a manipulação de múltiplos centros de consulta podem ser representados por meio de funções de distâncias específicas definidas pelos usuários e devidamente parametrizadas a priori. Neste caso é possível implementar (i) funções de distância que recebem cada centro em um parâmetro distinto e (ii) que recebem os múltiplos centros em parâmetros únicos.

A solução *i* permite um número grande de centros de consulta, pois as UDFs podem ter valores *default* nulos para os parâmetros de entrada. Adicionalmente, esta não necessita de preparação prévia pois o usuário pode selecionar os centros individualmente, indicando-os. No entanto esta última solução não é adequada para consultas que envolvem um número grande de centros. A alternativa *ii* pode ser implementada através de um tipo de dado que é um VArray de FVs. Esta solução permite um número potencialmente ilimitado de centros através de uma representação limpa.

Quanto à otimização, em ambos os casos o otimizador de consultas pode identificar que se trata de um operador de múltiplos centros a partir dos parâmetros da função de distância, que representam um conjunto de elementos e, portanto, em número maior (alternativa *i*) ou de um tipo de dado específico (alternativa *ii*).

Analisando todos os pontos levantados acima, o uso da solução *ii* é recomendado, pois ela facilita o uso de um número grande de centros de forma mais limpa.

Exemplo da solução *i*:

```
--Criação de uma função de distância por agregação.
CREATE DISTANCE FUNCTION my_aggregate_function_i
  (FV FV_Type, Grip_Factor NUMBER,
   FV_Center1 FV_Type, Weight1 NUMBER,
   FV_Center2 FV_Type, Weight2 NUMBER,
   ...,
   FV_CenterN FV_Type, WeightN NUMBER) RETURN NUMBER ...;

--Consulta por Abrangência Agregada
SELECT id FROM cophirfv cfv WHERE my_aggregate_function_i
(cfv.fv, 2,
 (SELECT fv FROM londonfv WHERE id=1), 1
 (SELECT fv FROM londonfv WHERE id=3), 2) <= 20;
```

Exemplo da solução *ii*, onde `slim_mass` é uma implementação da Slim-tree que possui algoritmos sobre este índice para a execução de consultas por similarida agregada.

```
CREATE TYPE FV_SET_TYPE AS TABLE OF FV_Type;
```

```
CREATE DISTANCE FUNCTION my_aggregate_function_ii
  (FV FV_Type, Grip_Factor NUMBER, FV_Center_Group FV_SET_TYPE)
  RETURN NUMBER ...;

--Consulta por Abrangência Agregada
DECLARE
  centers FV_SET_TYPE;
BEGIN
  SELECT fv BULK COLLECT INTO centers
    FROM cophir WHERE city IN ('LONDRINA','MARINGA');
  SELECT id FROM cophir cfv
    WHERE my_aggregate_function_ii(cfv.fv, 2, centers)<=10;
  ...
END;

-- Criação de um índice por similaridade agregada
CREATE INDEX cophir_my_weighted_manhattan_ix
 ON cophir(fv)
 USING slim_mass my_aggregate_function_ii;
```

4.2.2 Representação de Operadores por Similaridade Usando a SQL Padrão

Esta seção descreve como os alguns dos principais operadores por similaridade podem ser definidos na SQL padrão utilizando a representação de funções de distância proposta.

4.2.2.1 Consultas por Vizinhança

Esta seção ilustra como o operador k-NN pode ser representado, considerando a representação de FD definido na Seção 4.3. Por um lado, existem implementações em trabalhos anteriores que representam este operador através de operadores associados a UDFs (como o módulo `FMI-SiR` — Seção 3.6), como exemplificado a seguir (retorne as `5` imagens mais similares à imagem com `id=2`, considerando os vetores em `fv`):

```
SELECT image
FROM cophir WHERE manhattan_kNN(fv,
(SELECT fv FROM cophir WHERE id=2)) <= 5;
```

Embora seja uma representação utilizada em trabalhos anteriores, uma função não é apropriada para representar o operador k-NN porque a natureza intrínseca desse operador não pode ser expressa por ela. Pelo contrário, esta representação rompe com o

modelo da linguagem SQL, pois impõe uma condição de rankeamento na cláusula WHERE ao invés de uma condição lógica.

Por outro, este trabalho apresenta formas adequadas de representá-lo através de funções analíticas. Consultas por vizinhança podem ser representadas utilizando opções baseadas em ranqueamento documentadas no padrão SQL. O comando `FETCH FIRST k ROWS ONLY` é capaz de representar a Seleção por Vizinhança, enquanto as funções de janela (*window functions*) baseadas em ranqueamento `ROW_NUMBER` e `RANK`[2] podem ser usadas para representar tanto a Seleção quanto a Junção por Vizinhança. Os dois exemplos apresentados a seguir se beneficiam desses comandos para representar a Seleção por Vizinhança citada anteriormente (as `5` imagens mais similares à imagem com `id=2`, considerando os vetores em `fv`) e uma Junção por Vizinhança (para cada imagem da tabela `london`, concatene-a com cada uma das `5` imagens da tabela `cophir` mais similares a ela).

```
-- Seleção kNN
SELECT id FROM cophir
ORDER BY my_weighted_manhattan(fv,
 (SELECT fv FROM cophir WHERE id=2))
FETCH FIRST 5 ROWS ONLY;

-- Junção kNN
SELECT l_img, c_img FROM (
 SELECT l.image AS l_img, c.image AS c_img
  ROW_NUMBER() OVER (PARTITION BY l.fv ORDER BY
  my_weighted_manhattan(l.fv, c.fv)) AS rn
 FROM london l, cophir c)
WHERE rn <= 5;
```

Na execução convencional dessas expressões, os elementos são ordenados e depois os k primeiros elementos são recuperados[3]. Esta sequência de operações corresponde à execução sequencial das consultas por similaridade indicadas. Contudo, como as ordenações são baseadas em uma função de distância, o otimizador pode traduzir as instruções para os operadores por similaridade correspondentes (Seleção k-NN e Junção k-NN) e utilizar algoritmos e estruturas de dados que resultem em uma execução mais eficiente. Observe-se que, no exemplo da Junção por Vizinhança, se houvesse outras condições de filtragem na subconsulta seriam aplicadas antes da Junção k-NN, ao passo que condições de filtragem na consulta externa seriam aplicadas depois da junção.

As duas representações de consultas apresentadas permitem que o processador de consultas execute o respectivo operador por similaridade de forma correta, pois elas

[2] A função `RANK` produz o mesmo efeito do modificador `WITH TIES` da construção sintática `FETCH FIRST`.

[3] Normalmente, não é uma ordenação completa, pois só é preciso manter os k primeiros elementos ordenados a cada iteração.

representam a natureza intrínseca do mesmo. Com isso o SGBDS pode respeitar as regras de equivalência das consultas, que podem ser diferentes dos operadores tradicionais (conforme apresentado na Subsubseção 2.3.2.1).

4.2.2.2 Consulta por Vizinhança Reversa

Para ilustrar o poder de representação destas construções sintáticas da SQL padrão, a consulta a seguir é uma Consulta por Vizinhança Reversa. A Rk-NNq é composta por três operações, as duas primeiras correspondentes à $\bowtie_{k\text{NN}}$ e o terceiro filtra da $\bowtie_{k\text{NN}}$ com o elemento da busca, que no exemplo abaixo é identificado com o `id` 45:

```
-- kNN Reversos (RkNNq)
SELECT c_image FROM (
 SELECT c.image AS c_image,
  center.id AS center_id,
  c.fv AS c_fv, center.fv as center_fv,
  ROW_NUMBER() OVER (PARTITION BY c.fv
  ORDER BY my_weighted_manhattan(c.fv,
   center.fv)) AS rn
 FROM cophir c, london center
 WHERE c.id != center.id)
WHERE rn <= 1 AND center_id=2;
```

Nesta consulta, primeiro identifica-se o vizinho mais próximo de cada elemento da tabela com *alias* `c` (subconsulta + `rn <= 1`). Em seguida, identifica-se quais elementos da tabela com *alias* `c` têm como seu vizinho mais próximo o elemento de consulta (`center_id=2`).

4.2.2.3 Consulta Pelos Pares Mais Próximos

O raciocínio para construir o operador de Junção por pares de vizinhança é similar ao usado na construção de Seleção por vizinhança reversa, *i.e.*, o operador de Junção por pares de vizinhança é na verdade composto por duas duas operações: **ordene os pares de elementos** e **recupere os k pares mais próximos**. Portanto, encontrar os k pares mais próximos é uma junção que pode ser representada pelos comandos `partition by ... over`, como ilustrado abaixo.

```
-- Junção por pares de vizinhança
SELECT oiid
FROM ( SELECT out.id AS oiid, inn.id AS iiid,
ROW_NUMBER() OVER ( ORDER BY
manhattan_distance(inn.fv, out.fv)) AS rn
```

```
FROM cophir inn,london out) where rn <= 4;
```

4.3 Armazenamento de Vetores de Características em SGBDR

Neste trabalho é considerado um conjunto consistente de tipos de dados previstos no SQL padrão. São eles: *binário*, *relacional*, *semiestruturado* e *objeto-relacional* (OR). A avaliação se dá quanto ao respectivo desempenho e quanto a possibilidade de se representar as funções de distância de um centro (tanto simples quanto composta) no modelo apresentado na Subseção 4.2.1.

4.3.1 Binário

O padrão SQL prevê tipos de dados que armazenam cadeias de caracteres em atributos binários - o BINARY(tamanho), o VARBINARY(tamanho máximo) e o BLOB. O primeiro permite o armazenamento de uma cadeia de caracteres de tamanho fixo. O segundo é uma cadeia de caracteres de tamanho variável na qual o usuário determina o tamanho máximo do atributo no momento de criação da tabela. Por fim, o tipo BLOB é um tipo de objeto LOB (**L**arge **OB**ject), os quais foram desenvolvidos para armazenar dados não estruturados.

