Índices para consultas por conteúdo em bancos de dados heterogêneos

GUSTAVO TADAO OKIDA

DISSERTAÇÃO APRESENTADA AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA
UNIVERSIDADE DE SÃO PAULO
PARA OBTENÇÃO DO GRAU DE MESTRE EM
CIÊNCIA DA COMPUTAÇÃO

Área de Concentração: **Banco de Dados**
Orientador: **Prof. Dr. João Eduardo Ferreira**

- São Paulo - 2004 -

Índices para consultas por conteúdo em bancos de dados heterogêneos

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Gustavo Tadao Okida e aprovada pela comissão julgadora.

São Paulo, 6 de maio de 2004

Banca examinadora:

Prof. Dr. João Eduardo Ferreira (orientador)	IME-USP
Prof. Dr. Roberto Marcondes César Junior	IME-USP
Prof. Dr. Caetano Traina Júnior	ICMC-USP

Agradecimentos

Agradeço à minha esposa Caroline pela compreensão, apoio e carinho.

Agradeço aos meus pais por terem me incentivado e apoiado.

Agradeço ao Prof. João Eduardo Ferreira pela oportunidade e dedicação.

Agradeço ao amigo Osvaldo Takai pela ajuda na releitura deste texto.

Agradeço aos meus amigos do Laboratório de Bancos de Dados Avançados do IME-USP pelas ajudas imprescindíveis e pelas experiências trocadas.

Agradeço também a todos os amigos e familiares que de um modo ou de outro me ajudaram a realizar este trabalho.

Agradeço à Deus por tudo.

Gustavo Tadao Okida

Resumo:

O objetivo do trabalho é o de criar uma camada de indexação de dados para consultas por conteúdo em banco de dados heterogêneos. Em sistemas de banco de dados complexos é comum a separação física e lógica dos módulos de dados. Tal separação gera formas heterogêneas de armazenamento de dados nas várias opções de gerenciadores de dados relacionais disponíveis. A criação da camada de indexação foi feita através da implementação de uma linguagem de consulta por conteúdo. Esta linguagem cria um índice de busca para cada significado semântico identificado na consulta em questão.

Abstract:

The aim of this work is creating of a data index layers to answer queries by content in heterogeneous databases. It is common to separate the databases physically and logically into modules of data in complex database systems. This segregation generates a data store heterogeneous types in different relational databases systems. The creation of the index layer was done by the implementation of a content query language. This language creates a search index through query analysis.

Lista de Tabelas

Sumário

Capítulo 1

Introdução

1.1 Motivação e caracterização do problema

O objetivo deste trabalho é o de implementar uma camada de software de modo a criar índices para integração de consultas por conteúdo nos bancos de dados heterogêneos. Em sistemas complexos é fundamental separá-los física e logicamente em módulos de acordo com as regras de negócio [FF00]. Este é o caso dos sistemas de bio-informática que pode ser visto em [BCJFG03].

Um módulo é um subconjunto do sistema que implementa um ou mais conteúdos de negócio. Na maioria dos casos, o desenhista do projeto levanta requisitos em funcão dos conteúdos que o usuário define e esses conteúdos são mapeados em estruturas internas que priorizam a performance ou flexibilidade da implementação. Assim, o usuário sabe o que e como cada módulo implementa os conteúdos definidos por ele, mas não é trivial para ele extrair os dados diretamente das estrtuturas internas.

Cada módulo possui seu próprio banco de dados e suas próprias interfaces de manipulação do seu conteúdo. São essas interfaces bem definidas que permitem que os módulos sejam relacionados entre si. Mas a visão global não é simples, a medida que é necessário conhecer as estruturas de dados interna de cada módulo e saber juntá-las. A visão global é essencial para analisar os fluxos dos dados e seu comportamento.

Assim, é essencial que o usuário consiga abstrair de cada módulo os conteúdos implementados, ou seja, o usuário deve ter uma visão de negócio de cada módulo. Para isso, é necessário que seja disponibilizada uma ferramenta que permita a manipulação desses conteúdos.

Essa ferramenta deve possuir uma linguagem de manipulação de conteúdos e uma estrutura interna para atender eficientemente as consultas. Para

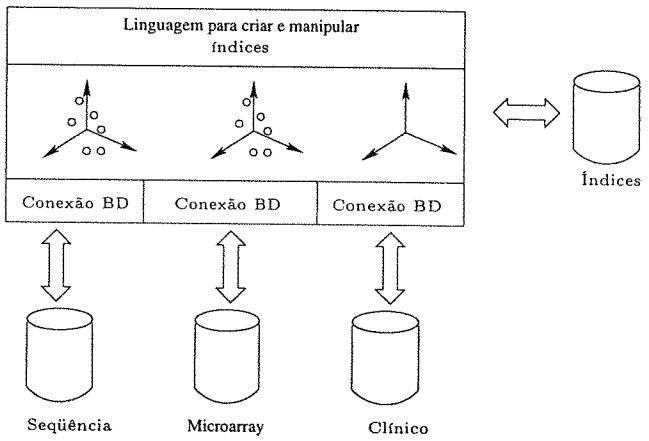

Figura 1.1: Arquitetura do sistema proposto

atender eficientemente as consultas, está sendo proposto dois tipos de índices: Índice Integrador e Índice por Conteúdo. O Índice Integrador está sendo proposto para integrar seletivamente esses dados. A partir do momento que um conjunto de conteúdos e atributos são escolhido para consultas, eles definirão um espaço de consulta. Esse espaço é o Índice Integrador. E para indexar consultas já efetuadas está sendo sugerido um índice baseado nos predicados das consultas. Esse índice armazena hierarquicamente, formando uma árvore, as regioões consultadas, possibilitando o reaproveitamento dos dados das regiões já consultadas. Esse é o Índice por Conteúdo.

No sistema proposto em [BCJFG03] existe uma camada composta por bancos de dados operacionais modularizados, denominada banco de dados primário, para o armazenamento das informações iniciais dos experimentos biológicos. Neste sistema, o banco de dados primário é composto pelos dados experimentais, armazenados separadamente por suas respectivas aplicações conforme figura 1.2.

O banco de dados primário é composto atualmente por quatro módulos:

- Módulo de Seqüência
- Módulo de Microarray
- Módulo Clínico
- Módulo de Controle de Acesso

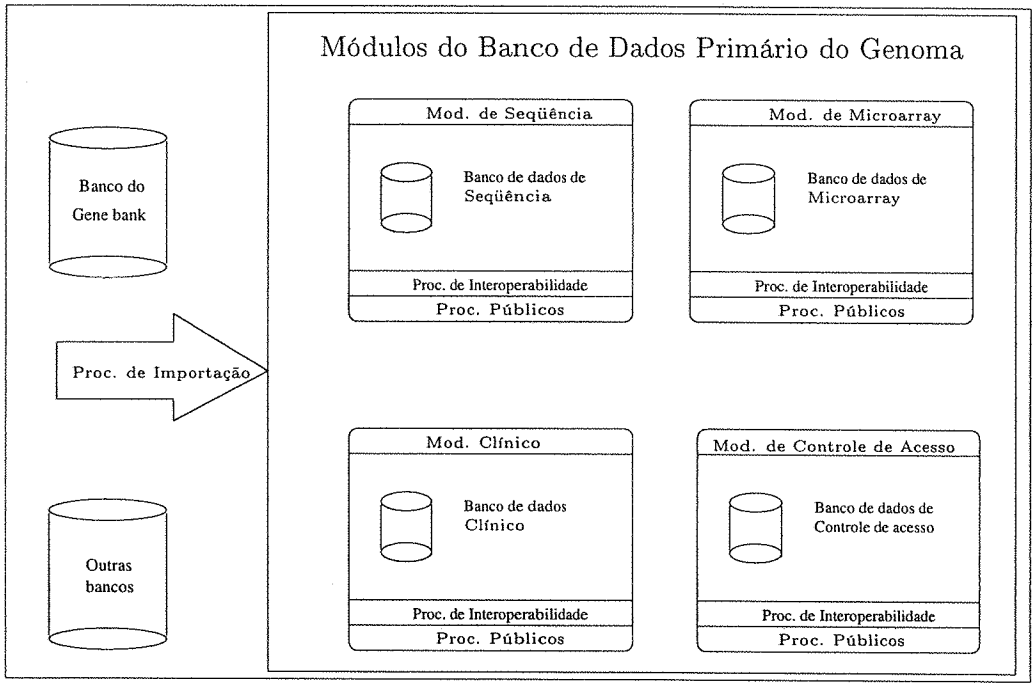

Figura 1.2: Camada de bancos de dados operacional modularizada.

Os três primeiros módulos possuem uma estrutura elaborada para armazenamento dos diversos tipos de informações relativos às seqüências de bases ou proteínas (Módulo de Seqüência), sinal de expressão (Módulo de Microarray) e dados clínicos (Módulo Clínico), respectivamente. O quarto módulo, controle de acesso, é usado para controlar os serviços dos usuários, através de regras definidas por administradores ou pelos próprios usuários.

Para acessar os dados destes módulos, foram definidos três tipos de procedimentos públicos:

- Consultas diretas aos bancos de dados (comandos SQL ou interfaces gráficas).
- Funções de recuperação (em linguagem C++).
- Acesso WEB.

Os módulos, seus procedimentos e alguns bancos de dados externos podem ser vistos na figura 1.3.

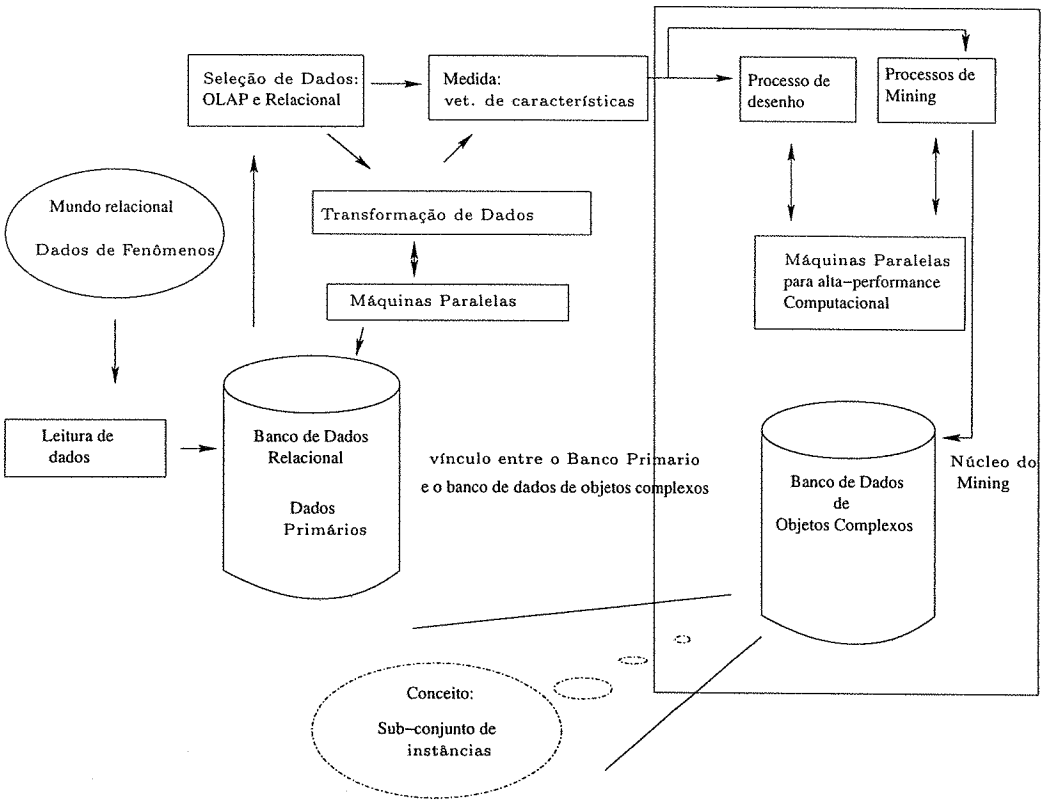

Figura 1.3: Módulos, procedimentos e alguns bancos de dados externas

O Módulo de Seqüência tem uma estrutura de dados flexível que permite alteração periódica do seu conjunto de dados (por exemplo: UniGene, GenBank [int86] [int02] [GBM00] e PRODOM), através de scripts adaptados para cada origem e consultas relacionais, a qual é mais flexível que a oferecida pelos banco de dados públicos. Esta estrutura flexível permite que se integre, facilmente, novos conjuntos de dados de seqüenciamento.

O Módulo de Microarray tem uma estrutura de dados que representa todas as informações necessárias para descrever o chip de DNA (lista de genes, expressões do gene, localização física), os experimentos nos quais ele é aplicado (imagens de microarray, condições experimentais) e os resultados dos experimentos (valor de expressão, erro de leitura, condições de ambiente no momento da coleta dos resultados).

O Módulo Clínico tem informações sobre dados de pessoas, população,

exames clínicos e resistência às drogas.

Os sistemas de bancos de dados que utilizam tais módulos podem ser estendidos facilmente. Para cada novo dado, pode-se localizá-lo através de um identificador no módulo específico. Este método permite que o modelo seja estendido incrementalmente. Módulos adicionais para estruturas de proteínas e metabolismo estão em desenvolvimento.

Os dados primários permitem outro serviço importante: consultas *Online Analytical Process* (OLAP). Este serviço é recomendado para extração de informações em banco de dados com grande volume de informações (na ordem de gigabytes), como é o caso dos bancos de dados de genética. Assim, é possível:

- Formular consultas que englobam tanto dados internos como externos, permitindo que sejam integrados funcionalmente.
- Aumentar substancialmente a eficiência de consultas que são feitas freqüentemente.

Dados internos e externos podem ser extraídos através de interfaces externas padrão ou *gateways*. *Gateways* são aplicações que permitem a realização de consultas aos dados em banco de dados heterogêneas.

As consultas comuns são otimizadas através de classificações dos dados e dos índices pré-compilados. Índices pré-compilados são estruturas de apoio à consulta, persistentes em disco, que se mantém atualizadas com a manipulação dos dados.

As consultas otimizadas são editadas e armazenadas previamente e são acessadas através de interfaces especiais chamadas *views*.

A arquitetura do banco de dados primário, que atende aos serviços OLAP[1], foi desenhada de acordo com o paradigma da estrutura do *Data Warehousse* [ICI01]. É composto por quatro níveis hierárquicos (veja figura 1.4):

- Dados brutos (*raw data*), ou seja, bancos de dados operacionais: seqüência, microarray, clínico e externo.
- Bancos de dados de depósito (*warehouse database*).

[1]Os servidores OLAP tem como objetivo apresentar as informações multidimensionais para as ferramentas de acesso, análise, geradores de relatórios, planilhas e ferramentas de mineração de dados. Basicamente, o servidor OLAP interpreta as consultas dos usuários convertendo-as em instruções adequadas, muitas vezes complexas, para o acesso do *data warehouse*

- Servidores MOLAP/ROLAP [ICI01] e seu banco de dados denominado *db_ molap.*

- Ferramentas de pré-análise, ou seja, visualizadores de consultas relacionais otimizadas, relatórios, planilhas e seleção de dados para o núcleo de *mining.*

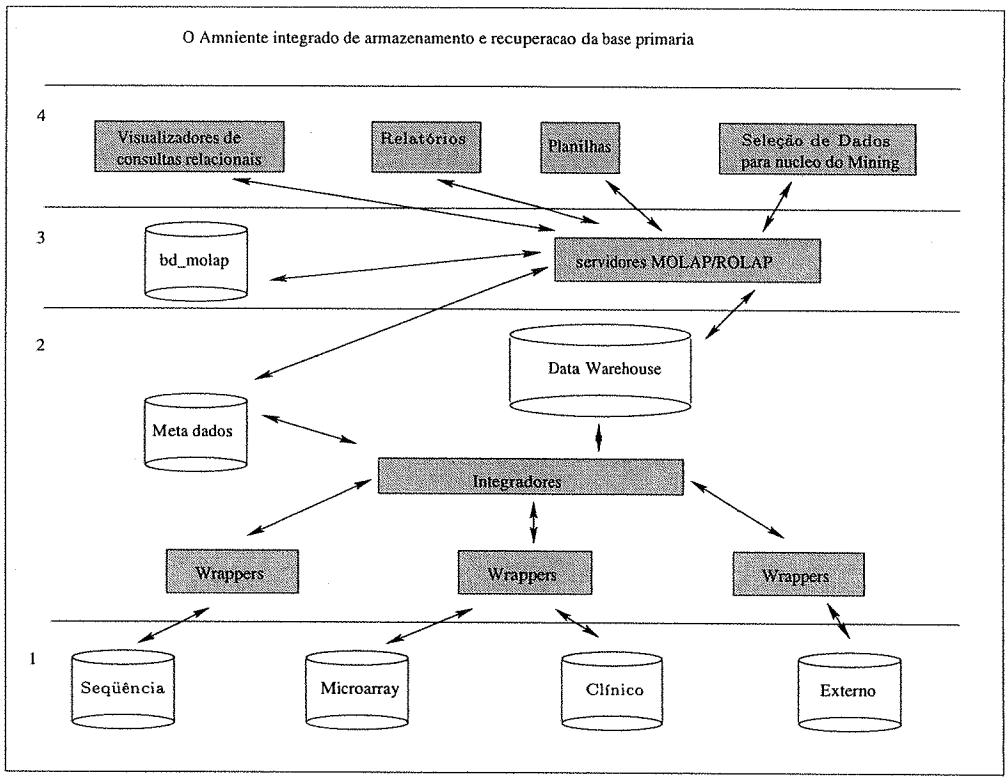

Figura 1.4: Arquitetura da base de dados primária

O primeiro nível mantém os dados em suas estruturas mínimas, ou seja, mantém os dados em seu estado bruto. Os bancos de dados de depósito, no segundo nível, mantém estruturas especializadas, configuradas através dos dados brutos para otimizar o desempenho das consultas de análise de dados. *Gateways*, implementados por *wrappers* e *integradores* que estão no segundo nível, são ferramentas de software que permitem a comunicação entre a base

operacional e os dados do *warehouse*[2]. Os servidores MOLAP[3]/ROLAP[4], no terceiro nível, são armazenadores de dados multidimencionais para melhorar o desempenho de consultas relacionais. Os dados utilizados no quarto nível são acessados através desses servidores MOLAP/ROLAP.

No quarto nível, o módulo *seleção de dados para o núcleo Mining* é composto por aplicações que geram e exportam arquivos intermediários em ASCII e formato XML. Este módulo seleciona e transforma dados dos bancos de dados operacionais (brutos) e OLAP em vetores de atributos, que são processados pelo núcleo de *mining*.

Atualmente, a troca de arquivos implementa a comunicação e integração entre o núcleo *mining* e os bancos de dados. Esta integração permite escalabilidade de processo de *mining* em sistemas de bancos de dados com grande volume de dados, permitindo que diferentes *lay-outs* e conteúdos de arquivos sejam extraídos.

Entretanto, em cada nível desta estrutura de banco de dados, há a aplicação de um tipo de indexação diferente, levando-se em conta a própria estrutura e o tipo de consulta. Isto remete a necessidade de integração das várias formas de indexação, alvo deste trabalho.

As consultas podem ser classificadas na seguinte ordem [ea97]:

- *Consulta por compatibilidade total*, quando são especificados valores para todos os atributos;
- *Consulta por compatibilidade parcial*, quando são especificados valores para alguns dos atributos;
- *Consulta por intervalo*, quando são especificados intervalos para todos ou parte dos atributos;
- *Consulta Booleana*, quando mistura as classes acima;
- *Consulta por similaridade* (por exemplo *vizinho mais próximo* ou *mais compatível*), quando são procurados dois objetos que estejam mais próximos entre si, de acordo com uma métrica de distância ou dissimilaridade especificada. [GSVGM98]

[2]Será usado o termo em Inglês pois é muito utilizado na literatura da língua portuguesa.

[3]Servidores OLAP que utilizam modelos relacionais a tecnologias de busca multidimensionais em cubos pré-construídos [ICI01]

[4]Servidores OLAP em construídos sobre gerenciadores relacionais incrementados com tecnologias de índices bitmap e recuperação de dados com listas invertidas [ICI01]

No sistema proposto em [BCJFG03], os dados serão armazenados por diferentes módulos e representados por diferentes modelos de dados (relacional, objeto-relacional, orientado a objetos e semi-estruturados - XML), além de serem definidas consultas, classificadas de acordo com os critérios apresentados anteriormente, as quais requerem uma estrutura de índices especializada para que a eficiência seja alcançada.

No modelo de *Data Warehouse* proposto, cada nível possui alguns tipos de consultas mais comuns. No primeiro nível, dados brutos, os tipos de consultas mais comuns são do tipo *compatibilidade total e parcial* e *por intervalo*. No segundo e terceiro níveis, banco de dados de *Data Warehouse* e banco de dados OLAP, aplicam-se os tipos de consultas: *compatibilidade total e parcial*, *por intervalo*, *booleana* e por *similaridade*.

Devido ao problema de integração de bancos de dados, este trabalho apresenta uma abordagem para a execução de consultas por conteúdo em bases heterogêneas através de índices globais. Baseado em estudos de integração de bases heterogêneas e distribuídas [BR90] [RAS03], este índice foi desenvolvido com o objetivo de fornecer ao usuário um apoio para consultas globais. Também é proposta uma estrutura para indexar regiões definidas pelas consultas. Esta estrutura é uma variação de árvore-R (ver seção 3.4.1) que armazena hierarquicamente os espaços definidos pelas consultas.

O trabalho está organizado assim: no capítulo 2 são apresentados os principais índices para banco de dados orientado a objetos, no capítulo 3 são apresentados os principais métodos de acesso em espaços vetoriais, e no capítulo 4, os principais métodos de acessos métricos. No capítulo 5 é apresentada a arquitetura do sistema desenvolvido e as duas estruturas de índices sugeridas para resolver os problemas de integração de bancos de dados heterogêneos e indexação por conteúdo de consulta. Já no capítulo 6 são apresentados os resultados obtidos nos testes de performance e finalmente no capítulo 7 está a conclusão da dissertação. No apêncide A, estão descritos os detalhes da linguagem de consulta implementada.

Capítulo 2

Índices para BDOOs

2.1 Banco de dados orientados a objetos (BDOOs)

Os sistemas de banco de dados orientado a objetos integram a tecnologia de bancos de dados com o paradigma de orientação a objeto. Este paradigma prima pela agregação do significado semântico da aplicação no modelo de dados. Assim, os modelos desses bancos de dados se caracterizam por serem mais próximos do modelo do domínio de aplicação em relação aos modelos dos bancos de dados relacional. Contudo, isso gera um grande problema: a recuperação dos dados devido a dificuldade de percorrer as hierarquias de herança de classe [DKR00]. Estruturas de índices tem sido desenvolvidas para auxiliar os algoritmos de busca a percorrer eficientemente essas classes de objetos.

2.2 Definições

Antes de apresentar os diversos tipos de indexações, serão apresensentadas algumas definições importantes. [Ber93]

Definição: Dado uma grafo de agregação H, o *caminho* P é definido como $C_1.A_1.A_2.....A_n$ $(n \geq 1)$ onde:

- C_1 é uma classe em H;
- A_1 é um atributo da classe C_1;
- A_i é um atributo da classe C_i em H, tal que C_i é o domínio do atributo A_{i-1} da classe C_{i-1}, $1 < i \leq n$;

e

len(P) = n representa o tamanho do caminho;

class(P) = $C_1 \cup \{C_i$, onde C_i é o domínio do atributo A_{i-1} da classe C_{i-1}, $1 < i \leq n\}$ representa o conjunto de classes ao longo do caminho;

dom(P) representa a classe domínio do atributo A_n da classe C_n;

duas classes C_i e C_{i+1}, $1 < i \leq n$ - *1*, são chamadas classes vizinhas no caminho.

Um caminho é simplesmente um ramo num dado grafo de agregação. O conceito de caminho está associado ao de caminho instanciado. Um caminho instanciado é uma sequência de objetos obtidos através da instanciação das classes pertencentes ao caminho.

Dado um caminho p=$C_1.A_1.A_2....A_n$, a instanciação de um caminho pode ser:

- completa: quando são definidos n + *1* objetos chamados $O_1.O_2...O_{n+1}$, onde O_1 é uma instância da classe C_1, O_i é o valor do atributo A_{i-1} do objeto O_{i-1} (ou seja, $O_{i-1}.A_{i-1} = O_i$ ou $O_i \in O_{i-1}$, $1 \leq i \leq n-1$).
- parcial: quando são definidos os objetos $O_1.O_2...O_j$, j < n + 1, onde: O_1 é uma instância da classe C_k em class(p) tal que $k + j + 1 = n + 1$; e O_i é o valor do atributo A_{i-1} do objeto O_{i-1} , 1 < i ≤ j.
- não redundante: quando dado uma instância parcial p=$O_1.O_2...O_j$, não existem intâncias (competa ou parcial) p'= $O'_1.O'_2...O'_j$ de p, k>j, tal que O_i=O'_{k-j+1} (i=1,....,j).

Definição: *Grafo indexado* (GI) é a representação abstrata de um conjunto de índices instanciados junto ao caminho P. Dado um caminho

$$P=C_1.A_1.A_2....A_n,$$

um grafo indexado (GI) contém n + 1 vértices, um para cada classe C_i no caminho, mais um vértice adicional representando a classe domínio $C_n.A_n$[1] de um caminho, e um conjunto de arcos direcionados. Um arco direcionado de vértice C_i para o vértice C_j indica que a organização dos índices permitem que hajam associações diretas entre cada instância de C_i com a classe C_j. Note que se C_i e C_j forem classes vizinhas, a organização de índices materializa uma junção implícita entre as classes.

Definição: *Índice regular*: índice semelhante ao encontrado nos bancos de dados relacionais, que indexam valores primitivos (inteiro, palavra).

[1]Esta classe será chamada de C_{n+1} para facilitar a escrita

Definição: *Predicado simples*: Segungo Kim [Kim82], um predicado simples é um predicado de uma consulta Q que possui os seguintes elementos

$$[Conceito_1.atributo_1 \text{ <operador> X}]$$

onde, $Conceito_1.atributo_1$ é um alvo da consulta Q, <operador> é um operador de comparação escalar (<,>,=,<=,>=,<>) e X é um valor.

Definição: *Predicado aninhado*: Segungo Kim [Kim82], o predicado aninhado é uma extensão de um predicado simples, onde X é uma outra consulta. Assim, um predicado aninhado pode ser representado por

$$[Conceito_1.atributo_1 \text{ <operador> Q}]$$

onde, $Conceito_1.atributo_1$ é um alvo da consulta Q, <operador> é um operador de do tipo (contém, não contém) e Q é uma consulta.

Definição: *Predicado de junção*: Segungo Kim [Kim82], um predicado de junção tem a seguinte forma

$$[Conceito_1.atributo_1 \text{ <operador> } Conceito_2.atributo_2]$$

onde, $Conceito_1.atributo_1$ e $Conceito_2.atributo_2$ é um alvo da consulta Q e <operador> é um operador de comparação escalar (<,>,=,<=,>=,<>).

Definição: *Objetos complexos*: No contexto deste trabalho, um objeto complexo é um objeto composto por estrutudas de dados complexas, como, imagens e estruturas de DNA.

2.3 Organização de índices para grafos de agregação

2.3.1 Técnicas básicas

Índice Múltiplo - *Multi-index*

Dado um caminho $P = C_1.A_1.A_2....A_n$, um Índice Múltiplo (IM) [MS86] é definido por um conjunto de n índices simples (chamado índices componentes) I_1,I_2,...,I_n onde I_i é um índice definido sobre $C_i.A_i$, $1 \leq i \leq n$. Todos os índices I_1,I_2,...,I_{n-1} são índices identidade, ou seja, eles tem valores chave (OID). Este tipo de índice atende apenas às comparações == (idêntico à) e $\sim\sim$ (não idêntico à). O último índice I_n pode ser tanto um índice identidade ou índice de igualdade, dependendo do domínio de A_n. O índice de igualdade é um índice regular, como no RDBMS, onde sua chave é um tipo primitivo (números, caracteres...). Este tipo de índice permite comparações escalares como = (igual a), <> (diferente de) , $\leq$, $\geq$, <, >.

Sobre este tipo de organização, a recuperação é feita, primeiramente, através da busca no último índice do caminho. Então, o resultado desta busca restringirá a busca no índice imediatamente superior a este, e assim por diante, até que o primeiro nível seja alcançado. A maior vantagem está no baixo custo de manutenção.

O grafo indexado (GI) para o Índice Múltiplo pode ser visto na figura 2.1. Seja P um caminho de tamanho n. O grafo contém um arco indo da classe C_{i+1} para classe C_i, para $i = 1, ..., n - 1$.

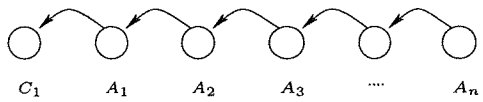

Figura 2.1: Exemplo de um grafo indexado do Índice Múltiplo (IM)

Índice de Junção - *Join-index*

O Índice de Junção (IJ) [Val87] tem boa performance nos bancos relacionais e foi implementado também na indexação dos objetos. Um Índice de Junção Binário (IJB) *binary equijoin index* pode ser difinido como:

Dada duas relações R e S e seus atributos A e B, respectivamente, o Índice de Junção Binário (IJB) é:

IJB=$\{(r_i, s_j) \mid r_i.A = s_k.B\}$

onde:

- $r_i(s_k)$ representa o identificador de uma instância da tupla de $R(S)$.
- tupla r_i(tupla s_k) refere-se a tupla que tem $r_i(s_k)$ como o seu identificador.

O Índice de Junção Binário é implementado como um relação binária do Índice de Junção, mantendo-se duas cópias, uma para r_i e outra para s_k; cada cópia é implementada como uma árvore-B^+ [EN00]. No grafo de agregação, a seqüência de BJIs pode ser usada no Índice Múltiplo para implementar os vários componentes do índice junto ao caminho dado. Esta seqüência é referida como estrutura Índice de Junção.

A estrutura Índice de Junção atende tanto a estratégia de navegação de ida como de volta, onde ambas as cópias são alocadas para cada Índice de Junção. A estratégia de volta percorre o índice da classe domínio até a classe

raiz (como no Índice Múltiplo). Já a estratégia de ida percorre no sentido da raiz até a classe domínio.

O grafo indexado (GI) para o Índice de Junção Binário pode ser visto na figura 2.2. Para cada par de classes C_i e C_{i+1} junto ao caminho P, o grafo contém dois arcos (C_i,C_{i+1}) e (C_{i+1},C_i). O primeiro arco corresponde a cópia do Índice de Junção Binário entre C_i e C_{i+1} na classe C_i enquanto o segundo arco correponde a cópia na classe C_{i+1}.

Note-se que quando a organização Índice de Junção é usada para a passagem de ida, a ordem da seqüência de buscas na árvore-B^+ corresponde a cadeia de arcos no GI. Alem disso, esta cadeia de arcos consiste apenas nos arcos da esquerda-para-direita. Por outro lado, o uso da estrutura Índice de Junção para a navegação de volta corresponde a cadeia de arcos direita-para-esquerda.

O Índice de Junção ou o Índice de Junção Binário podem resolver junções (parte mais complexa de uma consulta) com eficiência, já que não precisa acessar os dados base. No entanto, existem casos em que os índices tradicionais (índice de seleção sobre atributos de junção) são mais eficientes. Por exemplo, um índice tradicional é mais eficiente quando a consulta consiste numa junção precedida por uma seleção altamente restritiva. O Índice de Junção é recomendado para as consultas muito complexas envolvendo várias junções.

O custo de atualização do Índice de Junção Binário é, em geral, o dobro do custo do Índice Múltiplo, devido ao fato de ter duas cópias para cada Índice de Junção. O custo pode ser reduzido, mantendo apenas uma cópia para um ou mais Índice de Junção na organização. Manter uma única cópia, no entanto, aumenta o custo de navegação, e é dependente da cópia que será mantida. A medida para se decidir qual das cópias será mantida depende do estudo prévio dos tipos e frequência das consultas e das atualizações.