Ao analisar mais profundamente os tipos BINARY e VARBINARY têm comportamento semelhante, respectivamente, aos CHAR e ao VARCHAR.

Os objetos LOB estão incluídos na OCCI (**O**racle **C**++ **C**all **I**nterface) e podem ser usados tanto em processos externos ao Banco de Dados quanto em processos internos. Internamente no Banco de Dados este atributo é armazenado como um destes três tipos de dados: CLOB (**C**haracter **L**arge **OB**ject) e BLOB (**B**inary **L**arge **OB**ject) ou NCLOB (**N**ational **C**haracter **L**arge **OB**ject).

Os objetos LOB são compostos pelo LOB *value* - que armazena informações do tamanho, do valor do atributo, entre outros - e o LOB *locator* - que é um ponteiro responsável por indicar a posição onde o LOB *value* está armazenado, sendo que internamente, objetos LOB usam uma semântica de cópia - aqueles que são inseridos ou atualizados de outras colunas também do tipo LOB recebem cópia dos dados de origem (tando o LOB *value* quanto o LOB *locator* são copiados). O LOB *locator* é armazenado junto dos outros atributos da tabela, assim como os LOB *values* menores do que 4.000 bytes, quando são maiores os LOB *values* são armazenados externamente a tabela do Banco de Dados.

O BLOB é mais flexível que os outros dois, pois não impõe limitações de tamanho ao vetor de características. Por este motivo, o tipo BLOB representa o modelo de armazenamento binário neste trabalho.

O armazenamento binário consiste em utilizar atributos do tipo BLOB para ar-

mazenar FVs e FDs implementadas como funções externas. Dessa forma, o SGBDR não tem ciência das informações armazenadas no FV, portanto não oferece um suporte mais elaborado, atuando quase como um repositório de arquivos. Os meta-dados do vetor de características devem ser inclusos no próprio FV (Figura 17), para que as funções externas saibam como tratá-lo. A representação de consultas é idêntica ao apresentado na Seção 4.2. Não obstante, o armazenamento e a recuperação desses vetores de características podem ser conduzidos por linguagens externas, inicialmente mais eficientes do que as baseadas no padrão SQL/PSM, embora exista uma sobrecarga ao invocá-las.

4.3.2 Relacional

O armazenamento relacional consiste em criar uma tabela para cada FV do banco de dados, referenciando a tabela que armazena o respectivo dado complexo. Essa abordagem permite que o usuário manipule as características armazenadas por meio de comandos DML que apresentam grande flexibilidade. Contudo, a passagem de parâmetros para as funções de distância exige informar cada atributo de cada FV individualmente, o que, além da sintaxe carregada, exige criar FDs específicas para cada dimensionalidade, como ilustrado a seguir.

```
my_weighted_manhattan(a1, ..., an, b1, ..., bn)
```

Uma alternativa poderia ser indicar apenas a relação corresponde a cada FV, mas esta alternativa não é genérica pois mantém as funções de distância amarradas ao esquema do banco. Outra opção seria passar cursores como parâmetros, mas essa opção perde totalmente a verificação de tipos, que é um recurso nativo e importante oferecido pelos SGBDRs. É possível, contudo, fornecer um procedimento ao usuário que cria a função de distância com o número de dimensões desejado, o qual é exemplificado abaixo.

```
RDBMS_SIM.GEN_SIM_REL_FUNCTION (
base_func  => "L1", func_name => 'L1_128', dimensions => 128);
```

Além disso, a escrita de consultas com dados de alta dimensionalidade é enfadonha. Isto pode ser contornado com o auxílio de um procedimento fornecido pelo SGBDS, ilustrado abaixo. Repare que procedimentos análogos podem ser criados para qualquer operador por similaridade.

```
--Escreve uma Rq para o usuário com uma tabela de 128 dimensões.
RDBMS_SIM.GEN_SIM_REL_Rq (
func_name => 'L1_128', table_name => 'cophirfv_128',
table_center => 'cophirfv_128', center_id => 38);
```

```
--Escreve uma kNNq para o usuário com uma tabela de 128 dimensões.
RDBMS_SIM.GEN_SIM_REL_kNNq (
func_name => 'L1_128', table_name => 'cophirfv_128',
table_center => 'cophirfv_128', center_id => 38);
```

Outra limitação da abordagem relacional é para armazenar vetores de características Adimensionais, pois exige que a tabela tenha o mesmo número de atributos do maior FV armazenado, o que pode gerar muitos parâmetros nulos. Além disso, o esquema da tabela tem que ser atualizado a cada elemento com mais características que o maior armazenado até então.

As tabelas do modelo relacional são capazes de armazenar vetores de características simples tanto os Dimensionais quanto os Adimensionais. Sendo que no segundo caso, cria-se uma tabela com o número máximo de características, com colunas que podem ser nulas. Além disso, é possível utilizar uma solução similar a do `SIREN` (Seção 3.1): por meio de dados do tipo `PARTICULATE` são armazenadas as características em uma tabela do sistema. Por fim, para armazenar os vetores de características compostos se faz necessário o uso do relacionamento entre tabelas, onde cada uma delas representa um componente.

4.3.3 Semiestruturado

Os armazenamentos semiestruturados mais utilizado são o XML (*e**X**tensible **M**arkup **L**anguage*) e o JSON (***J**ava**S**cript **O**bject **N**otation*), dentre eles o que possui suporte mais maduro em SGBDRs é o primeiro. Por isto o XML é o padrão utilizado neste trabalho.

O modelo semi-estruturado é o semanticamente mais próximo a natureza diversificada dos dados complexos e permite a representação de consultas como apresentado na Seção 4.2. Nesse estudo adotamos a representação `XML` para o tipo semi-estruturado, pois a implementação dessa estrutura se encontra madura em SGBDRs comerciais, em particular com amplo suporte no Oracle. Assim, torna-se possível representar todos os tipos de vetores de características apresentados por meio de *XML Schemas* e manipular as características armazenadas através dos comandos `DML`, embora isso seja feito via um *parser*, o que introduz um limitador de desempenho. Adicionalmente, os SGBDRs permitem que os dados sejam armazenados como estruturas binárias (que chamaremos de tipo XML binário), objeto-relacionais (que chamaremos de XML e XML OR, alternadamente). Abaixo são representados os diferentes tipos de vetores de características através do XML Schema.

Vetor de características simples dimensional homogêneo:

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="FeatureVector">
```

```
<xs:complexType><xs:sequence>
<xs:element name="feature" type="xs:integer"
minOccurs="282" maxOccurs="282"/>
</xs:sequence></xs:complexType>
</xs:element></xs:schema>
```

Vetor de características simples adimensional homogêneo:

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="FeatureVector">
<xs:complexType><xs:sequence>
<xs:element name="feature" type="xs:string"
minOccurs="1" maxOccurs="282"/>
</xs:sequence></xs:complexType>
</xs:element></xs:schema>
```

Vetor de características composto dimensional homogêneo (dinâmico):

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="FeatureVector">
<xs:complexType>
<xs:sequence>
<xs:element name="Component">
<xs:complexType><xs:sequence><xs:element
name="feature" type="xs:integer"
minOccurs="282" maxOccurs="282"/>
</xs:sequence></xs:complexType></xs:element>
</xs:sequence></xs:complexType></xs:element>
</xs:schema>
```

Vetor de características composto adimensional heterogêneo (dinâmico):

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="FeatureVector">
<xs:complexType>
<xs:sequence>
<xs:element name="Component_Integer">
<xs:complexType><xs:sequence><xs:element
```

```
name="feature" type="xs:integer"
minOccurs="282" maxOccurs="282"/>
</xs:sequence></xs:complexType></xs:element>
<xs:element name="Component_String">
<xs:complexType><xs:sequence><xs:element
name="feature" type="xs:string"
minOccurs="1" maxOccurs="32"/>
</xs:sequence></xs:complexType></xs:element>
</xs:sequence></xs:complexType></xs:element>
</xs:schema>
</xs:schema>
```

Por fim, este trabalho utiliza objetos DOM no desenvolvimento de FDs. Apesar de se ser possível a utilização de consultas que fazem uso do XPATH para o desenvolvimento das FDs, elas se mostraram menos eficientes em análises preliminares se comparadas as primeiras. Para ilustrar o uso de XPATH no desenvolvimento de FDs, abaixo é apresentado o código da função manhattan desenvolvida através deste tipo de consulta.

```
create or replace FUNCTION XML_manhatan_distance(
  innXML XMLType,
  outXML XMLType
) RETURN number
IS
  total NUMBER;
BEGIN
  SELECT SUM( ABS(oFV.feature - iFV.feature) )
  INTO   total
  FROM   XMLTABLE(
           '//FeatureVector/feature'
           PASSING outXML
           COLUMNS rn       FOR ORDINALITY,
                   feature  NUMBER  PATH '.'
         ) oFV
         INNER JOIN
         XMLTABLE(
           '//FeatureVector/feature'
           PASSING innXML
           COLUMNS rn       FOR ORDINALITY,
                   feature  NUMBER  PATH '.'
         ) ifv
         ON ( oFV.rn = iFV.rn );
```

```
  RETURN total;
END;
/
```

4.3.4 Objeto-relacional

O SQL padrão possibilita que o usuário defina os próprios tipos através do comando CREATE TYPE, os quais são compostos por um conjunto de atributos de diferentes tipos. Isto permite a extensão do modelo relacional para que o mesmo possa suportar a criação de objetos, dando origem ao modelo objeto-relacional. Muitas ferramentas de gerenciamento de Banco de Dados suportam este novo modelo (por exemplo: Oracle, SQLite, PostgreSQL e SQLServer), por isso são chamadas SGBDR.

A partir destes tipos definidos pelo usuário é possível criar colunas que se comportam como tabelas - desenvolve-se o que é chamado de relação aninhada (no inglês *nested relation*), ou seja, estrutura-se uma tabela dentro de outra tabela. No SQL padrão, as colunas que têm uma relação aninhada junto a tabela que a compõem são do tipo MULTISET, SET e LIST (chamado também de ARRAY).

Atributos do tipo MULTISET e do tipo SET são tabelas ou coleções de linhas, sendo que o conteúdo dos dados armazenados no primeiro tipo pode ser duplicado, enquanto no segundo isso não ocorre. Já os atributos do tipo ARRAY são diferentes das tabelas apenas em um ponto: seu conteúdo é ordenado.

No Oracle existem três tipos de coleção de dados no padrão SQL e implementados no SGBDR Oracle: listas associadas (*associative arrays*), tabelas aninhadas e ARRAYS (implementados no Oracle como VARARRAYS) - os dois últimos podem ser armazenados de forma persistentes e por isso serão explorados mais a fundo nesta Seção.

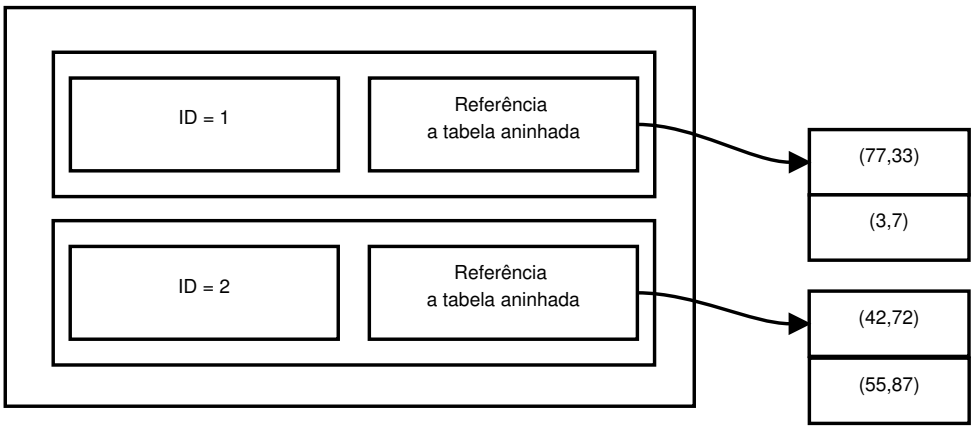

Figura 21 – Armazenamento de uma tabela aninhada no Oracle

Através dos UDTs é possível implementar as relações aninhadas, ilustradas na Figura 21. Nesta ferramenta é possível armazenar um UDT como uma tabela aninhada ou

como um VARARRAY. As tabelas aninhadas no Oracle são a implementação do MULTISET apresentado anteriormente. Já a coleção do tipo ARRAY é implementado no Oracle com o nome de VARARRAY.

Abaixo é ilustrado como criar um tipo de vetor de características composto predefinido através de uma tabela aninhada. Depois de criar o tipo, é ilustrado como criar a tabela e inserir as informações dos respectivos vetores de características na mesma.

```
--Criação do tipo que representa o vetor de características composto predefinido
CREATE OR REPLACE TYPE featureVector AS OBJECT (
 YACCoeff5_a           integer,
 YACCoeff5_b           integer);
/

--Criação de uma tabela aninhada no Oracle
CREATE TABLE cophir (
 id                    integer,
 fv                    featureVector)
 NESTED TABLE fv STORE AS fv_tab;
/

--Inserção de dados em uma tabela aninhada no Oracle
INSERT INTO cophir VALUES(1,featureVector(fv(77,33),
fv(3,7)));
INSERT INTO cophir VALUES(2,featureVector(fv(42,72),
fv(55,87)));
```

Depois de criar a tabela aninhada, o usuário pode inserir dados na mesma e após a inserção, as informações podem ser acessadas, conforme ilustrado a seguir.

```
--Consulta filtra todo conteúdo da tabela aninhada contida em coluna
select fv FROM cophir;

--Consulta filtra coluna específica da tabela aninhada contida em coluna
select t.YACCoeff5_a
from   cophir cfv,
table (cfv.fv) t;
```

Outro tipo de coleção de dados implementado no Oracle é o VARARRAY, o qual é definido através de um número máximo de elementos e deve ser acessado sequencialmente. Além disso, diferente das tabelas aninhadas exploradas nos parágrafos anteriores que são

armazenadas de forma esparsa, os dados do tipo VARARRAY são armazenados de forma densa (i.e. de forma consecutiva).

O VARARRAY deve ser usado quando o volume de informações a serem armazenados por atributo não passar de 4KB - caso a quantidade de dados a ser armazenada seja maior, o SGBD da Oracle armazena as informações fora da tabela, o que impacta no desempenho das operações que envolvem o dado atributo. O usuário precisa ter em mente que dada qualquer operação envolvendo o VARARRAY, as informações armazenadas têm todo o seu conteúdo carregado na memória de uma vez.

O ponto negativo de se usar o tipo VARARRAY é a falta de flexibilidade na sua manutenção. Por exemplo, não é possível atualizar elementos específicos através de comandos SQL, o que deve ser feito apenas através de procedimentos PL/SQL ou através de funções externas.

A criação da tabela e inserção dos dados se dá como segue neste caso.

```
--Criação da tabela aninhada
CREATE TABLE cophir (
 id                    integer,
 fv                    integerComponent2)

--Inserção de dados no VARARRAY
 INSERT INTO cophir VALUES (
 1, integerComponent2(33,77));
  INSERT INTO cophir VALUES (
 2, integerComponent2(44,55));
 ...

--Consulta ao conteúdo da tabela aninhada
 SELECT fv
 FROM cophir;
```

O tipo de armazenamento Objeto-relacional (OR) é implementado por meio de UDTs, o que permite definir hierarquias de tipo e representar funções de distância como apresentado na Seção 4.2. No armazenamento OR, há três opções para a representação dos vetores de características, a saber: *(i)* cada característica pode ser armazenada em uma coluna, *(ii)* armazenamento em tabelas aninhadas e *(iii)* armazenamento em *arrays*. Como as duas primeiras opções compartilham limitações com a alternativa relacional, a terceira opção foi a selecionada nesse estudo para representar o armazenamento de vetores de características como OR. *Arrays* permitem armazenar todos os tipos de FV, incluindo FVs Adimensionais, e não requer implementar funções de distâncias para diferentes dimensionalidades. Além disso, o modelo OR permite representar FVs Compostos como

objetos complexos, o que facilita a interpretação. A seguir, são ilustrados exemplos de declaração de dois FVs Simples e de um FV Composto Heterogêneo[4].

```
--Vetor de características simples homogêneo
--Características do tipo inteiro.
CREATE TYPE INT_FV AS
     VARRAY(64) OF INTEGER;

--FV simples homogêneo
--Características do tipo texto.
CREATE TYPE CHAR_FV AS
     VARRAY(128) OF VARCHAR(45);

--FV composto homogêneo
CREATE TYPE dynamic_composed_fv AS VARRAY(300) OF INT_FV;

--FV composto heterogêneo predefinido
CREATE TYPE static_composed_fv AS OBJECT
( scalableColor       INT_FV,
  colorStructure      INT_FV,
  tags         CHAR_FV );
```

4.4 Discussão: Armazenamento e Recuperação de Dados Complexos em SQL Padrão

Todos os tipos de dados analisados para o armazenamento de vetores de características possuem suas respectivas vantagens e desvantagens. O modelo de armazenamento binário apresenta flexibilidade na implementação de todos os tipos de vetores de características, porém a manipulação é via funções externas ao SGBDR. O modelo de armazenamento relacional é o mais conhecido, mas requer a criação de novas FDs para cada dimensionalidade e possui uma sintaxe carregada para passagem de elementos para funções de distância. O modelo de armazenamento semi-estruturado é o mais flexível na representação dos diversos tipos de vetores de características, contudo exige o uso de *parsers*. Por fim, o modelo de armazenamento objeto-relacional é o melhor modelo quando considera-se flexibilidade e usabilidade. Além de ter suporte amplo em SGBDs comerciais, possui sintaxe simples para passagem de elementos para funções de distância, permite a melhor verificação de tipos e é capaz de representar todos os tipos de vetores de características de forma elegante.

[4] No Oracle define-se o número máximo de características presentes no *array*, mas só as posições efetivamente ocupadas são armazenadas. A FD deve validar se o mesmo é dimensional, se necessário.

5 IMPLEMENTAÇÃO E AVALIAÇÃO DAS ESTRATÉGIAS DE ARMAZENAMENTO E RECUPERAÇÃO DE DADOS COMPLEXOS

Todas as variações discutidas no capítulo anterior foram implementadas no módulo *e*FMI-SiR, que é uma extensão do FMI-SiR [8], acoplado ao SGBDR Oracle, com o objetivo de verificar a aplicabilidade de cada modelo, identificar suas vantagens e limitações para a representação dos diferentes recursos, e avaliar empiricamente o desempenho de cada um deles. O *e*FMI-SiR é capaz de representar os seguintes operadores: seleção e junção por abrangência (conforme apresentado na Subseção 4.2.1), seleção e junção k-NN, k-CN, Rk-NNq, ARq, k-ANNq e os operadores por agrupamento por similaridade, tanto unidimensionais quanto mutidimensionais, todos eles apresentados na Subseção 4.2.2.

Este capítulo apresenta: *(i)* um conjunto consistente de experimentos acerca da inserção, armazenamento e recuperação de dados complexos (Seção 5.1); e *(ii)* uma discussão referente ao armazenamento e recuperação de dados complexos nos modelos implementados no *e*FMI-SiR (Seção 5.2).