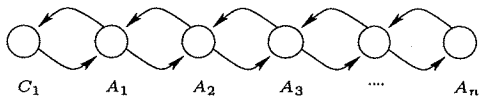

Figura 2.2: Exemplo de um grafo indexado do Índice de Junção

Índice Aninhado - *Nested-Index*

Na resolução de um predicado aninhado, tanto a solução índice múltiplo quanto o Índice de Junção pressupõem que o número de índices a serem

acessados seja proporcinal ao seu tamanho. O Índice Aninhado (IA) se propõe a reduzir o número de acessos à estrutura provendo acesso direto entre a classe de objetos do início do caminho e a classe de objetos do final do caminho.

Note que este tipo de índice resolve apenas a passagem de volta. Contudo, é possível manter duas cópias do índice: a primeira tendo como chave os OIDs da instância da classe no início do caminho, e a segunda contendo como chave os OIDs das instâncias da classe referenciadas pelo arco original.

A recuperação neste tipo de organização é muito eficiente pois não requer que todas as classes intermediárias sejam percorridas. Consultas envolvendo classes não subseqüentes no caminho são resolvidas em apenas um passo. O maior problema deste tipo de índice é o custo de atualização, que requer que muitas classes sejam acessadas para se fazer a atualização.

O grafo indexado (GI) para o Índice Aninhado pode ser visto na figura 2.3. O GI contém apenas dois arcos, chamados (C_1,C_{n+1}) e (C_{n+1},C_1). O primeiro arco é inserido no grafo se a segunda cópia do índice, que atende recuperação do tipo ida, estiver alocado.

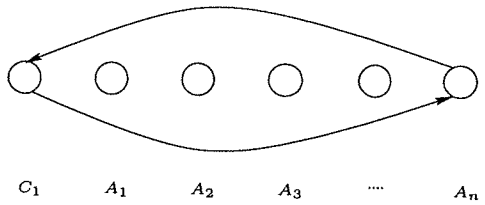

Figura 2.3: Exemplo de um grafo indexado do Índice Aninhado

Índice de Caminho - *Path-index*

Este tipo de índice é parecido com o Índice Aninhado, com a diferença de que o objeto O do final do caminho está associado a todos os objetos que estão no caminho das instâncias que terminam em O. Para um caminho de tamanho n, o objeto folha O possui n componentes.

O Índice de Caminho (IC) [BK89] grava intâncias parciais à direita e à esquerda. Diferentemente do Índice Aninhado, o Índice de Caminho pode ser usado para resolver predicados aninhados com todas as classes pertencentes ao caminho.

Esta característica é muito importante quando se está trabalhando com objetos complexos. Ele permite um tipo especial de projeção, chamado *pro-*

jeção no caminho da instância. Esta projeção permite que sejam recuperados OIDs de várias classes do caminho com uma única pesquisa. [BG93]

Atualizações no Índice de Caminho são muito caras, pois é necessário fazer passagens de ida, como no caso do Índice Aninhado. No entanto, a passagem de volta não é necessária.

O grafo do índice (GI) para o Índice de Caminho pode ser visto na figura 2.4. O IG no caminho P contém n arcos, chamados (C_{n+1},C_i) para todos os i's no intervalo 1,...,n.

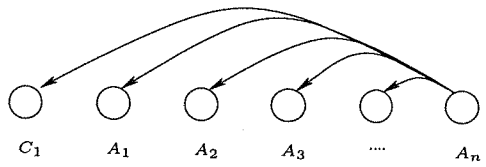

Figura 2.4: Exemplo de um grafo indexado do Índice de Caminho

Relação de Apoio ao Acesso - *Access support relation*

A Relação de Apoio ao Acesso (RAA) [KKM92] é muito similar ao Índice de Caminho pois envolve o cálculo de todas as instâncias do caminho e armazenamento de sua relação. Dado um caminho P=$C_1.A_1.A_2...A_n$, todas as instâncias do caminho são armazenadas como resgistros numa relação de ordem (n+1). O i-ésimo atributo desta relação corresponde a classe C_i. Tanto as instâncias completas como as parciais são representadas por tabela.

O grafo indexado (GI) para o índice Relação de Apoio ao Acesso pode ser visto na figura 2.5. No GI alocado no caminho P, qualquer vétice para a classe C_i, i = 2,....,n-1 tem dois arcos de entrada (C_1, C_i) e ($C_n.A_n$, C_i). Na figura pode-se verificar arcos saindo da primeira classe e da última classe do caminho, sobre as quais são alocadas uma árvore-B^+ para cada classe.

Comparação

Na comparação entre as técnicas de indexação: Índice Múltiplo, Índice Aninhado e Índice de Caminho, um parâmetro significativo é o *grau de referência compartilhada.* Dois objetos compartilham uma referência se eles referenciam o mesmo objeto como valor do atributo.

Segundo [BF95], a recuperação do índice aninhado tem o menor custo e o Índice de Caminho tem custo menor que o Índice Múltiplo. O Índice Aninhado tem performance melhor que o Índice de Caminho na recuperação

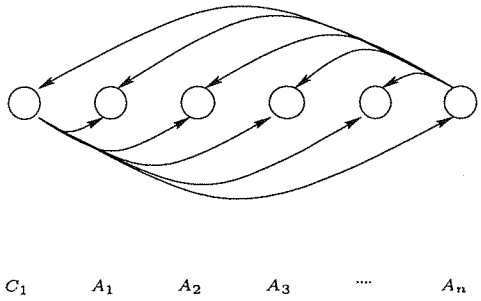

Figura 2.5: Exemplo de um grafo indexado do índice Relação de Apoio ao Acesso

porque o Índice de Caminho contém os OIDs de todas as classes do caminho, enquando o Índice Aninhado contém apenas OIDs das instâncias das primeira classe do caminho. No entanto, um Índice de Caminho simples permite que predicados sejam resolvidos para todas as classes junto ao caminho, enquanto o Índice Aninhado não permite. Para a atualização, o Índice Múltiplo tem o menor custo. O Índice Aninhado tem um custo ligeiriramente menor para caminhos de dois arcos. Para caminhos maiores que 2, o Índice Aninhado tem um custo menor que o Índice de Caminho se a atualização está nas duas primeiras classes do caminho. No entanto, o custo da atualização para o Índice Aninhado são computados sobre a hipótese de que existem referências de volta entre os objetos. Quando não existem tais referências, operações de atualização para o Índice Aninhado se tornam muito mais caras. [BK89]

2.3.2 Técnicas avançadas

As soluções de índices apresentadas na seção anterior são otimizadas para a consulta ou atualização. Nenhuma solução atende bem as duas operações. Nesta seção, serão estudados alguns tipos de índices avançados. Estes índices foram desenvolvidos para melhorar a performance de um determinado conjunto de consultas ou visando algum tipo de atualização.

Quebra de Caminho - *Path-spliting*

A proposta deste tipo de índice é superar os problemas de três índices básicos: o custo alto de atualização do Índice Aninhado e Índice de Caminho e o custo alto de recuperação do Índice Múltiplo. A técnica é baseada na quebra do

caminho original em sub-caminhos, e a aplicação em cada sub-caminho de um dos três tipos de índices: Índice Aninhado, Índice Múltiplo e Índice de Caminho. Por exemplo, seja o caminho $P_1 = C_1.A_1.A_2.A_3.A_4$, pode-se dividí-lo em dois sub-caminhos:

$$P_{11} = C_1.A_1$$
$$P_{12} = A_2.A_3.A_4$$

A figura 2.6 ilustra este exemplo.

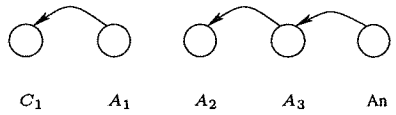

Figura 2.6: Exemplo de um grafo indexado de P_1

Estudos sobre como fazer a quebra dos caminhos têm sido realizados [Ber94] levando-se em conta os seguintes parâmentros: freqüência de recuperação e atualização para classes ao longo do caminho. Além disso, leva-se em conta a existência de arcos de volta, assim como as características lógicas e físicas.

Um exemplo de grafo indexado (GI) para o índice Quebra de Caminho pode ser visto na figura 2.7.

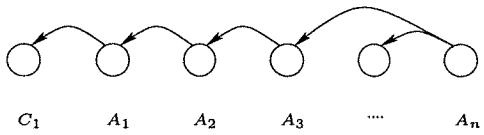

Figura 2.7: Exemplo de um grafo indexado do índice Quebra de Caminho

Decomposição RAA

Cada caminho tem uma única tabela, ou seja, uma única tabela para todas as instâncias. Seguindo o mesmo raciocínio do índice Quebra de Caminho, pode-se quebrar o caminho em sub-caminhos e assim gerar uma tabela para cada sub-caminho.

Um exemplo de grafo indexado para o índice Decomposição RAA pode ser visto na figura 2.8

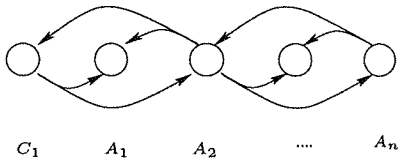

Figura 2.8: Exemplo de um grafo indexado do índice Decomposição RAA

Índice de Junção Hierárquico - *Join Index Hierarchy*

O Índice de Junção Hierárquico (IJH) [FKL00] completo consiste de duas partes: Índices de Junção básicos e Índices de Junção derivados [XH94]. Índice de Junção básicos que formam a base do Índice de Junção Hierárquico são construídos sobre pares de classes vizinhas no caminho, e índices derivados são construídos com pares de classes não-vizinhas.

O custo da manutenção do Índice de Junção Hierárquico é alto tanto em termos de armazenamento quanto de atualização. Assim, o Índice de Junção Hierárquico parcial que contenha todos os Índices de Junção básicos e somente alguns índices derivados são mais eficientes para muitos casos reais. Na hierarquia parcial, qualquer Índice de Junção derivado requerido para a execução de uma consulta, mas não incluso no Índice de Junção Hierárquico parcial, é derivado de índices do Índice de Junção Hierárquico parcial através de uma seqüência de operações de junção. A seleção de Índices de Junção derivados a serem incluídos no Índice de Junção Hierárquico parcial é feita sob algumas heurísticas e métricas. Como reportado em [XH94], o Índice de Junção Hierárquico parcial pode produzir melhores resultados que o Índice de Junção Hierárquico total e o índice Relação de Apoio ao Acesso.

Um exemplo de grafo indexado para o Índice de Junção Hierárquico pode ser visto na figura 2.9. Se o índice contém um arco da classe C_i para a classe C_j, então ele contém o arco de C_j para C_i.

2.4 Organização de Índices para Hieraquia de Herança

Uma consulta pode ser aplicada a apenas uma classe ou a uma classe e todas as suas sub-classes (herdada direta ou indiretamente). Quando um atributo de uma classe C é herdado por todas as sub-classes, a preocupação que se deve ter está em como resolver eficientemente o predicado contra um atributo quando o escopo da consulta é a herança de hierarquia com a raiz em C. Será

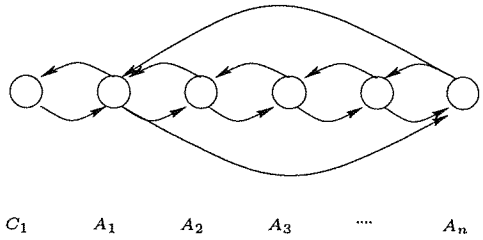

Figura 2.9: Exemplo de um grafo indexado do Índice de Junção Hierárquico

apresentado um resumo dos principais aspectos dos algoritmos, levando-se em conta o custo de atualização e recuperação, bem como o espaço gasto de armazenamento. O custo de recuperação irá depender do tipo da consulta: consulta pontual ou por intervalo.

Seja uma classe C e um atributo A herdado por todas as suas sub-classes. Existem dois tipos de consultas: consulta de uma única classe (*single-class query*), neste trabalho será chamado de consulta-UC para abreviar, se o escopo da consulta for apenas uma classe da hierarquia com raiz em C. Outro tipo de consulta é o consulta de hierarquia de classes (*class-hierarchy query*), neste trabalho será chamado de consulta-HC para abreviar, e seu escopo inclui a sub-hierarquia da herança de hierarquia, ou seja, alguma classe da hierarquia e todas as suas sub-classes. Uma consulta-HC raiz é uma consulta-HC cuja a raiz da sub-hierarquia de pesquisa coincide com a raiz da classe C. Caso contrário, a consulta é uma consulta-HC parcial.

2.4.1 Índice para uma Única Classe - *SC-index (single class index)* e Índice para uma Hierarquia de Classes *CH-index (class hierarchy index)*

Won Kim [KKD89] foi o primeiro a escrever sobre o problema de herança de hierarquia e propor as duas soluções. A primeira, chamada Índice para uma Única Classe - Índice-UC, é baseada na manutenção, em separado, de uma árvore-B^+ sobre atributo indexado para cada classe na hierarquia. O problema é que se a classe possuir m classes, haverá m árvores-B^+. Este índice é muito eficiente para as consultas-UC. No entanto, não é tão eficiente para as consultas-HC, pois requer que todos os índices da hierarquia sejam percorridos.

A segunda solução, chamada Índice para uma Hierarquia de Classes - Índice-CH, é baseada na manutenção de uma única árvores-B^+ para todas

as classes da hierarquia. O nó da árvore-B^+ pode conter OID de qualquer classe da hierarquia. O nó consiste em um valor chave, a chave de diretório e a lista de OIDs para instâncias que assumem este valor chave. O diretório de chaves contém uma entrada para cada classe com chave no atributo do índice. Uma entrada de classe consiste de identificadores de classe e do deslocamento no registro de índices onde a lista de OIDs para a classe está localizada.

Com a estrutura de árvore do Índice-HC, uma consulta-UC é resolvida da seguinte forma: suponha que C seja a classe em que está sendo feita a consulta. Percorre-se a árvore-B^+ a procura da folha com o valor chave da consulta. O diretório de chaves é acessado para verificar se existe uma entrada para a classe C. Se não encontrada, não existe instâncias de C que satisfaça a consulta. Uma consulta-HC é processada da mesma forma, exceto pelo fato que a busca no diretório de chaves é executada para cada classe envolvida na consulta. Em geral, a árvore-HC é boa para auxiliar em consultas que envolvam todas ou quase todas as classes de um caminho.

Kim [KKD89] considerou que a *distribuição de valores chaves através das classes* seria um importante parâmetro de medida de eficiência. Ou seja, se os valores chaves forem formados por instâncias de uma única classe, a árvore do Índice-HC será menos eficiente que a árvore Índice-UC. O custo de atualização da árvore-HC é maior que o Índice-UC porque o tamanho de uma árvore-B^+ para uma classe tende a ser menor que a árvore-B^+ de toda a hierarquia.

2.4.2 Árvore-H - *H-tree*

Para melhorar a performance do Índice-UC nas consultas-HC, Low [LOL92] alterou a estrutura da árvore-B^+. Como no Índice-UC, a árvore-H mantém estruturas de árvores-B^+ para cada classe, mas a diferença está no fato das árvores estarem ligadas através de ponteiros com suas sub-classes. Sejam as classes C e C' numa hierarquia de heranças, onde C' é sub-classe direta de C. Um conjunto adicional de ponteiros é mantido indo dos nós da árvore-B^+ de C até os nós da árvore-B^+ de C'. Os ponteiros conectam nós com os mesmos valores de chave.

Para executar uma consulta-HC, a árvore-H executa uma varredura completa na árvore-B^+ da classe de consulta , seguida por consultas parciais em cada árvore-B^+ das sub-classes na sub-hierarquia a qual a classe consulta é raiz. A consulta parcial consiste em percorrer o ponteiro da árvore-B^+, da classe raiz da consulta, até a árvore-B^+ das sub-classes da classe raiz. Porém o problema de performance não foi totalmente resolvido. Apesar da árvore-H reduzir o número de acesso internos às árvores-B^+, ela ainda requer acesso

a mais páginas que os acessados pelo Índice-UC. Pode-se, ainda, considerar o aumento no espaço requerido para armazenamento, devido aos ponteiros adicionais. Como conseqüência, o custo de atualização do índice é maior.