5.1 Experimentos

Esta Seção descreve uma análise experimental conduzida utilizando-se o módulo *e*FMI-SiR. Os experimentos realizados neste trabalho avaliaram o armazenamento e a recuperação de dados métricos em quatro modelos de armazenamento previstos na SQL padrão, implementados no Oracle: *relacional*, *objeto-relacional* (representado pelo armazenamento em *arrays*), *semiestruturado* (armazenamento XML OR e XML binário) e *binário* (representado por arquivos BLOB presentes no Oracle).

A seguir é feita uma descrição dos experimentos e a metodologia utilizada (Subseção 5.1.1). As seções a seguintes apresentam resultados dos experimentos divididos em três dimensões, em que foram avaliados os desempenhos das diferentes abordagens de representação e armazenamento de dados complexos em SQL padrão implementadas no *e*FMI-SiR. Elas são: recuperação (Subseção 5.1.2), inserção (Subseção 5.1.3) e consumo de espaço em disco (Subseção 5.1.4).

5.1.1 Descrição dos Experimentos

Foram realizados dois conjuntos de experimentos. No conjunto de experimentos i foram avaliados os vetores de características dimensionais. Foram utilizados um milhão de vetores de características de 282 dimensões provenientes do *Content-based Photo Image*

Retrieval (`CoPhIR`) [88] [1], que é uma coleção de dados provenientes de imagens do Flickr[2].

Na recuperação dos dados foram consideradas duas FDs computacionalmente mais baratas provenientes da família Minkowski e outra mais caras, respectivamente: L_1, L_2 e Mahalanobis. Na execução de experimentos, os vetores de características foram armazenados considerando-se um subconjunto de suas dimensões: 12 dimensões, 64 dimensões, 128 dimensões e 282 dimensões. Os FVs de 12 dimensões representam o *color layout* (i.e. a distribuição espacial das cores) das imagens, os de 64 dimensões representam o *scalable color* (i.e. um histograma derivado do descritor da imagem), os de 128 dimensões representam os scalable color e color structure (i.e., FV resultado de uma segmentação da imagem), 282 dimensões possuem todas as características — Scalable Color, Color Layout, Color Structure, Edge Histogram (histograma largamente utilizado na detecção de formas), Homogeneous Texture (que representa padrões idealizados).

O conjunto de experimentos *ii* avaliou vetores de características adimensionais. Foi utilizado um conjunto de resumos do *NSF Research Award Abstracts*[3]. O texto considerado foi até o primeiro ponto final de cada resumo. Os textos passaram por um pré-processamento que removeu espaços duplicados, a pontuação e os números e, por fim, passaram por um procedimento de *stemming*[4]. Cada palavra do resumo consiste em uma característica do FV e foram utilizados FVs que contêm de 2 a 32 palavras. O número total de FVs adimensionais utilizados é de 86569.

O SGBDR utilizado nos experimentos é o Oracle 12c, o qual apresenta uma implementação madura dos diferentes modelos de dados previstos na SQL padrão e possui os recursos necessários para a implementação de todos os modelos de dados considerados. O desempenho global dos modelos pode ser semelhante para outros SGBDRs, pois o comportamento dos mesmos se dá devido à natureza intrínseca dos tipos de dados utilizados, conforme explanado na Seção 5.2. Além disso, a recuperação de dados complexos considerou a avaliação do desempenho apenas da Rq executada por meio de uma varredura sequencial, pois o desempenho de execução de outros operadores para os diferentes modelos de dados é uma função do tempo de execução da Rq.

A execução dos testes se deu em um computador com as seguintes configurações: processador Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz, cache de 15MB, 8GB de memória RAM, HD modelo ST1000NM0033 SATA 3.0, com taxa de transferência de 6.0 Gb/s, e GNU/Linux Ubuntu 14.04.

[1] `http://cophir.isti.cnr.it/`

[2] O Flickr é uma rede social feito para compartilhar imagens, sendo considerado um *fotoblog*.

[3] `http://archive.ics.uci.edu/ml/datasets/NSF+Research+Award+Abstracts+1990-2003`

[4] *Stemming* consiste em reduzir as palavras para o seu radical (não necessariamente a raiz morfológica), para que as palavras relacionadas possam ser identificadas.

5.1.2 Desempenho da recuperação de dados

No conjunto de experimentos *i* foram utilizados 100 centros de buscas em todos os conjuntos de teste que não pertencem às tabelas pesquisadas. Nos experimentos foi executada uma busca por abrangência para cada centro de busca, as quais filtram entre 0,3% e 1% dos elementos consultados.

Conforme pode ser observado na Figura 22, o desempenho da recuperação de dados métricos armazenados no modelo relacional é superior na maior parte dos casos e é sempre acompanhado de perto do armazenamento em VArray. Ambos são superados pelo armazenamento BLOB no caso de vetores de características de alta dimensionalidade, recuperados pela FD mahalanobis (mais custosa). A recuperação de vetores de características de 282 dimensões pela FD mahalanobis armazenados em BLOB precisam de 7% do tempo necessário pelo relacional. Por outro lado, o armazenamento XML (OR e binário) apresenta o pior desempenho em todas as situações. A recuperação de dados complexos no modelo relacional necessita de no máximo 3%, 11% e 9% do tempo necessário para o armazenamento XML OR por meio das FD L_1, L_2 e mahalanobis, respectivamente.

Outro experimento do conjunto de experimentos *i*, executou consultas por abrangência em tabelas de 10.000 tuplas de diferentes dimensionalidades, cujo resultado é apresentado na Figura 23. Este segundo experimento ilustra como o desempenho do armazenamento binário não é afetado pela mudança na dimensionalidade.

No conjunto de experimentos *ii* foram executadas buscas utilizando igualmente 100 centros em todos os conjuntos de teste que não pertencem às tabelas pesquisadas. Seus resultados são apresentados na Figura 26(a). Neste experimento o armazenamento objeto-relacional superou o relacional, pois as funções de distância do último modelo verificam a cada palavra se o parâmetro é nulo, o que infere custo adicional - a recuperação de dados adimensionais do primeiro modelo precisou de 75% a 80% do tempo utilizado pelo segundo. Adicionalmente, o armazenamento binário superou o semi-estruturado, que apresenta o pior desempenho. O modelo objeto-relacional preciso entre 1,12% a 1,85% do tempo utilizado pelo modelo XML para recuperar as informações.

5.1.3 Desempenho da inserção de dados

Neste experimento foram feitas vinte inserções de 100, 1.000 e 10.000 tuplas em colunas dos tipos de dados analisados. As médias do tempo necessário para a inserção destes dados são ilustradas nas Figura 26(b) e Figura 24.

No conjunto de experimentos *i*, o desempenho da inserção de dados é em geral mais eficiente no armazenamento VArray e menos eficiente no BLOB. A inserção de vetores de características no VArray necessitou de 41% a 42% do tempo necessário para a inserção de dados no BLOB, sendo o tempo de inserção em ambos os modelos estável se comparado

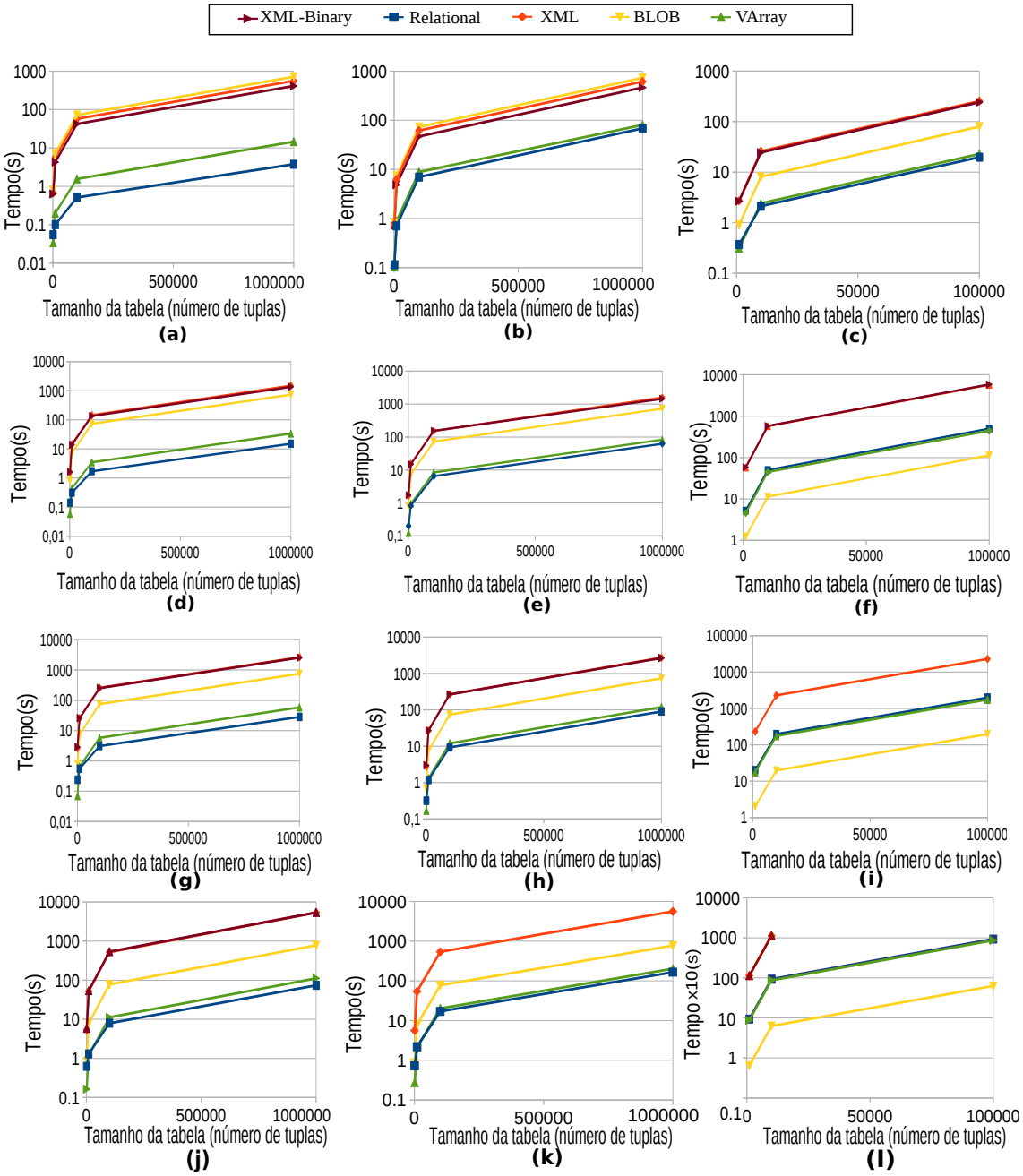

Figura 22 – Busca por abrangência de até 1% das tuplas contidas nas tabelas através, respectivamente, das FDs L_1, L_2 e mahalanobis. Nos gráficos é apresentada a variação do tempo da consulta de acordo com o aumento do número de linhas da tabela de entrada. (a-c) Vetores de características de 12 dimensões, (d-f) vetores de características de 64 dimensões, (g-i) vetores de características de 128 dimensões e (j-l) vetores de características de 282 dimensões.

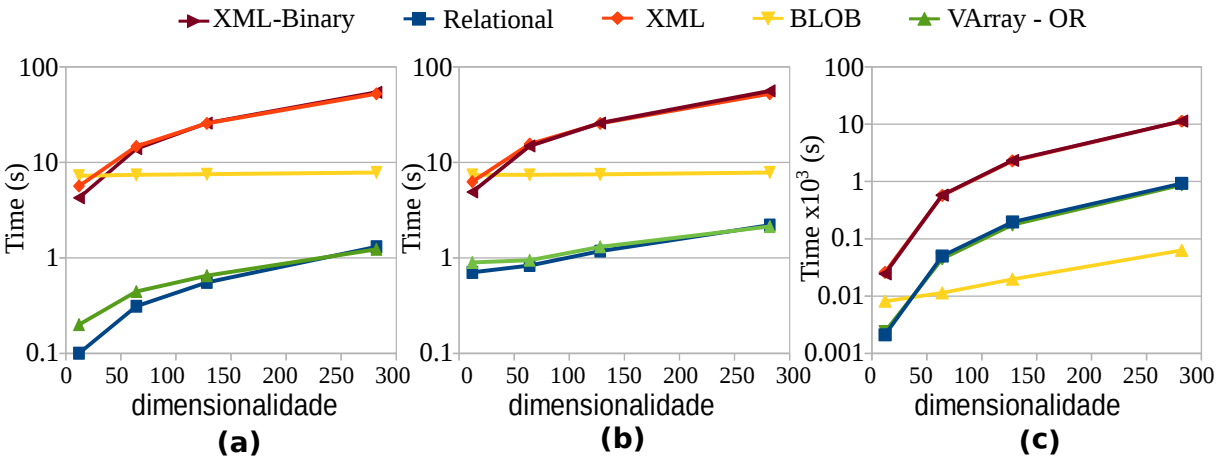

Figura 23 – Busca por abrangência de até 1% das tuplas contidas nas tabelas de 10.000 tuplas através, respectivamente, das FD L_1, L_2 e mahalanobis.

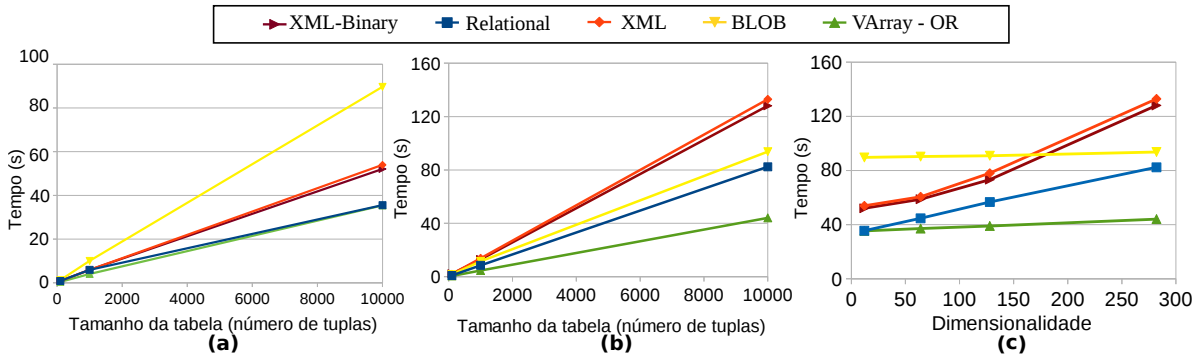

Figura 24 – Inserção de vetores de características de (a) 12 e (b) 282 dimensões. (c) Inserção de 10.000 vetores de características.

com as demais formas de armazenamento. O armazenamento no modelo relacional e o armazenamento em XML (OR e binário) figuram respectivamente em segundo e terceiro. Estes últimos têm seu desempenho prejudicado com o aumento da dimensionalidade. Para armazenar dados de 12 e 282 dimensões no VArray é necessário, respectivamente, 68% e 54% do tempo necessário para o relacional.

No conjunto de experimentos *ii*, a inserção de dados é em geral mais eficiente no modelo relacional. O modelo relacional precisou de 3% a 12% do tempo necessário para inserir os vetores de características no modelo binário, o qual apresentou o pior desempenho neste caso.

5.1.4 Consumo de espaço em disco

Neste experimento foi feita uma consulta através dos metadados do Oracle que forneceu o espaço de disco ocupado por tabelas que contêm vetores de características. No

conjunto de experimentos *i*, as tabelas analisadas possuem vetores de características de diferentes dimensionalidades e o número de tuplas armazenadas é variável (Figura 25). No conjunto de experimentos *ii*, foi avaliado o espaço do disco ocupado por tabelas de 1000, 10000 e 86569 tuplas (Figura 26(c)).

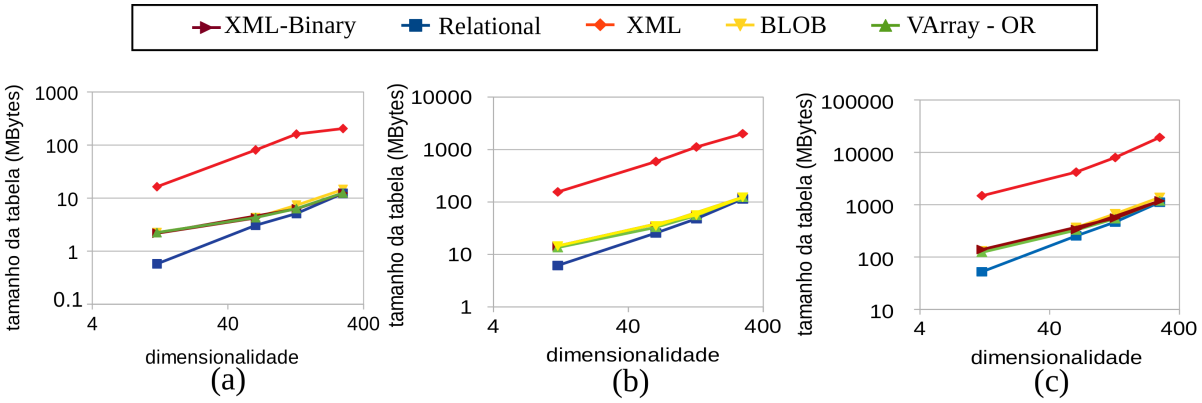

Figura 25 – Espaço em disco ocupado por vetores de características, variando-se o número de características. Tabelas de (a) 10.000, (b) 100.000 e (c) 1.000.000 tuplas.

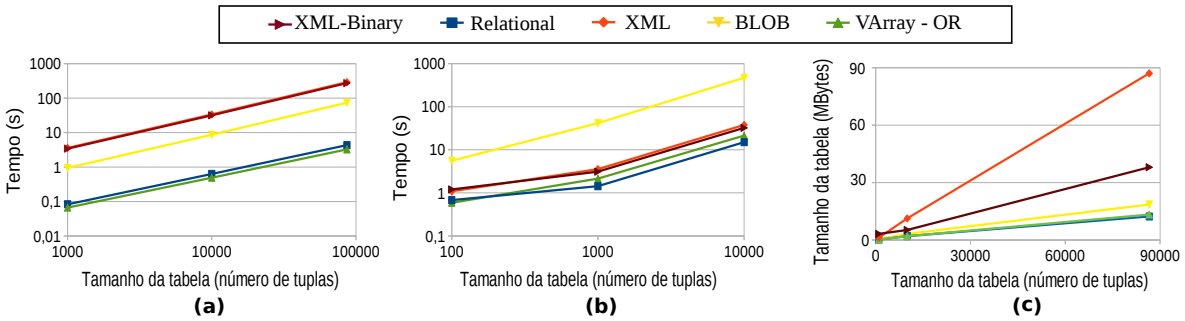

Figura 26 – Desempenho das tabelas que armazenam FVs adimensionais: (a) recuperação de até 1% dos elementos da tabela utilizando a distância Jaccard, (b) inserção de vetores de características adimensionais e (c) consumo de espaço em disco para armazenamento de vetores de características adimensionais.