2.4.3 Árvore-CG - *CG-tree*

A árvore-CG [KM94] é uma alteração na estrutura da árvore-H. A alteração consiste em juntar todos os ponteiros entre diferente índices de classes em nós especiais, criando um nível adicional localizado um nível antes do nível do nó folha da árvore-B^+.

Dada a hierarquia de m classes, a árvore-CG mantém m árvores-B^+, um para cada classe. Em cada árvore-B^+, um nível p é incluído entre os nós internos e o nó folha. Cada nó deste nível contém um vetor de m referências para nós folhas, chamado *diretório de classes*. O i-ésimo componente do diretório de classes contém referência para o nó folha contendo os elementos da classe C_i que possuírem o mesmo valor chave. A posição i da classe C_i é dada pela busca pré-ordem da hierarquia de herança.

2.4.4 Árvore-hcC - *hcC-tree*

A árvore-hcC [SS94] é uma outra organização que tenta combinar vantagens do índice-SC e árvore-CH. Como a árvore-CH é baseada na manutenção de uma única estrutura semelhante à árvore B^+ para indexar a cadeia de hierarquia completa. Além dos nós internos e folhas já conhecidos, introduziu-se o nó conhecido como nó OID. O nó OID fica um nível abaixo do nó folha e contém a lista de OIDs associados aos valores de atributos.

Dada uma hierarquia com m classes, a árvore-hcC mantém $m + 1$ cadeias de nós OID com cadeia de m classes (uma cadeia para cada classe) e uma cadeia de OIDs correspondente a hierarquia inteira. A cadeia de classes para a classe C agrupa os OIDs pertencentes a C, e a cadeia hierárquica agrupa todos os OIDs de todas as instâncias de todas as classes da hierarquia. Praticamente, a cadeia de classe se parece com a cadeia de folhas do Índice-UC. Já a cadeia de hierarquia é similar a cadeia de nós folhas na árvore-CH. Os nós OID são referenciados por entradas nos nós folhas. Cada entrada do nó folha, além do valor chave, contém um bitmap com n bits e um conjunto P de $(m + 1)$ ponteiros. Cada bit no bitmap corresponde a uma classe na hierarquia tal que se o i-ésimo bit estiver marcado, o i-ésimo ponteiro em P apontará para o primeiro nó na cadeia de classes da classe C que contiver OIDs com o valor chave. Cada entrada de nó interno consiste de um valor chave, um ponteiro para nó e um bitmap de n bits.

Para consultas-UC, a performance da árvore-hcC é comparável a da Índice-UC pois ambas requerem busca em apenas uma classe da cadeia. Para as consultas-HC de intervalo na raiz, a árvore-hcC tem performance comparável a da árvore-CH, pois faz a busca em apenas uma cadeia da hierarquia. No entanto, para consultas-HC por intervalo parciais, a árvore-hcC se comporta como o Índice-UC pois requer a busca no número de cadeias igual ao número de classes que envolve a consulta. Como a árvore-hcC armazena cada OID duas vezes, o custo de armazenamento é grande.

Capítulo 3

Índices para espaços vetoriais

3.1 Introdução

Com o advento de sistemas que manipulam dados de natureza espacial, surgiu a necessidade de armazenar e consultar estruturas de dados como pontos, segmentos de retas, retângulos e poliedros. Devido à sua complexidade e a quantidade de informações, os bancos de dados tornaram-se volumosos e as consultas lentas. Outro problema inerente à falta de um padrão para tratamento destes dados é o fato da manipulação dos dados depender de cada domínio de aplicação. Além dos dados geométricos, é comum fazer o mapeamento de outros tipos de objetos em espaços vetoriais, ou seja, abstrai-se dos objetos um conjunto de informações possíveis de serem mapeados para um espaço vetorial.

Ainda existe o problema da ordenação, isto é, dois objetos podem ser considerados próximos quando levado em consideração uma de suas dimensões e muito distante em relação a uma outra dimensão, tornando difícil o estabelecimento de uma ordem absoluta entre os objetos e, como conseqüência, não é possível aplicar os métodos uni-dimensionais.

Os principais tipos de consultas sobre tais objetos estão descritos abaixo. Para efeito ilustrativo, considere um objeto de consulta c, um ponto p e um universo U da consulta, sendo que c pode ou não pertencer a U e $p \in U$.

- *Consulta exata*: achar todos os objetos de U que tem exatamente a mesma extenção de espaço que c.
- *Consulta por ponto*: achar todos os objetos de U que contenha um ponto p.

- *Consulta por intervalo*: achar todos os objetos do banco de dados que tenha pelo menos um ponto em comum com um intervalo dado.
- *Consulta por intersecção*: achar todos os objetos de U que tenha pelo menos um ponto em comum com c.
- *Consulta por abrangência*: achar todos os objetos de U que abranjam totalmente c.
- *Consulta por conteúdo*: achar todos os objetos de U que são abrangidos por c.
- *Consulta por adjascência* : achar todos os objetos adjascentes a c. Um objeto p é adjascente a um objeto q se possuirem somente fronteira em comum.
- *Consulta por vizinhos mais próximos* : achar todos os objetos de U que tenham uma distância mínima de c.

Devido ao problema do cálculo de operações entre objetos ser computacionalmente caro, foram introduzidos métodos de aproximação de dobjetos irregulares por objetos regulares. Um exemplo dessa aproximação é o conceito de MBR (Minimum Boundary Rectangles), menor intersecção de retângulos d-dimensionais que contenha um objeto. O MBR funciona como um filtro na verificação da existência de pontos em comum entre dois objetos, à medida que os respectivos MBRs não tiverem intersecção, estes objetos não tem a chance de ter pontos em comum.

Em, [WR84], 1984, Wong e Raghavan formalizaram um espaço vetorial e concluíram que o modelo é inapropriado para recuperação de informações, sugerindo um modelo com definições mais rigorosas. Devido ao fato de ser muito importante para o presente trabalho, a formalização de um espaço vetorial foi detalhada em 3.2. Três anos mais tarde, em 1987, o trabalho de Wong et. al. [WZRW87] generalizou o modelo de espaço vetorial, formalizando imprecisões e tratando adequadamente o fato dos vetores não serem absolutamente ortogonais entre si, relaxando o modelo para tratar vetores correlacionados. Em 1985, Wong apresentou o trabalho [WZW85] como um método sistemático para computar os termos de correlação diretamente de um esquema automático de indexação. No trabalho de Wang, Wong e Yao [WWY92], os autores apresentaram as limitações do uso do espaço vetorial para problema de geometria computacional. Em [WZRW86], Wong et. al. trabalharam na possibilidade de utilizar operadores lógicos (booleanos) para

resolver consultas em espaços métricos. Este trabalho mostrou uma alternativa para a introdução destes conceitos nos gerenciadores daquela época que não tratavam as operações vetoriais. Em [SWY75], Salton e Wong discutem sobre a relação entre densidade espacial dos dados e a performance do índice.

Os trabalhos de Gaed e Günther [GG98] e Samet [Sam95] são ótimas referências sobre métodos de acessos multi-dimensionais. Para compreender melhor os métodos de acessos espaciais (multi-dimensional), foi proposta a divisão em dois grupos: Métodos de Acesso a Pontos (PAM - Point Access Method) e Métodos de Acessos Espaciais (SAM - Spatial Access Method).

De todos estes estudos, serão concentrados esforços no entendimento dos índice com base na árvore- R.

3.2 Formalidades

Segundo Wong e Raghavan [WR84], a existência de espaço vetorial implica que se tenha um sistema com propriedades lineares, como a soma. Além disso, dentre as propriedades necessárias estão a simetria ($\underline{x} + \underline{y} = \underline{y} + \underline{x}$) e a desigualdade triangular.

A representação de um documento em termos de índice se faz através da caracterização dos termos em vetores no espaço. Isto é, sejam $t_1, t_2, ..., t_n$ os termos usados para representar um documento; para cada termo, t_i, existe um vetor $\underline{t_i}$ no espaço. Assim, um documento D_r, $1 \leq r \leq m$, pode ser expresso em termos de $\underline{t_i}$:

$$\underline{D_r} = (a_{1r}, a_{2r}, ..., a_{nr}).$$

Os vetores $\underline{t_i}$ formam um conjunto gerador, ou seja, todo vetor deste subespaço, e em particular todos os vetores dos documentos, são combinações lineares dos vetores de termos ($\underline{t_i}$). Portanto,

$$\underline{D_r} = \sum_{1 \leq i \leq n} a_{ir} \underline{t_i},$$

onde, a_{ir}, para $1 \leq r \leq m$, são coeficientes correspondentes dos valores de cada vetor $\underline{t_i}$ de $\underline{D_r}$. O conjunto mínimo de vetores necessários para caracterizar um documento devem ser ortogonais entre si e, portanto, lineramente independentes. Um conjunto de vetores $\underline{y}_1, \underline{y}_2, ..., \underline{y}_k$ é linearmente independentes se não for possível achar valores escalares $a_1, a_2, .., a_k$, não todos zero, tal que:

$$a_1\underline{y}_1 + a_2\underline{y}_2 + ... + a_k\underline{y}_k = 0.$$

A função de similariade entre dois objetos é medida através do produto escalar de dois vetores. Dado dois vetores $\underline{v}_1$ e $\underline{v}_2$ e o ângulo θ entre eles, o produto escalar é definido por:

$$\underline{v}_1 * \underline{v}_2 = \mid v_1 \mid * \mid v_2 \mid * cos\theta.$$

A figura 3.1 ilustra o produto escalar entre os vetores $\underline{v}_1$ e $\underline{v}_2$.

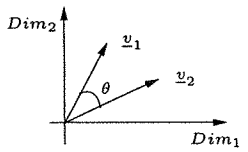

Figura 3.1: Produto escalar de $\underline{v}_1$ e $\underline{v}_2$

3.3 Métodos de Acessos a Pontos - PAM

São métodos que atendem a buscas em bancos de dados que armazenam pontos que podem estar em duas ou mais dimensões. Métodos de acessos a pontos geralmente organizam os dados em *buckets*, os quais correspondem a uma página de disco e um sub-espaço. Os *buckets* pode estar organizados tanto em estruturas simples quanto em estruturas hierárquicas. Basicamente, estes métodos são divididos em métodos baseados em *HASH* e métodos baseados em estruturas hierárquicas.

Segue um resumo dos principais métodos de acessos a pontos.

3.3.1 EXCELL

O método EXCELL [TS82] (*Extended CELL*) possui uma estrutura de dados para prover acesso eficiente à objetos geométricos. Este método combina algumas características da Quadtree (ver seção 3.3.4) e da *fixed cell*. É uma árvore binária junto com a estrutura de vetores que permite acesso por endereço. Decompõe o universo regularmente, todas as células são do mesmo tamanho.

3.3.2 Grid-file

Definido em [JNS84], o *grid file* pode ser visto como uma extensão do HASH em múltiplas dimensões. A idéia deste método é colocar uma grade de linhas no espaço em que estão os pontos, esta grade deve ser ajustada de acordo com a densidade dos pontos. Cada célula da grade deve corresponder, no máximo, a uma página de disco, mas várias células podem estar numa mesma página.

Pontos positivos:

- Garante apenas dois acessos a disco em caso de *consulta por compatibilidade total*;
- É simétrico para o atributo, isto é, trata indiferentemente as chaves primárias e secundárias;
- Adapta-se em caso de distribuição não uniforme dos pontos;
- É uma estrutura dinâmica;

Pontos negativos:

- Não funciona bem se os atributos tiverem correlação, pois a distribuição não é esparsa;
- Necessita de um grande diretório se a dimensão for grande. (*"dimensionality curse"*)

Portanto, esta solução é boa quando se trabalha com dimensões pequenas e com atributos não correlacionados.

Entre suas variações, é importante citar:

- **Twin Grid-File**[1] [HSW88]: tenta melhorar a utilização do espaço utilizado pela estrutura introduzindo um segundo *Grid-file*. O conceito por trás desta introdução é o balanceamento e não a hierarquia, ou seja, todo o universo está espalhado nas duas estruturas e a distribuição dos dados é feita dinamicamente.

[1] Será usado o termo em Inglês pois é muito utilizado na literatura da língua portuguesa.

3.3.3 Árvore-Kd

Uma árvore-Kd [BER85] K-dimensional) é uma generalização em memória da estrutura da árvore binária de busca para dados multi-dimensionais. Uma árvore-Kd é uma árvore binária na qual os nós interiores tem associado a si um atributo a e um valor V que divide o ponteiro de dados em duas partes: aqueles que tem valor a e são menores que V e aqueles com valor a e são iguais ou maiores que V. Os atributo de níveis subseqüentes na árvore são diferentes, com alternância de nível entre os atributos em todas as dimensões.

Na árvore-Kd clássica, os ponteiros para dados são localizados nos nós, como na árvore binária. No entanto, serão feitas duas modificações no modelo inicial para aproveitar o fato do armazenamento ser em bloco:

1. Nós interiores terão apenas um atributo: o valor da divisão para este atributo e ponteiros para filhos a esquerda e direita.

2. Folhas terão blocos de armazenamento, com espaço para quantos registros couberem no bloco .

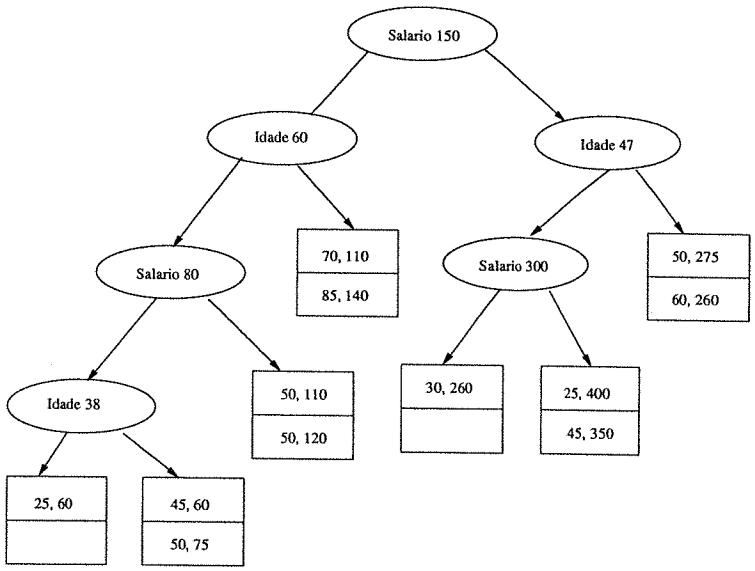

Figura 3.2: Uma exemplo de árvore-Kd

Dentre as suas variações, é necessário citar:

- **Árvore-K-d-B** [PAAV02]: Uma estrutura de dados que reparte os espaços multi-dimensionais como as árvore-Kd, mas a mantém balanceada, como a árvore-B.

- **Árvore-hb** [LS90] : Utiliza árvore-Kd para organizar o espaço, mas exclui intervalos, como nos arquivos BANG[2] [Fre87]. A busca é semelhante à busca da árvore-k-d-B.

3.3.4 Quadtree

Na *Quadtree*[3], cada nó interior corresponde a uma região quadrada em duas dimensões, ou a um cubo k-dimensional em k dimensões. Considerando o caso de um espaço bi-dimensional, se o número de pontos no quadrado não for maior que a quantidade que cabe num bloco, pode-se pensar neste quadrado como uma folha da árvore que está representado pelo bloco que contém os pontos, caso contrário, se existir muito mais pontos do que cabe no quadrado, pode-se tratar o quadrado como um nó interior, com filhos correspondentes a seus quatro quadrantes. Vide figura 3.3

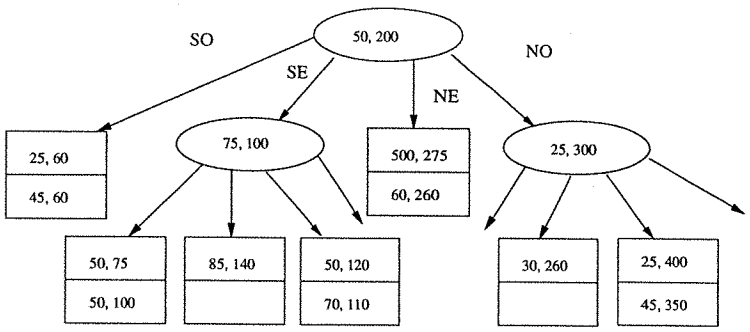

Figura 3.3: Um exemplo de Quadtree

Uma variação da Quadtree é a árvore-P [DMAW85], nela o espaço é particionado por politopos (caixas multi-dimensionais com lados não retangulares) encadeados hierarquicamente. A árvore-R, discutida na seção 3.4.1, é um caso especial de árvore-P, onde os politopos são retângulos.