No conjunto de experimentos *i*, o armazenamento relacional é o mais eficiente dentre os analisados, em termos de consumo de espaço em disco (i.e., baixo consumo de espaço). Ele é seguido pelo BLOB, XML binário e VArray que têm aproximadamente o mesmo desempenho, pois todos são armazenados da mesma forma - em arquivos binários. Armazenar vetores de características em tabelas relacionais demanda até 57% menos espaço do que em VArrays. O tipo de dado que mais demandou espaço em disco foi o XML OR, o qual necessitou entre 16x e 28x do espaço necessário para o armazenamento relacional.

Tabela 2 – Armazenamento e recuperação de dados métricos - análise do tipo de dados

Tipo do dado	Desempenho da recuperação (FD de baixo custo)	Desempenho da recuperação (FD de alto custo)	Sintaxe carregada para FDs
Relacional	Alto	Médio	Sim
VArray	Alto	Médio	Não
XML	Baixo	Baixo	Não
BLOB	Baixo	Alto	Não

Tipo do dado	Desempenho da inserção	Consumo de espaço em disco	Manipulação dos dados
Relacional	Alto	Baixo	DML
VArray	Alto	Baixo	DML e linguagem procedural
XML	Baixo	Baixo (binário) Alto (OR)	DML e SQL/XML
BLOB	Baixo	Baixo	Funções externas

Já no conjunto de experimentos *ii*, os modelos relacional e objeto-relacional tiveram desempenhos muito similares neste teste, enquanto que o armazenamento em atributos XML OR foi o que requisitou mais espaço em disco. O modelo relacional utilizou de 14% a 18% do espaço em disco gasto pelo XML OR.

5.2 Discussão

Neste capítulo foi apresentado como armazenar e recuperar dados complexos em SGBDR através de apenas uma alteração na SQL padrão. A implementação proposta é capaz de representar e executar os operadores físicos de um conjunto amplo de operadores por similaridade. Além disso, o desempenho de quatro modelos de representação de dados complexos foram avaliados de acordo com o seu desempenho referente a recuperação, inserção e armazenamento. Avaliação que é resumida na Tabela 2.

O modelo relacional apresenta o melhor desempenho em quase todas as situações avaliadas nos experimentos. Isso ocorre porque no armazenamento e na inserção não há nenhuma estrutura extra, apenas os valores das características que são inseridos no SGBDR e os respectivos metadados. Na recuperação de vetores de características o desempenho no geral é melhor para FV dimensionais porque o cálculo da distância é feito diretamente dos parâmetros passados. Para o caso de FV adimensionais o modelo relacional passa a apresentar o segundo melhor desempenho na recuperação de dados pois a FD precisa verificar o número de características de cada FV e faz isso através de uma condição que avalia se o valor de dada característica é nulo — quando encontra um valor nulo, finaliza os cálculos.

O modelo objeto-relacional, representado pelo armazenamento e recuperação de

FV em VARRAYs, apresenta um desempenho próximo do modelo relacional em todos os casos. Em geral, seu desempenho é ligeiramente inferior ao modelo relacional, sendo que a situação se inverte para o caso da recuperação de FV adimensionais.

O modelo semi-estruturado apresenta o pior desempenho nos experimentos em quase todas as situações, apesar de sua flexibilidade na representação de dados complexos. Os motivos para o baixo desempenho é que se faz necessário inserir e armazenar estruturas extras e devido à necessidade na manipulação de objetos usando um *parser* XML, o que consome recursos na execução das FDs.

O desempenho do modelo binário possui dois fatores que constrastam entre si e definem o desempenho observado. Por um lado ao invocar funções externas ao SGBDR implica em *overhead*. Por outro, as FD desenvolvidas por linguagem de terceira geração são mais eficientes. Por conta destas nuances, o desempenho deste modelo para FDs de baixo custo computacional é baixo e para FDs de alto custo computacional o desempenho é alto. Da mesma forma, observa-se que o desempenho para inserção de FVs aumenta com a dimensionalidade dos mesmos.

Assim, os experimentos mostram que na maior parte dos casos o modelo relacional tem um desempenho superior aos demais no armazenamento e recuperação de FVs dimensionais, o qual é acompanhado de perto do modelo objeto-relacional. Por outro lado, no armazenamento e recuperação de FVs adimensionais, o modelo objeto-relacional tem o melhor desempenho. Por fim, o modelo de armazenamento binário apresentou o melhor desempenho para as dimensionalidades mais altas quando foi considerada uma FD de custo computacional mais elevado.

6 CONCLUSÕES E TRABALHOS FUTUROS

Até onde sabemos, nenhuma avaliação da forma de representar e armazenar dados complexos utilizando padrões existentes foram realizadas anteriomente. Trabalhos anteriores não são focados na discussão de padrões existentes em linguagens de alto nível que permitem expressar consultas por similaridade com transparência para o processador de consultas de SGBDRs. Embora existam trabalhos na literatura que representam operadores por similaridade ora através de extensões a SQL ora utilizando a SQL padrão, eles não examinam se a respectiva solução de linguagem é adequada e satisfaz todos os requisitos esperados de um sistema de armazenamento e recuperação de dados.

Esta dissertação fez um levantamento dos sistemas projetados e implementados em estudos anteriores e apontou que todos eles possuem ao menos uma destas características, não apropriadas para um SGBDS: a representação dos operadores por similaridade é carregada, apenas um número limitado de operadores por similaridade é considerado, a possibilidade de reconhecimento do operador por similaridade físico por parte do SGBD não é abordada, um número considerável de extensões à linguagem SQL padrão é proposto, não é abrangente a vários tipos de vetores de características possíveis e o modelo de armazenamento escolhido não é baseado em uma análise aprofundada, sendo pouco eficientes e/ou inadequados em termos de representação de operações baseadas em similaridade.

Este estudo objetivou preencher estas lacunas fornecendo uma representação para as funções de distâncias que permite ao processador de consultas reconhecê-las e associá-las à operadores por similaridade físicos e também foi feita uma avaliação dos modelos de armazenamento existentes em SGBDRs, em termos de flexibilidade, eficiência e usabilidade.

6.1 Principais Contribuições

As contribuições desta dissertação podem ser sumarizadas conforme segue.

1. **Proposta de uma categorização de vetores de características e funções de distância.** Identificação de um conjunto reduzido de tipos diferentes de vetores de características e funções de distância para acomodar os diversos tipos existentes na literatura. As FDs e os FVs existentes na literatura foram categorizados neste trabalho, para que seja possível a respectiva representação e implementação adequada de todos. As primeiras foram categorizadas em: Simples, Compostas e Agregadas. Os FVs são categorizados de três formas diferentes: Dimensionais ou Adimensionais,

Simples ou Compostos, Homogêneos ou Heterogêneos.

2. **Proposta de uma representação abrangente para operadores por similaridade em SQL padrão.** Definição de um modelo de representação para todos os tipos de funções de distância presentes na literatura, capaz de representar os principais de operadores por similaridade e possibilitar a chamada do respectivo operador físico. Esta dissertação ilustrou como é possível representar a maioria dos operadores por similaridade encontrados na literatura através da SQL padrão, a saber: Rq, $\bowtie_{\mathrm{Rq}}$, kNN, Rk-NNq,$\bowtie_{k\mathrm{NN}}$, $\bowtie_{k\mathrm{CN}}$, ARq e k-ANNq. Além disso, foi introduzida uma forma de habilitar o SGBDS a reconhecer o bloco da consulta que representa os respectivos operadores físicos através da palavra reservada DISTANCE nos comandos que criam e manipulam as funções de distância, para diferenciá-las de outras UDFs.

3. **Implementação do módulo *e*`FMI-SiR` para análise da eficiência de diferentes estratégias de armazenamento de vetores de características.** Foram implementados a analisados quatro tipos de dados previstos na SQL padrão, quanto a representação e eficiência no armazenamento e recuperação de vetores de características: *binário*, *relacional*, *objeto-relacional* e *semiestruturado*. A partir da representação dos diferentes tipos de FDs e FVs e de um conjunto representativo de experimentos, os quatro modelos de armazenamento foram analisados. Os resultados experimentais mostraram que o desempenho dos modelos de armazenamento *relacional* e *objeto-relacional* superaram os outros modelos para a maioria dos cenários, enquanto o modelo de armazenamento *binário* obteve o melhor desempenho para consultas que empregam comparações dispendiosas de dados. Como resultado geral, o modelo de armazenamento *objeto-relacional* apresentou um melhor compromisso quando considerados simultaneamente os quesitos representação, armazenamento e recuperação.

6.2 Trabalhos Futuros

Os trabalhos concluídos neste trabalho abrem a possibilidade de uma variedade de trabalhos futuros. Alguns deles são descritos a seguir.

- Implementação de um sistema que estende o comando CREATE FUNCTION em SGBDRs comerciais, conforme previsto nesta dissertação.
  - Implementação de algoritmos dos operadores por similaridade em SGBDRs que têm a capacidade de reconhecer os blocos de consulta que representam operadores por similaridade e analisar o comportamento das transações dos mesmos junto a consultas envolvendo estes operadores.

  - Criação de um modelo de custo apropriado para avaliar planos de execução que envolvem operadores tradicionais e operadores por similaridade.

- Análise de modelos para armazenamento de dados complexos que consistem em tipos de dados presentes nos SGBDs NoSQL.
- Representação de outros operadores por similaridade utilizando a SQL padrão.
- Análise do comportamento do desempenho de consultas complexas, envolvendo outros operadores por similaridade além da Rq, comparando as respectivas execuções nos modelos de armazenamento com a sua execução indexada.

REFERÊNCIAS

[1] RAZENTE, H. L. et al. Aggregate similarity queries in relevance feedback methods for content-based image retrieval. In: ACM. *Proceedings of the 2008 ACM symposium on Applied computing*. [S.l.], 2008. p. 869–874.