[2]Possui uma estrutura de árvores de diretório balanceada e sempre expande com a mesma taxa que os dados, não importanto a forma que os dados são distribuídos.

[3]Será usado o termo em Inglês pois é muito utilizado na literatura da língua portuguesa.

3.4 Métodos de Acessos Espaciais - SAM

São métodos que atendem a buscas em bancos de dados que armazenam estruturas espaciais como retas, polígonos, poliedros, entre outros. Estes objetos podem estar em duas ou mais dimensões. [Sam95] apresenta uma visão geral do uso de estruturas de dados espaciais em banco de dados espaciais.

3.4.1 Árvore-R

Uma árvore-R (árvore de Região) é uma estrutura de dados semelhante à árvore-B para dados multi-dimensionais. Muitos pesquisadores têm trabalhado para incrementar esta estrutura [SRF87], [KCK01] e [TS96]. Lembrando que a árvore-B divide a linha em dois segmentos, e que pontos da linha pertencem apenas a um segmento, conforme a figura 3.4, buscar um valor numa árvore-B é trivial, se pensar que o ponto está em algum lugar da linha representado por um nó desta árvore. [NS98].

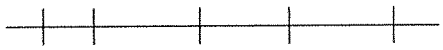

Figura 3.4: Uma árvore-B divide a linha em segmentos disjuntos

Uma árvore-R representa dados que consistem em regiões de duas ou mais dimensões, a qual é denominada de *região de dados*. O nó interior corresponte a *região interna*, ou apenas *região*, que não é normalmente uma *região de dados*. A princípio, a região pode ser qualquer figura, mas por conveniência de manipulação, elas são representadas por retângulos ou figuras simples. Um nó da árvore-R tem, no lugar de chaves, sub-regiões que representam o conteúdo dos filhos. A figura 3.5 sugere um nó para a árvore-R que está associado com um grande e sólido retângulo. Os retângulos pontilhados representam sub-regiões associadas aos seus quatro filhos. Note que as sub-regiões não cobrem todo a região, basta que elas cubram toda a *região de dados*. Estas sub-regiões podem se sobrepor, mas é bom que esta seja a menor possível.

A figura 3.6 mostra como as duas novas folhas entram na árvore-R. As folhas novas estão representadas pelos retângulos menores com linhas tracejadas, interna ao retângulo, também, com linhas tracejadas, denominada pai dessas duas novas folhas. O pai destas duas novas folhas tem ponteiros para ambas as folhas filhas e, associados a elas, estão os pontos superior-direito e inferior-esquerdo das regiões retangulares que cada folha cobre.

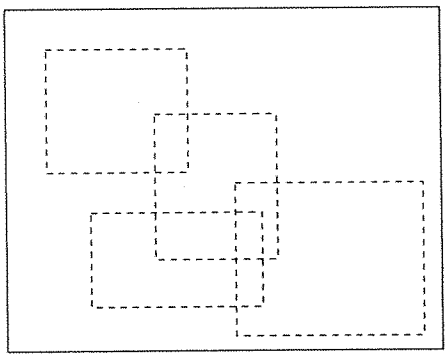

Figura 3.5: A região de uma árvore-R (região delimitada pela linha contínua) e sub-regiões de seus filhos (regiões delimitadas pelas linha tracejadas)

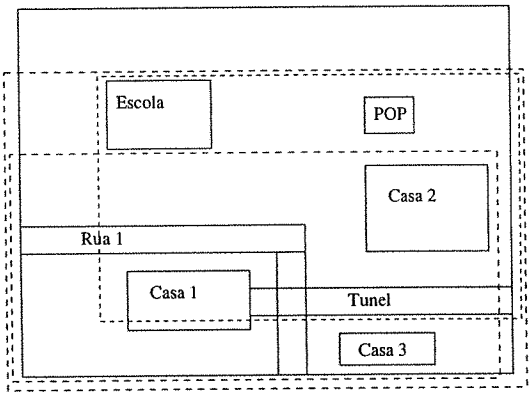

Figura 3.6: Estendendo uma região para acomodar mais dados

Estrutura da árvore

Na árvore-R, os nós podem ser intenos ou externos (também chamados de folhas). Os nós internos têm a seguinte forma:

$$(I, \textit{ponteiro para o filho})$$

onde $I = (I_1, I_1, I_2, ..., I_n)$ é um retângulo n-dimensional que serve como contorno do objeto indexado. O *ponteiro para o filho* é o endereço do nó abaixo na estrutura da árvore. Já os nós externos (folhas), possuem o seguinte formato:

(*I*, *identificador do objeto*)

Cada objeto da base possui um identificador único. Cada nó terá no mínimo $M/2$ e no máximo m ponteiros (para filhos ou para dados), exceto a raiz. O retângulo I de cada nó é o menor retângulo que contém os objetos (novamente, nós filhos ou dados). E, caracterizando o seu balanceamento, todas as folhas estão no mesmo nível.

Assim, a altura da árvore que contém N registros é no máximo

$$\mid log_m N \mid \text{-}1,$$

e o pior caso de utilização de espaço, excetuando-se a raiz, é m/n.

Variações

Dentre suas variações, é necessário citar:

- **Árvore-R^+** [SRF87]: Baseada na árvore-R, elimina a sobreposição, contudo quebra os MBRs, armazenado-os separadamente. Se o banco de dados contiver objetos grandes, sua performance é pior que a árvore-R.
- **Árvore-R^*** [NBS90]: Evolução da árvore-R, não elimina totalmente a sobreposição, contudo, propõe uma maneira mais eficiente de agrupar os MBRs.
- **Árvore-X** [SBK96]: Evolção da árvore-R^* para várias dimensões.
- **Árvore-QSF simples** [BYE99]: elimina completamente a sobreposição de regiões e discrimina totalmente as operações espaciais com uma relação topológica diferente. Entende-se por relações topológicas, as relações que não variam com a translação, rotação e mudança de escala.

3.4.2 Outros métodos de acessos espaciais

Segue uma lista de métodos espaciais que foram evoluções de métodos unidimensionais e métodos de acessos a pontos.

- **Árvore-cell** [Gun86]: Árvore onde níveis sucessivos são divididos por hiperplanos arbitrários. Objetos concavos são decompostos em pedaços convexos. Cada pedaço convexo é indexado em toda célula que o sobrepõe.

- **Árvore-kd estendida**: Um método espacial onde níveis sucessivos são particionados por diferentes dimensões em células sem sobreposição. Objetos são indexados em todas as células que eles intesectam.

- **Árvore-skd**: A árvore k-d espacial é particionada por diferentes dimensões nos seus níveis. Os objetos são indexados por seus centróides, e o MBB (*minimum bounding box*) de um nó é armazenado no próprio nó.

- **Árvore-GBD**: Uma generalização da árvore-BD [] que armazena objetos estendidos como uma hierarquia de MBB (*minimum bounding boxes*). É uma árvore balanceada de multiplos caminhos que serve de método de acesso espacial.

Capítulo 4

Índices para espaços métricos

4.1 Métodos de Acesso Métricos - MAM

Com a evolução dos sistemas computacionais e dos gerenciadores de bancos de dados, está sendo possível o armazenamento e consulta de objetos complexos tais como imagens e estruturas de DNA. Esses objetos são representados pelas suas características, como cor, textura, forma (no caso de uma imagem) e seqüência de caracteres (no caso de DNA). Assim tais características não permitem a existência de uma ordem absoluta. Quando se trata de objetos complexos, o tipo de consulta mais comum é do tipo *busca-por-similaridade* [Cla97]: ou seja, consultas que levam em conta a proximidade (similaridade) entre os objetos. Consulta por intervalo, par de objetos mais próximos e k-vizinhos mais próximos são exemplos de busca por similaridade.

Esses objetos não podem ser representados em espaços multi-dimensionais, pois seus atributos não são valores que podem ser expressos em dimensões vetoriais. Além disso, a similaridade entre objetos é medida através da correlação entre as chaves (suas características). Nos métodos métricos, a função de cálculo de distância tem alto custo, pois as chaves são grandes e complexas. Além disso, a caracterização de um objeto é feita através de muitas medidas, o que implica no uso de um espaço com muitas dimensões e, isto é um fator que degrada o tempo de consultas nos métodos de acessos espacias.

Nos espaços métricos, a forma de calcular uma distância (similaridade) entre dois objetos é através de uma função métrica (ou somente métrica). Uma função é dita métrica se obedecer as propriedades de não-negatividade, simetria e desigualdade triangular. A desigualdade triangular permite que dois objetos sejam comparados através das distâncias relativas a um terceiro

objeto. É nesta propriedade que os algoritmos se baseiam para diminuir a quantidade de cálculos de distâncias. Nos espaços métricos, a distância entre objetos é sempre relativa, ou seja, não existe uma posição absoluta como nos espaços multi-dimensionais.

Assim, tornou-se computacionalmente viável o cálculo da similaridade entre dois objetos complexos. As principais estruturas de índices em espaços métricos estão divididas em duas categorias, segundo Hjaltason e Samet [HS03b]:

- O primeiro, chamado métodos *embedding*, supõe que dos N objetos de U, escolhem-se k características e representa tais objetos como pontos num espaço k-dimensional, onde cada dimensão é uma característica. Assim, supõe-se que a proximidade dos pontos neste espaço k-dimensional implique na proximidade dos objetos originais. Os atrativos são a possibilidade de usar os métodos SAM (vide capítulo 3) para otimizar a consulta e a redução da dimensionalidade.

  Segundo o estudo realizado em [HS03a], é necessário que os métodos de mapeamento para os espaços k-dimensionais sejam contrativos[1] para garantir que a resposta seja correta (sem eliminar possíveis candidatos). Um método é contrativo se para qualquer par de objetos, a distância entre eles seja maior ou igual a distância entre os respectivos pontos mapeados no espaço k-dimensional através da escolha de k características. Assim,

  $$\delta(F(o_1), F(o_2)) \leq d(o_1, o_2), \text{ para todo } o_1 \text{ e } o_2 \in S$$

  onde, $F(o_i)$ representa a projeção do objeto no espaço F considerado, $\delta(F(o_1), F(o_2))$ representa a função de distância neste espaço, $d(o_1, o_2)$ é a distância entre os objetos e S é o universo dos objetos.

- O segundo, chamado de indexação baseada em distâncias (*distance-based indexing*), supõe a utilização da informação de distância para indexar os dados. As distâncias consideradas são as distâncias entre alguns objetos selecionados e todos os outros do espaço considerado. A vantagem é que as distâncias já estão pré-computadas, eliminando o custo alto para o seu cálculo. Contudo, se for necessário utilizar várias medidas de distância, é necessário criar um índice para cada tipo de medida.

[1] Em inglês, *contractive*

Hjaltason e Samet [HS03a] desenvolveram um *framework* para comparação de métodos do segundo grupo, *distance-based indexing*, para consultas por similaridade.

O trabalho escrito por Baeza-Yates et. al. [ECM01] também divide os métodos de acessos em dois grupos: o primeiro denominado métodos baseados em pivoteamento e o segundo baseado nos algoritmos de Voronoi. Nos métodos baseados em pivoteamento, são escolhidos N objetos denominados pivôs (também conhecidos como *vantage-point*) e são calculadas todas as distâncias entre os N objetos e os outros objetos do espaço considerado. Já nos métodos baseados no diagrama de Voronoi onde os objetos pivôs são escolhidos e é construído o diagrama de Voronoi, cada pivô determina uma região chamada sítio que possui a seguinte propriedade: todos os objetos deste sítio estão mais próximos do seu pivô que qualquer outro pivô.

Contudo, é possível também dividir os métodos em duas categorias que levam em consideração o modelo de construção da estrutura (árvore) e a sensibilidade da estrtura para a dinamicidade dos dados:

- Métodos Estáticos: grupo de soluções que não trata inserções, remoções ou atualizações na árvore. Utiliza a abordagem *top-down* de construção e não garante balanceamento, necessitando de reorganizações periódicas.
- Métodos Dinâmicos: atualiza a árvore à medida que os dados são alterados. São estruturas que privilegiam o balanceamento e ambientes dinâmicos.

Basicamente, os Métodos de Acesso Métricos dividem o conjunto de dados em regiões e escolhem estrategicamente objetos representantes para cada região. Em um nó, são armazenados o representante, os objetos pertencentes a sua região e suas distâncias destes objetos em relação ao representante. Os nós são armazenados hierarquicamente, formando uma árvore.

Uma consulta começa com a comparação do objeto de consulta com o representante de cada região e, então, a propriedade de desigualdade triangular é usada para cortar (*prune*) cálculos de distâncias desnecessárias.

4.2 Formalidades

Definição de função métrica

Dado um conjunto de objetos $S = \{s_1, s_2, ..., s_n\}$ de um domínio S, a função d que tiver as seguintes propriedades:

- 1. Simetria: $d(s_1,s_2) = d(s_2,s_1)$
- 2. Não negatividade: $0 < d(s_1,s_2) < \infty$, $s_1 <> s_2$ e $d(s_1,s_1) = 0$
- 3. Desigualdade triangular: $d(s_1,s_3) <= d(s_1,s_2) + d(s_2,s_3)$

é chamada função métrica. A figura 4.1 ilustra a propriedade da desigualdade triangular.

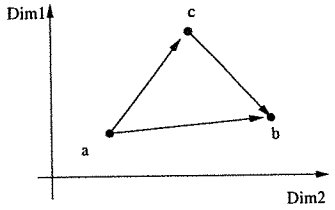

Figura 4.1: Desigualdade triangular: neste espaço 2D, a propriedade está ilustrada por $d(a,b) <= d(a,c) + d(c,b)$

Definição de espaço métrico : É o espaço M=<S,d> onde d é uma função métrica.

Definição de busca por intervalo (*range query*): Dado um objeto de consulta $s_q \in S$, a uma distância máxima de consulta r_q, a resposta é um subconjunto de S tal que $Rconsulta(s_q,r_q)=\{s_i \mid s_i \in S : d(s_i, s_q) <= r_q\}$.

Definição de busca por vizinho mais próximo (*nearest near neighbor query*): Dado um objeto de consulta $s_q \in S$, o vizinho mais próximo é o conjunto unitário de S tal que $Nconsulta(s_q)=\{s_n \in S \mid \forall s_i \in S : d(s_n, s_q) <= d(s_i, s_q)\}$

Definição de busca de par mais próximo (*closest pair query*): Este tipo de busca objetiva encontrar o par de objetos (s_i,s_j), i < j, s_i e $s_j \in$ S, de modo que qualquer outro par de objetos pertencentes à S (s_a,s_b), a < b, e diferentes de (s_i,s_j), vale a propriedade:

$$d(s_i,s_j)<d(s_a,s_b)$$

4.3 Métodos Estáticos

Os métodos estáticos foram as primeiras abordagens utilizadas pelos métodos de acesso métricos. Dentre os trabalhos mais significativos pode-se citar

o trabalho feito por Burkhard e Keller em [BK73] que sugeriram uma técnica recursiva para particionamento do espaço métrico em relação a um objeto representante (chamado de *vantage-point*), que gera uma árvore. Nesta mesma linha de raciocínio está o trabalho de Uhlmann em [Uhl84] , Yanilos em [Yia93] e Brin em [Bri95]. Em [Bz97], Bozkaya and Ozsoyoglu propuseram uma extensão da árvore-VP chamada árvore-MVP (*multi-vantage-point tree*) que escolhe m pontos de referências (*vantage points*).