[2] BARIONI, M. C. N. et al. SIREN: A Similarity Retrieval Engine for Complex Data. In: *Proceedings of the 32nd International Conference on Very Large Data Bases*. VLDB Endowment, 2006. (VLDB '06), p. 1155–1158. Disponível em: <http://dl.acm.org/citation.cfm?id=1182635.1164232>.

[3] BARIONI, M. C. N. *Operações de consulta por similaridade em grandes bases de dados complexos*. Tese (Doutorado) — Universidade de São Paulo, 2006.

[4] GULIATO, D. et al. PostgreSQL-IE: An image-handling extension for PostgreSQL. *Journal of Digital Imaging*, Springer-Verlag, v. 22, n. 2, p. 149–165, 2009. ISSN 0897-1889. Disponível em: <http://dx.doi.org/10.1007/s10278-007-9097-5>.

[5] BATKO, M.; NOVAK, D.; ZEZULA, P. Messif: Metric similarity search implementation framework. In: *Digital Libraries: Research and Development*. [S.l.]: Springer, 2007. p. 1–10.

[6] SILVA, Y. N. et al. Similarity joins: Their implementation and interactions with other database operators. *Information Systems*, Elsevier, v. 52, p. 149–162, 2015.

[7] BUDIKOVA, P.; BATKO, M.; ZEZULA, P. Query language for complex similarity queries. In: SPRINGER. *Advances in Databases and Information Systems*. [S.l.], 2012. p. 85–98.

[8] KASTER, D. S. et al. FMI-SiR: A flexible and efficient module for similarity searching on Oracle database. *Journal of Information and Data Management*, v. 1, n. 2, p. 229, 2010.

[9] SILVA, Y. N. et al. SimDB: A Similarity-Aware Database System. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2010. (SIGMOD '10), p. 1243–1246. ISBN 978-1-4503-0032-2. Disponível em: <http://doi.acm.org/10.1145/1807167.1807330>.

[10] LU, W. et al. MSQL: efficient similarity search in metric spaces using SQL. *The VLDB Journal*, Springer, v. 26, n. 6, p. 829–854, 2017.

[11] FALOUTSOS, C. *Searching multimedia databases by content*. [S.l.]: Springer Science & Business Media, 1996. v. 3.

[12] ZEZULA, P. et al. *Similarity Search: The Metric Space Approach*. [S.l.]: Kluwer, 2006. v. 32. 1–191 p. (Advances in Database Systems, v. 32). ISBN 978-0-387-29146-8.

[13] TORRES, R. da S.; FALCAO, A. X. Content-based image retrieval: theory and applications. *RITA*, v. 13, n. 2, p. 161–185, 2006.

[14] BARIONI, M. C. N. et al. Seamlessly Integrating Similarity Queries in SQL. *Software: Practice and Experience*, John Wiley & Sons, Inc., New York, NY, USA, v. 39, n. 4, p. 355–384, 2009. ISSN 0038-0644. Disponível em: <http://dx.doi.org/10.1002/spe.898>.

[15] LONG, L. R. et al. Content-based image retrieval in medicine: Retrospective assessment, state of the art, and future directions. *Int. J. of Healthcare Information Systems and Informatics*, v. 4, n. 1, p. 1–16, 2009.

[16] SILVA, Y. N.; PEARSON, S. S.; CHENEY, J. A. Database similarity join for metric spaces. In: SPRINGER. *International Conference on Similarity Search and Applications*. [S.l.], 2013. p. 266–279.

[17] KASTER, D. S. *Tratamento de condições especiais para busca por similaridade em bancos de dados complexos*. Tese (Doutorado) — Universidade de São Paulo, 2012.

[18] KASTER, D. S. et al. MedFMI-SiR: A powerful DBMS solution for large-scale medical image retrieval. In: SPRINGER. *International Conference on Information Technology in Bio-and Medical Informatics*. [S.l.], 2011. p. 16–30.

[19] WIRTH, M. et al. The effect of mammogram databases on algorithm performance. In: IEEE. *Computer-Based Medical Systems, 2004. CBMS 2004. Proceedings. 17th IEEE Symposium on*. [S.l.], 2004. p. 15–20.

[20] SMISTAD, E. et al. Medical image segmentation on GPUs – a comprehensive review. *Medical image analysis*, Elsevier, v. 20, n. 1, p. 1–18, 2015.

[21] SHYU, C.-R. et al. Local versus global features for content-based image retrieval. In: IEEE. *Content-Based Access of Image and Video Libraries, 1998. Proceedings. IEEE Workshop on*. [S.l.], 1998. p. 30–34.

[22] ANTANI, S.; KASTURI, R.; JAIN, R. A survey on the use of pattern recognition methods for abstraction, indexing and retrieval of images and video. *Pattern recognition*, Elsevier, v. 35, n. 4, p. 945–965, 2002.

[23] TANABE, J. et al. 18.2 a 1.9 tops and 564gops/w heterogeneous multicore soc with color-based object classification accelerator for image-recognition applications. In: IEEE. *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*. [S.l.], 2015. p. 1–3.

[24] WANG, S. et al. Secure and private outsourcing of shape-based feature extraction. In: SPRINGER. *International Conference on Information and Communications Security*. [S.l.], 2013. p. 90–99.

[25] BERTOLINI, D. et al. Texture-based descriptors for writer identification and verification. *Expert Systems with Applications*, Elsevier, v. 40, n. 6, p. 2069–2080, 2013.

[26] EL-NAQA, I. et al. A similarity learning approach to content-based image retrieval: application to digital mammography. *IEEE transactions on medical imaging*, IEEE, v. 23, n. 10, p. 1233–1244, 2004.

[27] KORN, F.; PAGEL, B.-U.; FALOUTSOS, C. On the "dimensionality curse" and the "self-similarity blessing". *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 13, n. 1, p. 96–111, 2001.

[28] LESKOVEC, J.; RAJARAMAN, A.; ULLMAN, J. D. *Mining of massive datasets*. [S.l.]: Cambridge university press, 2014.

[29] NG, R. T.; SEDIGHIAN, A. Evaluating multidimensional indexing structures for images transformed by principal component analysis. In: *Storage and Retrieval for Image and Video Databases (SPIE)*. [S.l.: s.n.], 1996. p. 50–61.

[30] JOLLIFFE, I. T.; CADIMA, J. Principal component analysis: a review and recent developments. *Phil. Trans. R. Soc. A*, The Royal Society, v. 374, n. 2065, p. 20150202, 2016.

[31] KHEDHER, L. et al. Early diagnosis of Alzheimer s disease based on partial least squares, principal component analysis and support vector machine using segmented mri images. *Neurocomputing*, Elsevier, v. 151, p. 139–150, 2015.

[32] FEDEROLF, P.; BOYER, K.; ANDRIACCHI, T. Application of principal component analysis in clinical gait research: identification of systematic differences between healthy and medial knee-osteoarthritic gait. *Journal of biomechanics*, Elsevier, v. 46, n. 13, p. 2173–2178, 2013.

[33] SHAFEY, L. E. et al. A scalable formulation of probabilistic linear discriminant analysis: Applied to face recognition. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 35, n. 7, p. 1788–1794, 2013.

[34] MURTAZA, M. et al. Face recognition using adaptive margin fisher's criterion and linear discriminant analysis. *International Arab Journal of Information Technology*, v. 11, n. 2, p. 1–11, 2014.

[35] WEN, Y.; HE, L.; SHI, P. Face recognition using difference vector plus KPCA. *Digital Signal Processing*, Elsevier, v. 22, n. 1, p. 140–146, 2012.

[36] MASAELI, M.; DY, J. G.; FUNG, G. M. From transformation-based dimensionality reduction to feature selection. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. [S.l.: s.n.], 2010. p. 751–758.

[37] WILSON, D. R.; MARTINEZ, T. R. Improved heterogeneous distance functions. *Journal of artificial intelligence research*, v. 6, p. 1–34, 1997.

[38] LEVENSHTEIN, V. I. Binary codes capable of correcting deletions, insertions, and reversals. In: *Soviet physics doklady*. [S.l.: s.n.], 1966. v. 10, n. 8, p. 707–710.

[39] AGGARWAL, C. C.; HINNEBURG, A.; KEIM, D. A. On the surprising behavior of distance metrics in high dimensional spaces. In: SPRINGER. *ICDT*. [S.l.], 2001. v. 1, p. 420–434.

[40] JIN, H. et al. An adaptive and efficient dimensionality reduction algorithm for high-dimensional indexing. In: IEEE. *19th International Conference on Data Engineering*. [S.l.], 2003. p. 87–98.

[41] LIU, Y. et al. A survey of content-based image retrieval with high-level semantics. *Pattern recognition*, Elsevier, v. 40, n. 1, p. 262–282, 2007.

[42] BUGATTI, P. H. et al. Feature selection guided by perception in medical CBIR systems. In: IEEE. *Healthcare Informatics, Imaging and Systems Biology (HISB), 2011 First IEEE International Conference on*. [S.l.], 2011. p. 323–330.

[43] BARROSO, R. F. et al. Using boundary conditions for combining multiple descriptors in similarity based queries. In: SPRINGER. *Iberoamerican Congress on Pattern Recognition*. [S.l.], 2013. p. 375–382.

[44] BUENO, R. et al. Unsupervised scaling of multi-descriptor similarity functions for medical image datasets. In: IEEE. *Computer-Based Medical Systems, 2009. CBMS 2009. 22nd IEEE International Symposium on*. [S.l.], 2009. p. 1–8.

[45] BUENO, R. et al. Using visual analysis to weight multiple signatures to discriminate complex data. In: IEEE. *Information Visualisation (IV), 2011 15th International Conference on*. [S.l.], 2011. p. 282–287.

[46] BARROSO, R. F. et al. Speeding up the combination of multiple descriptors for different boundary conditions. In: IEEE. *Computing Conference (CLEI), 2015 Latin American*. [S.l.], 2015. p. 1–11.

[47] BARIONI, M. C. N. et al. Querying multimedia data by similarity in relational DBMS. In: YAN, L.; MA, Z. (Ed.). *Advanced Database Query Systems: Techniques, Applications and Technologies*. 1st. ed. Hershey, PA, USA: IGI Global, 2011. cap. 14, p. 323–359.

[48] PENZO, W. Rewriting rules to permeate complex similarity and fuzzy queries within a relational database system. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 17, n. 2, p. 255–270, 2005.

[49] BALAMURUGAN, V.; KANNAN, K. S. A framework for computing linguistic hedges in fuzzy queries. *The International Journal of Database Management Systems*, v. 2, n. 1, p. 1–7, 2010.

[50] YI, Y.; MEI, J.; XIAO, Z. A fuzzy embedded GA for information retrieving from related data set. In: SPRINGER. *Mexican International Conference on Artificial Intelligence*. [S.l.], 2006. p. 943–951.

[51] JEŽKOVÁ, L.; CORDERO, P.; ENCISO, M. Fuzzy functional dependencies: A comparative survey. *Fuzzy Sets and Systems*, Elsevier, 2016.

[52] ADALI, S.; BUFI, C.; SAPINO, M.-L. Ranked relations: Query languages and query processing methods for multimedia. *Multimedia Tools and Applications*, Springer, v. 24, n. 3, p. 197–214, 2004.

[53] ADAH, S.; SAPINO, M. L.; MARSHALL, B. A rank algebra to support multimedia mining applications. In: ACM. *Proceedings of the 8th international workshop on Multimedia data mining:(associated with the ACM SIGKDD 2007)*. [S.l.], 2007. p. 5.

[54] SHAO, Y. et al. An efficient similarity search framework for simrank over large dynamic graphs. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 8, n. 8, p. 838–849, 2015.

[55] ACHTERT, E. et al. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In: *ACM SIGMOD International Conference on Management of Data*. [s.n.], 2006. p. 515–526. Disponível em: <http://doi.acm.org/10.1145/1142473.1142531>.

[56] OLIVEIRA, W. D. d. *Operação de busca exata aos k-vizinhos mais próximos reversos em espaços métricos*. Dissertação (Mestrado) — Universidade de São Paulo.

[57] SILVA, Y. N. et al. Similarity Queries: Their Conceptual Evaluation, Transformations, and Processing. *The VLDB Journal*, v. 22, n. 3, p. 395–420, 2013. ISSN 1066-8888. Disponível em: <http://link.springer.com/10.1007/s00778-012-0296-4>.

[58] PUTTASWAMY, K. P. et al. Preserving location privacy in geosocial applications. *IEEE Transactions on Mobile Computing*, IEEE, v. 13, n. 1, p. 159–173, 2014.

[59] CARVALHO, L. O. et al. MedInject: A general-purpose information retrieval framework applied in a medical context. In: IEEE. *Computer-Based Medical Systems (CBMS), 2014 IEEE 27th International Symposium on*. [S.l.], 2014. p. 308–313.

[60] CAMBAZARD, H. et al. A combinatorial optimisation approach to the design of dual parented long-reach passive optical networks. In: IEEE. *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*. [S.l.], 2011. p. 785–792.

[61] SEDDATI, O.; DUPONT, S.; MAHMOUDI, S. Deepsketch: deep convolutional neural networks for sketch recognition and similarity search. In: IEEE. *Content-Based Multimedia Indexing (CBMI), 2015 13th International Workshop on*. [S.l.], 2015. p. 1–6.

[62] ACHTERT, E. et al. Efficient reverse k-nearest neighbor estimation. *Informatik-Forschung und Entwicklung*, Springer, v. 21, n. 3-4, p. 179–195, 2007.

[63] KORN, F.; MUTHUKRISHNAN, S. Influence sets based on reverse nearest neighbor queries. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 29, n. 2, p. 201–212, maio 2000. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/335191.335415>.

[64] RAZENTE, H. L. et al. A novel optimization approach to efficiently process aggregate similarity queries in metric access methods. In: ACM. *Proceedings of the 17th ACM Conference on Information and Knowledge Management*. [S.l.], 2008. p. 193–202.

[65] TAO, Y.; PAPADIAS, D.; LIAN, X. Reverse knn search in arbitrary dimensionality. In: VLDB ENDOWMENT. *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. [S.l.], 2004. p. 744–755.

[66] LI, Y. et al. Flexible aggregate similarity search. In: ACM. *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. [S.l.], 2011. p. 1009–1020.

[67] LI, F. et al. Exact and approximate flexible aggregate similarity search. *The VLDB Journal*, Springer, v. 25, n. 3, p. 317–338, 2016.

[68] HOULE, M. E.; MA, X.; ORIA, V. Effective and efficient algorithms for flexible aggregate similarity search in high dimensional spaces. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 27, n. 12, p. 3258–3273, 2015.

[69] SILVA, Y. N.; AREF, W. G.; ALI, M. H. Similarity group-by. In: IEEE. *IEEE 25th International Conference on Data Engineering*. [S.l.], 2009. p. 904–915.

[70] TANG, M. et al. Similarity group-by operators for multi-dimensional relational data. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 28, n. 2, p. 510–523, 2016.

[71] KASTER, D. S. et al. Incorporating metric access methods for similarity searching on Oracle database. In: *SBBD*. [S.l.: s.n.], 2009. p. 196–210.

[72] PEARSON, S. S.; SILVA, Y. N. Index-based RS similarity joins. In: SPRINGER. *International Conference on Similarity Search and Applications*. [S.l.], 2014. p. 106–112.

[73] DOHNAL, V.; GENNARO, C.; ZEZULA, P. Similarity join in metric spaces using eD-index. In: SPRINGER. *International Conference on Database and Expert Systems Applications*. [S.l.], 2003. p. 484–493.

[74] BURKHARD, W. A.; KELLER, R. M. Some approaches to best-match file searching. *Communications of the ACM*, ACM, v. 16, n. 4, p. 230–236, 1973.

[75] SKOPAL, T. Where are you heading, metric access methods?: a provocative survey. In: ACM. *Proceedings of the Third International Conference on Similarity Search and Applications*. [S.l.], 2010. p. 13–21.

[76] SILVA, H. B.; PATROCÍNIO, Z. K. G.; GUIMARÃES, S. J. F. Analysis of using metric access methods for visual search of objects in video databases. *Revista de Informática Teórica e Aplicada*, v. 21, n. 1, p. 29–44, 2014.

[77] SOUZA, J. A. de; CAZZOLATO, M. T.; TRAINA, A. J. M. Clusmam: fast and effective unsupervised clustering of large complex datasets using metric access methods. In: ACM. *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. [S.l.], 2016. p. 986–991.

[78] CHEN, L. et al. Efficient metric indexing for similarity search. In: IEEE. *IEEE 31st International Conference on Data Engineering*. [S.l.], 2015. p. 591–602.

[79] 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>.

[80] TRAINA JR, C. et al. Fast indexing and visualization of metric data sets using Slim-trees. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 14, n. 2, p. 244–260, 2002.

[81] TRAINA JR, C. et al. Slim-trees: High performance metric trees minimizing overlap between nodes. In: SPRINGER. *International Conference on Extending Database Technology*. [S.l.], 2000. p. 51–65.

[82] DOHNAL, V. et al. D-index: Distance searching index for metric data sets. *Multimedia Tools and Applications*, Springer, v. 21, n. 1, p. 9–33, 2003.

[83] SHIMURA, T.; YOSHIKAWA, M.; UEMURA, S. Storage and retrieval of XML documents using object-relational databases. In: SPRINGER. *International Conference on Database and Expert Systems Applications*. [S.l.], 1999. p. 206–217.

[84] STHANIKAM, B. et al. *Document fidelity with binary XML storage*. Google Patents, 2012. US Patent 8,090,731. Disponível em: <https://www.google.com/patents/US8090731>.

[85] SIQUEIRA, P. H. B. et al. What lies beyond structured data? a comparison study for metric data storage. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2018. p. 283–291. Disponível em: <https://doi.org/10.1007/978-3-319-98812-2_24>.

[86] SIQUEIRA, P. H. B. et al. Standard sql approaches for similarity searching. In: *Proceedings of the CLEI 2018*. [S.l.]: Institute of Electrical and Electronics Engineers (IEEE), 2018. p. 1–10.

[87] CHA, S.-H. Comprehensive survey on distance/similarity measures between probability density functions. *City*, v. 1, n. 2, p. 1, 2007.

[88] BOLETTIERI, P. et al. CoPhIR: a test collection for content-based image retrieval. *CoRR*, abs/0905.4627v2, 2009. Disponível em: <http://cophir.isti.cnr.it>.

TRABALHOS PUBLICADOS PELO AUTOR

Trabalhos publicados pelo autor durante o programa.

Publicações principais do trabalho.

1. Siqueira, P. H. B., Oliveira, P. H., Bedo, M. V. N. and Kaster, D. S. What Lies Beyond Structured Data? A Comparison Study for Metric Data Storage. In: Hartmann S., Ma H., Hameurlain A., Pernul G., Wagner R. (eds) **29th International Conference on Database and Expert Systems Applications (DEXA)**. Lecture Notes in Computer Science, vol 11030. Springer, Cham, p. 283–291, ISBN 978-3-319-98811-5, 2018.

2. Siqueira, P. H. B., Oliveira, P. H., Bedo, M. V. N. and Kaster, D. S. Standard SQL Approaches for Similarity Searching. In: **XLIV Conferência Latino-americana de Informática (CLEI)**, São Paulo, SP, Brasil. Proceedings of the CLEI 2018. Institute of Electrical and Electronics Engineers (IEEE), p. 1–10, 2018.