A árvore-GH [Uhl84] particiona o espaço em hiperplanos. Este método pega dois pivôs p_1 *e* p_2 e divide o conjunto restante de objetos com base na sua distância em relação aos pivôs, ou seja:

$$S_1 = \{o \in S \backslash p_1, p_2 \mid d(p_1, o) \leq d(p_2, 0)\} \text{ e}$$
$$S_2 = \{o \in S \backslash p_1, p_2 \mid d(p_2, o) \leq d(p_1, 0)\}.$$

Os objetos de S_2 estão mais próximos de p_2 que p_1. Esta regra permite que haja dois conjuntos bem diferentes em relação a quantidade de elementos, caso não haja critério de escolha de pivôs. A árvore binária é montada através da aplicação recursiva desta regra.

O GNAT [Bri95] é uma generalização da árvore-GH, ou seja, são escolhidos mais de dois pivôs para particionar o conjunto, criando uma árvore n-ária.

Em 1995, Faloutsos descreveu o FastMap [FL95]. Basicamente, Faloutsos apresenta um algoritmo de mapeamento de objetos em espaços k-dimensionais, de tal modo que as propriedades de similaridade sejam preservadas. Esta abordagem traz dois benefícios imediatos: a possibilidade de utilizar os algoritmos SAM para acessar esses objetos e a visualização dos objetos em espaços 2D ou 3D. A árvore-DF, proposto em [TTFF02], minimiza os cálculos de distância entre objetos. O conceito básico usado é o de representantes globais para toda a árvore.

Estes métodos estão sendo modificados para tratar o dinamismo dos dados, tornando-os métodos dinâmicos.

4.4 Métodos Dinâmicos

São modelos mais recentes que levam em consideração a dinamicidade dos sistemas. Entre os trabalhos, é possível citar:

- **Árvore M ou** *M-tree* [PCZ97] e [CPRZ97]: é uma árvore balanceada capaz de gerenciar arquivos de dados dinâmicos, não necessitando

de reorganizações periódicas. Ela indexa objetos através da função métrica que compara os valores das características que não estão em espaços vetoriais ou utilizam funções de distância, como a métrica L_p. Organiza os objetos em nós de tamanho fixo, que correspondem a regiões de espaço métrico.

- **Árvore M^+ ou M^+-*tree*** [XZY03] : A árvore-M^+, segundo estudo feito por Zhou et. al. [XZY03], leva vantagem sobre a árvore-M e a árvore-MVP, usando um conceito chamado chave de dimensão, que diminui de fato o tempo de resposta para consultas por similaridade. A idéia é particionar um árvore grande em dois sub-espaços, chamados nós-gêmeos. Com isso, é possível dobrar a eficiência da filtragem. Os dados são alocados dinamicamente entre os nós gêmeos.

- **Árvores Magras** *Slim-tree* [TTSF00] : Uma estrutura semelhante a uma árvore onde os dados são armazenados nas folhas e os nós não folhas estão organizados hierarquicamente. A diferença está na existência de um algoritmo de quebra baseado no algoritmo de árvore geradora mínima (Minimal Spanning Tree), um novo algoritmo de inserção e a introdução do algoritmo denominado Slim-down que faz a árvore métrica mais estreita e rápida. Este processo ocorre após a inserção. São apresentadas as medidas *fat-factor* e *bloat-factor* para medir o grau de sobreposição da árvore, onde valores próximos de zero indicam baixo grau de sobreposição.

- **Familia Omni** [FTJF01]: É uma família de Métodos de Acesso Métricos que propõe que um conjunto de objetos sejam escolhidos como focos globais (**Omni-foci base**) e que sejam calculadas todas as distâncias (**Omni-coordinates**) entre esses objetos e os demais do conjunto. O objetivo de se ter os focos é diminuir o número de cálculos de distâncias para chegar no resultado final. A cada inserção de um novo objeto no conjunto, os **Omni-coordinates** são calculados e armazenados, para que sejam usados como valores de comparação. Para consultas por similaridade, Santos et. al. [FTJF01] comprovou sua eficiência através da diminuição do número de cálculos de distância em cada consulta.

Capítulo 5

Arquitetura do Sistema

A arquitetura proposta tem o objetivo de criar uma camada de índices para integrar os bancos de dados modularizados. A integração é o alvo deste trabalho, assim como a indexação de consultas por conteúdo. Em sistemas complexos é fundamental a separação física e lógica dos módulos de acordo com as regras de negócio [FF00]. Este é o caso dos sistemas de bio-informática que podem ser encontrados em [BCJFG03].

Contudo, com esta abordagem, perde-se a visão global das informações à medida que os dados ficam encapsulados em seus módulos e, na maioria das vezes, em bancos de dados distintos. O usuário necessita ter a visão global para analisar o fluxo dos dados e o seu comportamento. A visão global é obtida através da junção e cruzamento das diversos bancos de dados relacionados direta ou indiretamente.

No entanto, geralmente os usuários do sistema são pessoas especializadas no conteúdo da informação (negócio) que estes sistemas disponibilizam. Portanto, para esses usuários, é conveniente que a interface de manipulação dos dados globais utilize como objetos os conteúdos do negócio, abstraindo a forma como tais objetos são implementados.

No contexto de sistemas de biologia proposto em [BCJFG03], cada módulo possui seu próprio banco de dados e os relacionamentos são muito bem delimitados, e definem a interface de comunicação entre os módulos (vide figura 5.1). Os relacionamentos entre os módulos são utilizados para a união e extração dos conteúdos. Esses relacionamentos, no caso de banco de dados modularizados tendem a ser por conteúdo. Assim, não existe uma restrição de integridade entre os módulos de dados em virtude da separação dos dados e da possibilidade da existência de gerenciadores de dados heterogêneos.

Em cada módulo são extraídos conteúdos significativos para o negócio e

relacionados entre si através dos relacionamentos de interface entre os módulos. Como exemplo, no banco de dados do Módulo de Microarray é possível extrair os conteúdos CHIP, EXPERIMENTO e EXPRESSÃO. No banco de dados de Módulo de Seqüências (DNA) é possível extrair MUTAÇÃO e RESISTÊNCIA. Esses conteúdos são relacionados através dos relacionamentos de interfaces. Deste modo, o usuário enxerga o banco de dados como um repositório de conteúdos e não como um conjunto de tabelas, muitas vezes relacionadas entre si de modo não intuitivo.

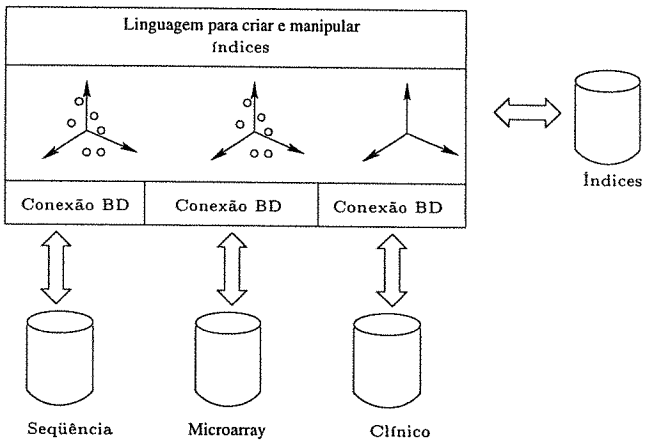

Figura 5.1: Arquitetura do sistema

Com o objetivo de oferecer uma interface de consulta por conteúdo, foi criado um sistema que implementa uma linguagem de consulta (vide figura 5.1). Tal linguagem de consulta cria um índice de busca para cada significado semântico pertencente a uma determinada consulta. Detalhes dessa proposta serão apresentadas no decorrer desse capítulo.

Foi definida como Meta-Consulta (MC) a linguagem de consulta baseada em conteúdos. Esta linguagem é semelhante à consulta SQL, porém o usuário tem a sua disposição um conjunto de conteúdos e atributos ao invés de um conjunto de tabelas e atributos. A MC tem como função abstrair conteúdos de negócio dos bancos de dados para que o usuário não se preocupe com o modelo físico do banco de dados, mas apenas com o modelo de negócio. Resumidamente, a MC é uma linguagem de consulta baseada em conteúdos.

No caso de estudo utilizado nesta dissertação, existem três módulos de bancos de dados:

1. Banco de Dados de Seqüência, pertencente ao módulo de Seqüencia.

2. Banco de Dados de Microarray, pertencente ao módulo de Microarray

3. Banco de Dados Clínicos, pertencente ao módulo Clínico

O pesquisador, ao interagir com o sistema proposto, não está interessado em saber como foram mapeados os objetos de negócio nos respectivos módulos de dados. Ele objetiva recuperar, quantificar e analisar os conteúdos ou elementos dos conteúdos de seu negócio, que estão de alguma forma representados nos módulos.

Então, com base nas soluções de integração de esquemas globais em bases distribuídas e heterogêneas, surgiu a idéia de se construir uma camada chamada DICIONÁRIO que mapeia e relaciona os objetos de negócio para os objetos implementados nos módulos de banco de dados físicos. Ou seja, cria um CONTEÚDO que é a abstração dos objetos dos bancos de dados

Com isso, o usuário tem à disposição uma linguagem de consulta que manipula os objetos de modo a facilitar a elaboração das consultas e compreensão dos seus resultados.

A escolha pelo padrão SQL se deu em função da forma natural de manipulação de conjuntos e do seu amplo uso em sistemas de gerenciadores bancos de dados relacionais.

O DICIONÁRIO pode ser entendido como um serviço que disponibiliza CONTEÚDOS e será utilizado pelo tradutor de consultas para mapear as consultas elaboradas pelo usuário em consultas que possam ser executadas pelos bancos de dados.

No DICIONÁRIO, existem dois tipos de mapeamentos:

1. Mapeamento simples (CONTEÚDO simples): onde um objeto de negócio é mapeado para um conjunto de objetos físicos (tabelas) relacionados entre si (tabelas com suas respectivas junções - *joins*), de modo que envolva apenas uma ou duas tabelas com grandes volumes de dados e que seu custo de recuperação será baixo. Por exemplo, o objeto de negócio SCANNER é mapeado diretamente para a tabela *scanner_pri.*

2. Mapeamento complexo (CONTEÚDO complexo): onde um objeto de negócio é mapeado para uma coleção de objetos físicos relacionados entre si (tabelas com suas respectivas junções - *joins*), de modo que envolva tabelas com grande quantidade de registros ou que possua funções de agregação, e por isto o seu custo de recuperação é alto (em relação ao tempo). Por exemplo, o objeto de negócio EXPERIMENTO é mapeado para as tabelas *experimento*, *protocolo_experimento*, *slide*,

chip, *meta_experimento*, *expressao*, *material* e *slide_detalhe*, pertencentes ao banco de dados de Microarray.

No primeiro caso, os atributos do objeto físico são mapeados diretamente para o objeto de negócio, apenas dando-se um nome mais amigável. Já no segundo caso, existe um problema de semântica que deve ser tratado, ou seja, dois campos podem ter nomes diferentes, mas com o mesmo significado ou, por outro lado, ter o mesmo nome com significados diferentes.

Um espaço de consulta, chamado somente de ESPACO, é um espaço vetorial onde as suas dimensões são atributos do objetos definidos no DICIONARIO. A figura 5.2 ilustra este fato. Os atributos são escolhidos de tal forma a criar um espaço que represente um universo que conterá as regiões definidas pelos predicados das consultas. Em outras palavras, cada consulta definirá sub-regiões dentro deste espaço conforme o seu predicado. Os ESPAÇOS, por terem sido criados com dimensões pertencentes a CONTEÚDOS diferentes, definem um índice para integração de consultas (ver seção 5.1.3).

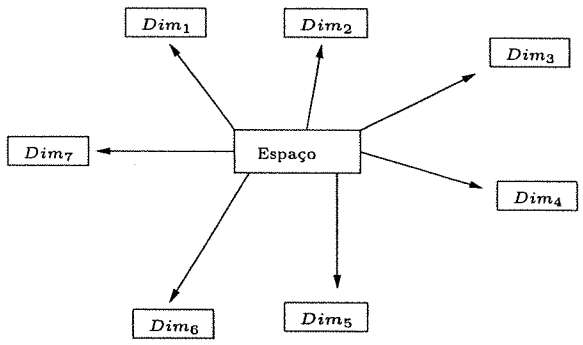

Figura 5.2: Neste caso, o espaço é definido pelas dimensões de 1 a 7, sendo que cada dimensão representa um atributo de um objeto do DICIONÁRIO.

Os valores de cada dimensão do ESPAÇO são obtidos através do mapeamento definido pelos objetos no DICIONÁRIO.

A seguir, são detalhados os componentes do sistema sugerido.

5.1 Componentes

A arquitetura do sistema proposto possui os seguintes componentes:

1. Componente de interpretação (ver seção 5.1.1).

2. Componente de índices (ver seção 5.1.2).

3. Componente de recuperação de dados (ver seção 5.1.3).

A figura 5.3 ilustra como estes componentes estão relacionados entre si.

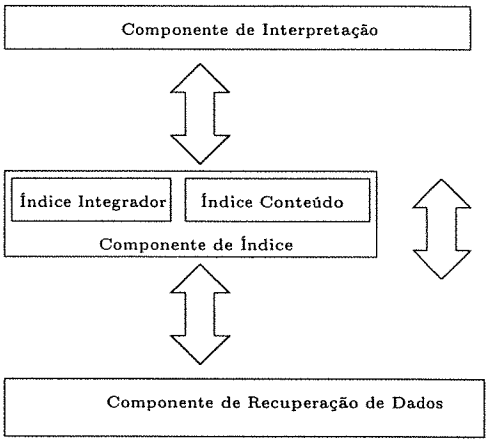

Figura 5.3: Componentes do sistema

5.1.1 Componente de interpretação

O componente de interpretação é responsável por analisar sintaticamente os comandos escritos pelo usuário. Os comandos que este componente aceita são apresentados em três grupos:

1. Comandos de sistema: são comandos que alteram o comportamento do sistema (ver apêndice A.1).

2. Comandos de manipulação de estrutura: são comandos de criação, alteração e remoção das estruturas dos objetos (ver apêndice A.2).

3. Comandos de manipulação de dados: são comandos que permitem manipular os dados (ver apêndice A.3).

Quando o comando é uma consulta, este componente também é responsável por dividí-la em sub-consultas de tal modo que cada sub-consulta represente uma consulta sobre apenas um conteúdo.

5.1.2 Componente de Índices

O objetivo desta camada é oferecer uma base de pré-consulta de tal modo que as consultas sejam otimizadas. Nesta camada, estão as estruturas de índices que indexarão os objetos de negócios.

Esta camada não tem como objetivo a sobreposição da camada de índices de cada SGBD, mas aproveitá-la de modo que todo o trabalho de otimização de cada SGBDs seja aproveitado. Mas, quando se trata de um esquema global representando os diversos bancos de dados, se faz necessário um índice integrador, ou seja, um índice que integra os diversos CONTEÚDOS. Para otimizar consultas por intervalo, isto é, consultas que definem regiões, está sendo proposto um índice baseado no conteúdo das consultas.

Estes índices baseiam-se no DICIONÁRIO para conhecer como os diversos objetos de negócio se relacionam e, por conseqüência, como os objetos físicos estão relacionados.

Como dito anteriomente, esta camada possui dois conceitos de índices:

- **Índice Integrador**: ou Índice de Integração de Banco de Dados, tem como objetivo integrar os objetos definidos no DICIONÁRIO. É baseado na árvore-B devido a natureza dos dados do projeto [BCJFG03]. Contudo, pode ser estendido a outros tipos de árvore à medida que o tipo de consulta e dos dados se modifiquem. Este índice é implementado na estrutura ESPACO. (ver seção 5.2)

  A figura 5.4 ilustra um índice de integração de dados.

  O Índice Integrador pode indexar parcialmente os dados, de acordo com a região em que as consultas ocorrerão. Esse tipo de abordagem permite a criação de índices mais enxutos, ou seja, índices com parte dos dados do banco de dados. Isso não é possível nos bancos de dados relacionais.

- **Índice por Conteúdo**: ou Índice Baseado nos Conteúdos das Consultas, tem como objetivo a indexação de regiões de consultas de acordo com as consultas feitas pelos usuários. Sobre os atributos definidos no DICIONÁRIO, este índice monta uma árvore de regiões ordenadas de tal modo que seja possível recuperar os dados que já foram consultados. Como é necessário indexar regiões de consultas, a estrutura escolhida foi a árvore-R, porém com algumas modificações significativas: a árvore não é balanceada e pode possuir mais de uma raiz. Não ser balanceada é conseqüência do fato do índice ser construído com

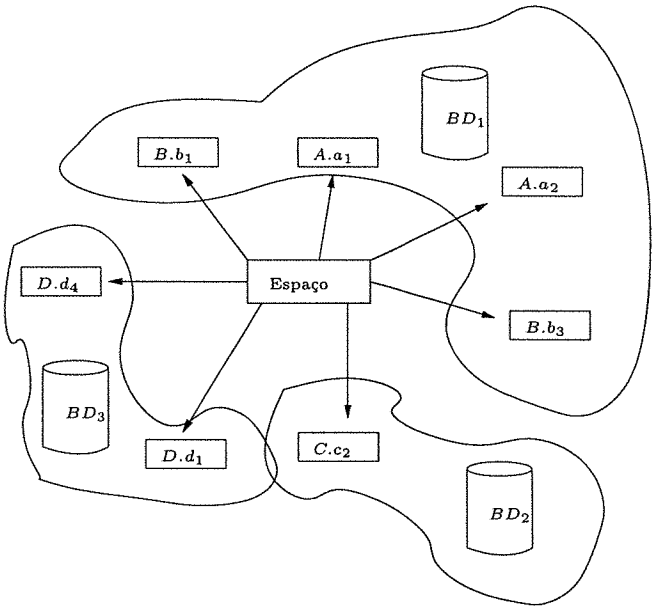

Figura 5.4: Ilustração de um índice de integração de dados

base nas consultas efetuadas. Ter mais de uma raiz é conseqüência da não sobreposição das consultas. (ver seção 5.3)

A figura 5.5 ilusta um Índice por Conteúdo.

5.1.3 Componente de recuperação de dados

Este componente é responsável por recuperar os dados das bases de dados. Após a interpretação de um comando de consulta, o componente de interpretação (ver seção 5.1.1) gera comandos SQL para cada gerenciador e, assim, cada comando SQL é executado nos SGBDs específicos, ou busca os dados dos ESPAÇOS pré-definidos pelo administrador.

Este componente trata os resultados, colocando-os em uma estrutura interna que permite a integração destes dados com outros resultados de comandos SQL. Ele também trata os eventuais erros de comunicação com os SGBDs.

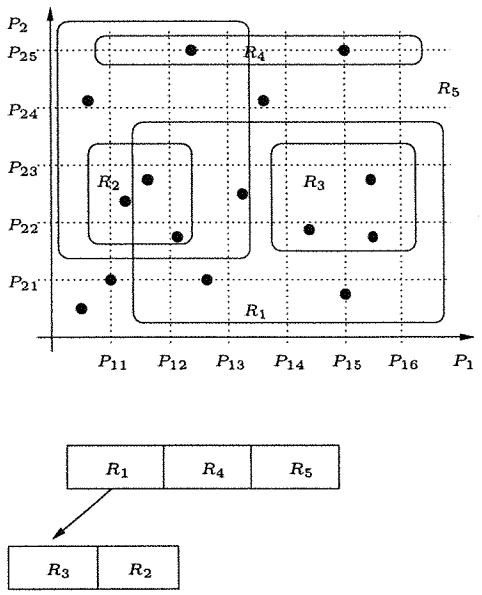

Figura 5.5: Ilustração de um espaço bi-dimensional e um Índice por Conteúdo, onde cada região R_i representa uma região definida pelo predicado da consulta, e os pontos representam os objetos do banco de dados mapeados neste espaço.

5.2 Índice integrador

O índice proposto para a integração dos bancos de dados, Índice Integrador, consiste na criação de espaços k-dimensionais de consulta. Os espaços são definidos pelas possíveis dimensões que farão parte do predicado de uma consulta. A figura 5.6 ilustra um espaço definido por atributos de diversos conteúdoss.

Cada dimensão é um atributo de um CONTEÚDO. A árvore é montada de acordo com os valores que este atributo assume e armazena nas suas folhas os identificadores lógicos[1] (OID) dos objetos nos bancos de dados. O armazenamento dos identificadores lógico permite que outros atributos desse objeto sejam recuperados com uma consulta simples.

Esta estrutura permite que sejam criados índices de uma determinada sub-região de dados, ou seja, é possível selecionar do banco de dados apenas objetos com determinados valores no atributo indexado. Os índices dos

[1] Um identificador lógico é o identificador do objeto no banco de dados

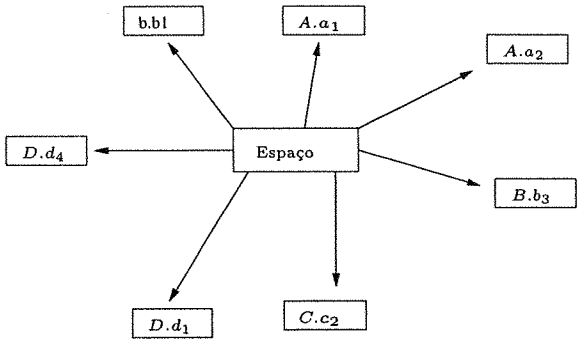

Figura 5.6: Espaço k-dimensional

bancos de dados relacionais não permite a indexação parcial de um atributo. Assim, através da seleção dos objetos a serem indexados, é possível construir uma árvore menor e, portanto, mais eficiente.

5.3 Índice por Conteúdo

O índice proposto tem o objetivo de indexar o conteúdo das consultas; isto é, gerar regiões de consultas de acordo com os predicados especificados para cada conteúdo, dentro do espaço especificado. Após análise dos diversos tipos de índices existentes, foi implementada uma estrutura que indexa regiões retangulares. A escolha foi feita depois de analisar os tipos de consultas descritas em [BCJFG03] e pela característica dos dados, isto é, dados que permitem um ordem absoluta entre eles, além de permitir sobreposição de regiões.

A extrapolação de um ESPAÇO de consulta em um espaço k-dimensional, permitiu criar uma estrutura de árvore para indexar as consultas efetuadas no sistema.

5.3.1 A estrutura de dados

A seguir são detalhadas as principais estruturas de dados do sistema proposto.

Esta estrutura representa um possível espaço de consulta, ou seja, é uma estrutura k-dimensional onde cada dimensão é um atributo de um objeto definido no DICIONÁRIO. A figura 5.7 ilustra um espaço com três dimensões, onde as dimensões são o atributo a do objeto A, o atributo b do objeto

B e o atributo c do objeto C. Os pontos representam os objetos deste espaço.

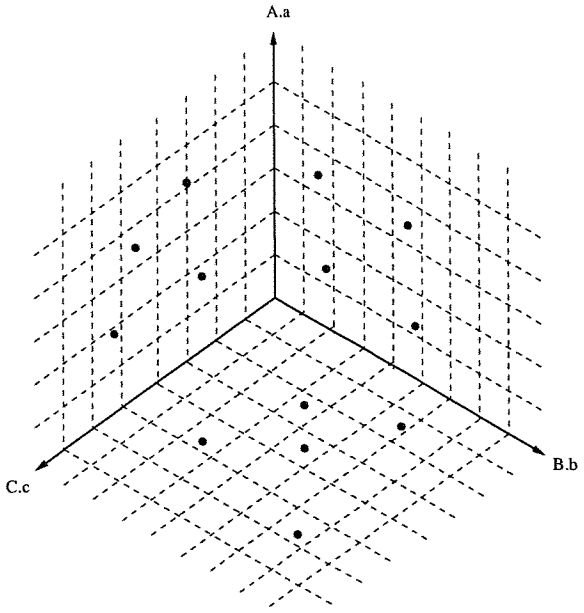

Figura 5.7: Espaco 3-D

Cada dimensão é composta por uma árvore-B que indexa os possíveis valores de atributo desta dimensão. Cada valor está associado a uma lista de identificadores de objetos que possuem este valor. A figura 5.8 ilustra esta estrutura.

A Árvore

A árvore de indexação de consultas é uma árvore que indexa regiões de consultas. Estas regiões são formadas através dos predicados de cada consulta. A figura 5.10 ilustra uma região definida pelo seguinte predicado (A.a[=3,5=] $\wedge$ B.b«4). A sintaxe dos predicados está explicado no apêndice A.4. A figura 5.9 ilustra esta árvore.

A árvore é composta por nós (ver seção 5.3.3). Outra grande diferença em relação às arvores de indexação tradicionais (árvore-R e árvore-B, por exemplo) é que a nas árvores tradicionais existe a diferença entre nós folhas e nós internos, onde apenas os nós folhas possuem apontadores para os dados. Na árvore proposta, todos os nós possuem uma lista de identificadores. Esta lista representa os objetos dos bancos de dados que estão dentro do espaço

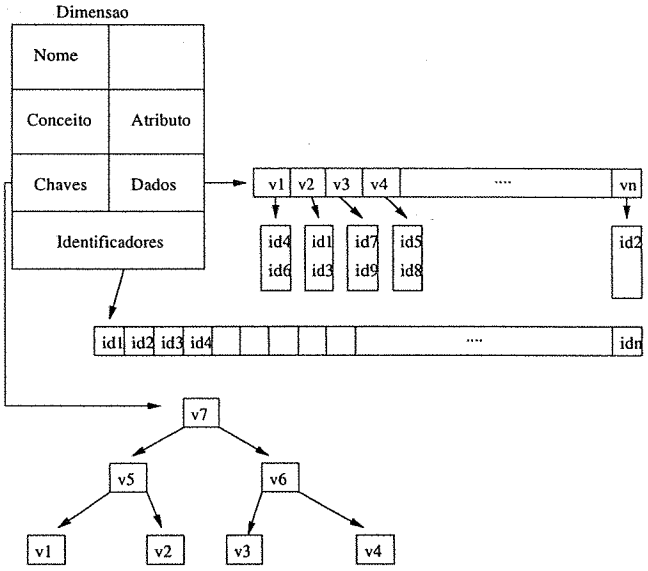

Figura 5.8: Dimensão

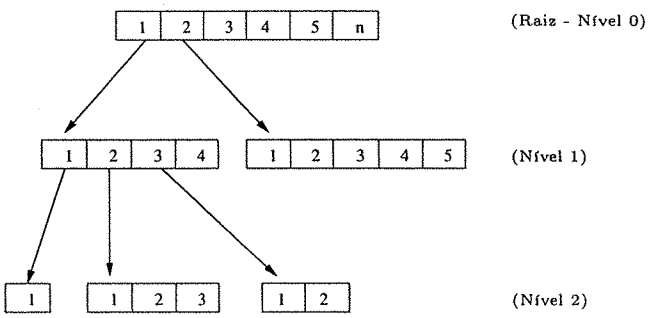

Figura 5.9: Árvore

definido pelo nó. A forma de representar esta região é definida pela estrutura MBR (vide 5.3.2). Sejam N e M dois nós pertencentes a esta árvore, a hierarquia é baseada na seguinte relação:

- M e N são irmãos se e somente se $M <> N$ ou $M \subset_p arN$ ou $N \subset_p arM$
- M é filho de N se e somente se $N \subset_t otM$

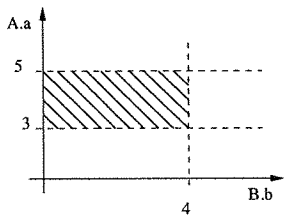

Figura 5.10: Região de consulta em um espaço 2-d, definido por (A.a[=3,5=] $\wedge$ B.b«4)

Os algoritmos de busca e inserção estão definidos nas seções 5.3.4 e 5.3.4, respectivamente.

5.3.2 O MBR

A região definida pelo predicado da consulta é chamada de MBR, assim, cada MBR representa um sub-espaço. As operações definidas para o MBR são:

- Adição (MBR1 + MB2 = MBR3): a adição entre dois MBRs significa somar as regiões definidas por cada MBR. A figura 5.11 ilustra esta operação.

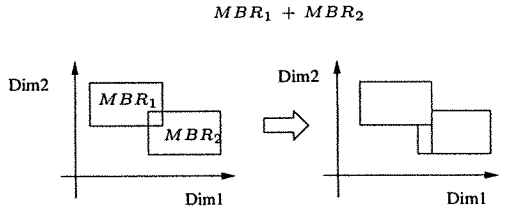

Figura 5.11: Adição de MBRs

- Subtração (MBR1 - MB2 = MBR3): a subtração entre dois MBRs significa subtrair do MBR1 a região comum com MBR2. A figura 5.12 ilustra esta operação.
- Comparação (MBR1.comparar(MBR2)) pode ser (ver seção 5.18):
  - Contido totalmente ($MBR1 \subset MBR2$) : se MBR1 está contido

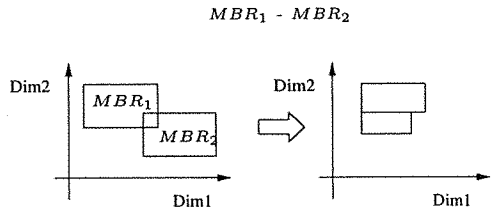

Figura 5.12: Subtração de MBRs

totalmente MBR2, ou seja, o sub-espaço definido por MBR1 está totalmente dentro do sub-espaço definido por MBR2.

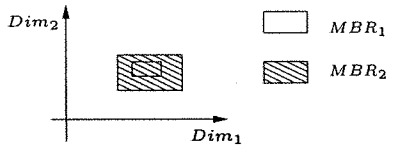

Figura 5.13: $MBR1 \subset MBR2$

- Contém totalmente ($MBR1 \supset MBR2$): se MBR1 contém totalmente MBR2, ou seja, o sub-espaço definido por MBR1 contém totalmente o sub-espaço definido por MBR2.

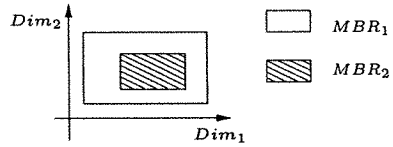

Figura 5.14: $MBR1 \supset MBR2$

- Contem parcialmente: se parte de $MBR1 \subset MBR2$, mas não totalmente. Existe uma região de MBR2 não contida em MBR1.
- Igual: se MBR1 == MBR2, ou seja, o sub-espaço definido pelos dois MBRs são idênticos.
- Diferente: se MBR1 <> MBR2, ou seja, não existe nenhum ponto em comum.

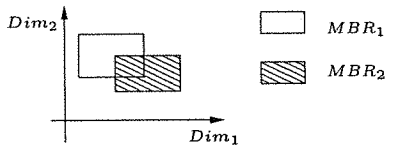

Figura 5.15: MBR1 contém parcialmente MBR2

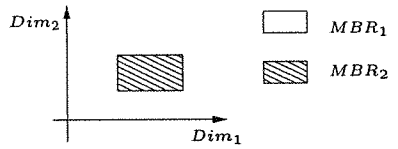

Figura 5.16: MBR1 == MBR2

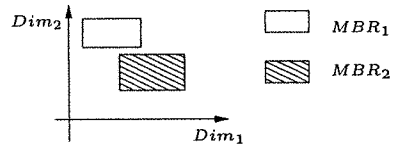

Figura 5.17: MBR1 <> MBR2

5.3.3 O Nó

Um MBR (ver seção 5.3.2) está associado a uma lista de identificadores de objetos de um conteúdo na estrutura denominada NÓ. A comparação entre nós é definida pela comparação entre seus MBRs (ver seção 5.3.2), assim, sejam N e M dois nós, a relação entre eles é definida por:

- M contém totalmente N se e somente se a região definida por M contém totalmente a região definida por N. (item (a) da figura 5.18)
- M contém parcialmente N se e somente se existe ponte de intersecção nas regiões definidas por M e N, mas existe pelo menos um ponto de N que não está em M. (item (b) da figura 5.18)
- M está contido totalmente em N se e somente se toda a região definida por M está contida na região definida por N. (item (c) da figura 5.18)
- M é igual à N se e somente se a região definida por M é igual a região definida por N. (item (d) da figura 5.18)

- M é diferente de N se e somente se não existe ponte de intersecção entre as regiões definidas por M e N. (item (e) da figura 5.18)

A figura 5.18 ilustra as definições acima.

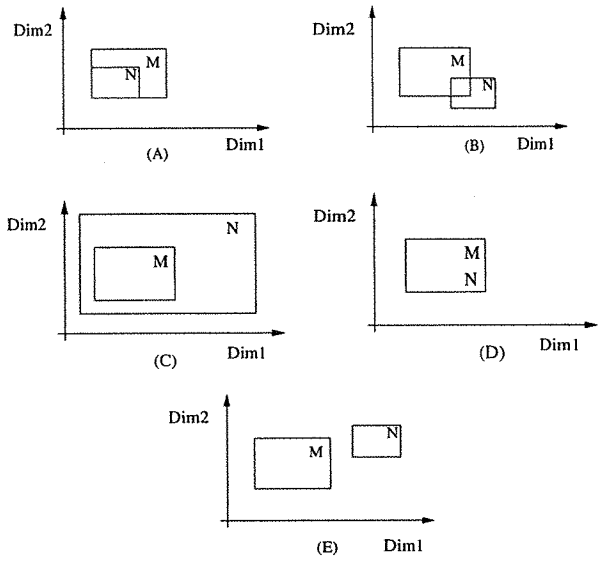

Figura 5.18: Comparação entre nós

Seja S um conjunto de nós e um nó N, então a comparação entre o conjunto S e o nó N é definida unindo-se todas as regiões definidas por cada nó de S e a região definida por N. Assim, a relação entre S e N pode ser: (A figura 5.19 ilustra as definições)

- S cobre perfeitamente N se existe um nó de S, que seja igual a N. (item (A) da figura 5.19)
- S cobre totalmente N se toda a região coberta por N cobre a região definida por N. (item (B) da figura 5.19)
- S cobre parcialmente N se existe intersecção das regiões definidas por S e N, mas existe pelo menos um ponto da região definida por N que não está em S. (item (C) da figura 5.19)
- S é coberto totalmente por N quando toda a região coberta por N cobre a região definida por S. (item (D) da figura 5.19)

- S não cobre N se não existe nenhum ponto na intersecção entre as regiões definidas por N e S. (item (E) da figura 5.19)

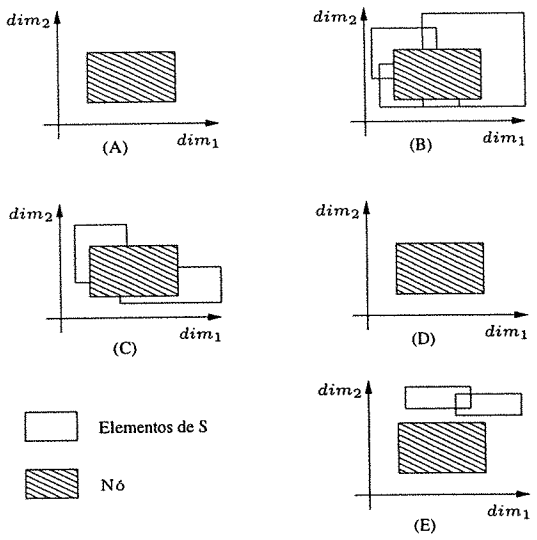

Figura 5.19: Comparação entre um nó N e um uma região definida por um conjunto de nós S

5.3.4 Algoritmos

A seguir, os algoritmos de busca e inserção serão apresentados.

Algoritmo de busca

O algoritmo de busca recebe como parâmetro um nó (*noBusca*) e retona a comparação deste nó com os nós da árvore. O algoritmo *obterFolhas* devolve um vetor de objetos que melhor cobre o espaço definido pelo *noBusca*. O algoritmo *verificaCobertura* compara o vetor de nós com o *noBusca*.

```
consultar(NoSimples noBusca)  {
      vetNosCobertura = obterFolhas(raiz, noBusca);
      retorna verificarCobertura(noBusca, vetNosCobertura);
}

obterFolhas(Vector raizLocal, NoSimples noBusca) {
      S = cobertura_mínima(no, raizLocal)
            Se (S == noBusca) então retorna S
            Se (S está contido em noBusca) então
                  retona S
            Se (Existe s em S tal que s contenha noBusca) então
                  obterFolhas(S.lilho, noBusca)
            Se (S está contido parcialmente noBusca) então
                  retona S
}

cobertura_mínima(no, raizLocal) {
      Se existe s em raizLocal tal que s.MBR == no.MBR então retorna IGUAL
      Seja S={todo s em raizLocal tal que s.MBR esteja contido em no.MBR}
            Se S <> NULL então
                  retorna S
            Senão
                  Seja S'={todo s em raizLocal tal que s.MBR contenha no.MBR)
                        retorna S
}

verificarCobertura(No noBusca, Vector ret) {
      Se noBusca está contido em ret
         então retorna COBRE TOTALMENTE
      Se noBusca contem ret
```

```
            então retorna COBERTO TOTALMENTE
        Se existe um nó em ret igual à noBusca
            então retorna COBERTO PERFEITAMENTE
        Se noBusca está contido parcialmente em ret
            então retorna COBERTO PARCIALMENTE
        Se noBusca não está coberto por ret
            então retorna NAO COBERTO
}
```

Algoritmo de inserção

O algoritmo de inserção insere na árvore um nó de acordo com as regras de leação irmão e filho (ver definição em 5.3.1).

```
adicionar(No no, Vector raizLocal, int nivel) {
      S = cobertura_mínima(no, raizLocal)
            Se (S == no) então retorna Falso
            Se (S está contido em no) então
                  Todos os elementos de S serão filhos de no
                  Inserir no em raizLocal
            Se (Existe s em S tal que s contenha no) então
                  adicionar(no, s.filho, nivel)
            Se (S está contido parcialmente no) então
                  Inserir no em raizLocal
}
```

Capítulo 6

Resultados

O protótipo do sistema foi construído para validação dos conceitos de índices apresentados nesse trabalho. Esse sistema foi integrado aos bancos de dados propostos em [BCJFG03], implementados no Laboratório de Bancos de Dados Avançado do IME-USP. Para validação do processo, as estruturas dos bancos de dados de Microarray, Clínico e de Seqüência foram duplicadas em bancos de dados PostgreSQL. Os dados foram gerados pseudo-aleatoriamente, de modo que os resultados das consultas não fossem tendenciosos.

Este protótipo foi desenvolvido em Java 1.4 e testado nos seguintes ambientes:

1. $Ambiente_1$: uma máquina V880 da Sun Micro Systems com as seguintes caracterísiticas:
   - Memória Física: 32 Gb de RAM
   - Número de processadores: 8
   - Clock do processador: 400 MHz
2. $Ambiente_2$: uma máquina PC AthlonXp 2.700+ com as seguintes características:
   - Memória Física: 1 Gb de RAM
   - Número de processadores: 1
   - Clock do processador: 2.16 GHz

Os resultado apresentados neste trabalho são baseados na comparação entre os tempos de resposta de consultas envolvendo as estruturas de indexação proposta e acessos diretos aos bancos de dados. A comparação entre os

tempos é um indicativo da viabilidade da aplicação da estrutura indexadora. Para efeito de medição, a cada consulta direta efetuada no banco de dados, o gerenciador PostgreSQL e o sistema foram reinicializados.

Serão apresentados dois conjuntos de testes:

- *Teste sobre um CONTEÚDO simples* (ver seção 6.1.1): foi criado um CONTEÚDO envolvendo duas tabelas (*estudo*, *cobaia*) do banco de dados Clínico. Este CONTEÚDO possui três dimensões.

- *Teste sobre um CONTEÚDO complexo* (ver seção 6.1.2): foi criado um CONTEÚDO envolvendo nove tabelas (*experimento*, *protocolo_experimento*, *slide*, *chip*, *meta_experimento*, *expressao*, *material* e *slide_detalhe*) do banco de dados de Microarray. Este CONTEÚDO possui quatro dimensões.

A variação da complexidade do CONTEÚDO representa medição do benefício da integração. Isto significa quantificar o benefício obtido com a utilização de uma estrutura auxiliar fora do contexto do banco de dados em função aplicação de consultas SQL diretamente aos gerenciadores bancos de dados. Para medir o comportamento das estruturas propostas em função do volume de dados, cada ESPAÇO foi criado com diferentes quantidades de registros. Foi aplicado o mesmo conjunto de consultas nesses ESPAÇOS. Os resultados estão no decorrer deste cepítulo.

Cada conjunto de consultas é composto por cinco consultas ($Consulta_i$, $1 \leq i \leq 5$), definindo, assim, cinco regiões. Essas regiões foram escolhidas arbritariamente de tal forma que cada região representasse uma região total ou parcialmente nova, forçando a utilização das duas estruturas de índices.

Cada consulta $Consulta_i$ foi executada três vezes e obtidos três tempos $tempo_1$, $tempo_2$ e $tempo_3$, de modo que:

- O $tempo_1$ representa o tempo gasto pelo sistema para responder a consulta utilizando somente o Índice Integrador, já que a região R_i definida pela consulta $Consulta_i$ não está indexada pelo Índice por Consulta. Este tempo também engloba o tempo de navegação e inserção desta região na árvore do Índice por Conteúdo.

- O $tempo_2$ representa o tempo gasto pelo sistema para processar a mesma consulta $Consulta_i$. Porém, como a região R_i definida pela consulta já foi indexada na sua primeira execução ($tempo_i$), este tempo representa a busca na árvore do Índice por Conteúdo e o processamento dos identificadores contidos na folha desta árvore (esta folha representa a região R_i).

- O $tempo_3$ representa o tempo gasto na execução de uma consulta eqüivalente sem a estrutura de indexação, ou seja, a consulta SQL eqüivalente à $Consulta_i$ foi executada diretamente no banco de dados.

Os tempos foram obtidos em segundos.

6.1 Resultados

6.1.1 Resultados dos testes usando o Conteúdo Simples

O índice criado possui três dimensões. Segue a lista das regiões definidas por cada consulta efetuada.

- $Consulta_1$:

Cobaia:nomeCidade«SUDESTE_003 and
Cobaia:nomeRegiao==SUDESTE

- $Consulta_2$:

Cobaia:nomeCidade»SUDESTE_001 and
Cobaia:nomeCidade«SUDESTE_003 and
Cobaia:nomeRegiao==SUDESTE

- $Consulta_3$:

Cobaia:nomeCidade»SUDESTE_001 and
Cobaia:nomeCidade<=SUDESTE_004 and
Cobaia:nomeRegiao==SUDESTE

- $Consulta_4$:

Cobaia:nomeCidade»SUDESTE_001 and
Cobaia:nomeCidade<=SUDESTE_006 and
Cobaia:nomeRegiao==SUDESTE

- $Consulta_5$:

Cobaia:nomeCidade>=SUDESTE_003 and
Cobaia:nomeCidade<=SUDESTE_025 and
Cobaia:nomeRegiao==SUDESTE

Nome do espaço	Quantidade de Registros	Tempo (Segundos)
spc_cob_050k	50.000	32
spc_cob_150k	150.000	79
spc_cob_250k	250.000	132
spc_cob_350k	350.000	186
spc_cob_450k	450.000	235

Tabela 6.1: Tabela comparativa de tempo de criação do índice por número de registros.

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	561	1	0	5
$Consulta_2$	177	0	0	2
$Consulta_3$	567	0	1	3
$Consulta_4$	907	1	1	3
$Consulta_5$	4152	0	0	3

Tabela 6.2: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_cob_050k.*

Tempo de criação do índice

A tabela 6.1 mostra o tempo gasto para a criação do Índice Integrador sobre o CONTEÚDO simples. Devido ao fato do eixo "Espaços"não estar em escala, o gráfico apresentou uma distorção nas pontas, mas os dados mostram que o tempo de criação é linear.

Tempo de execução das consultas

Seguem as listas e gráficos ilustrando os tempos de consultas de cada $Consulta_i$.

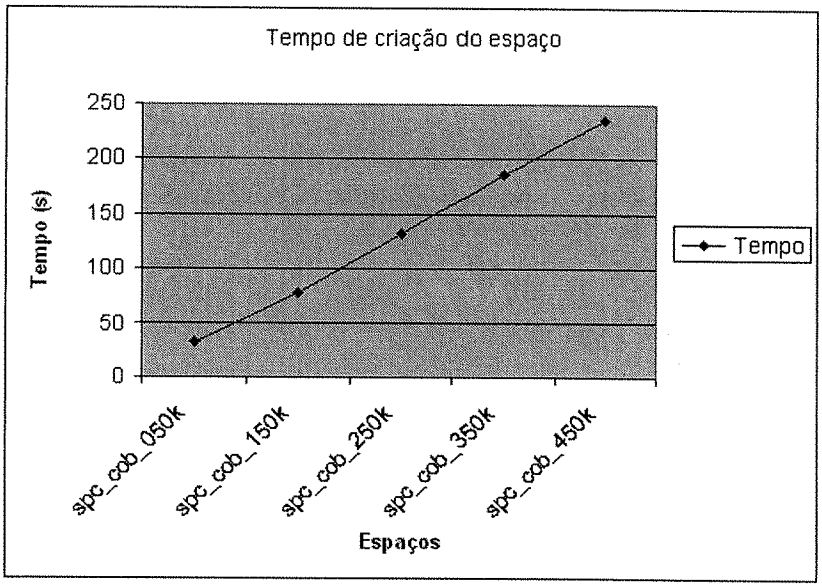

Figura 6.1: Gráfico que mostra o tempo de demora de criação de cada espaço.

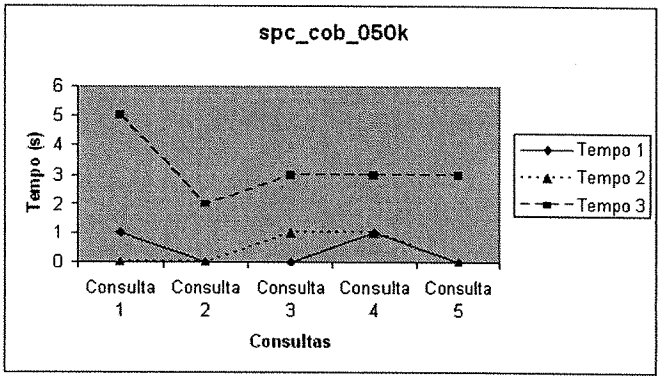

Figura 6.2: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_cob_050k*.

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	1719	1	0	6
$Consulta_2$	537	1	1	4
$Consulta_3$	1684	1	1	3
$Consulta_4$	2796	1	2	4
$Consulta_5$	12054	2	1	4

Tabela 6.3: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_cob_150k*.

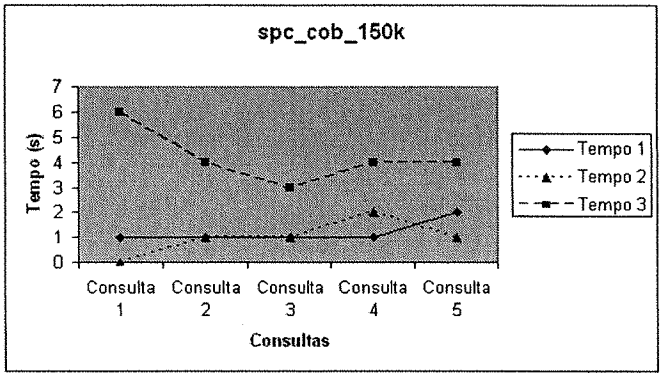

Figura 6.3: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_cob_150k*.

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	2769	2	0	4
$Consulta_2$	876	2	2	3
$Consulta_3$	2742	2	3	4
$Consulta_4$	4566	2	3	4
$Consulta_5$	20012	3	3	5

Tabela 6.4: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_cob_250k*.

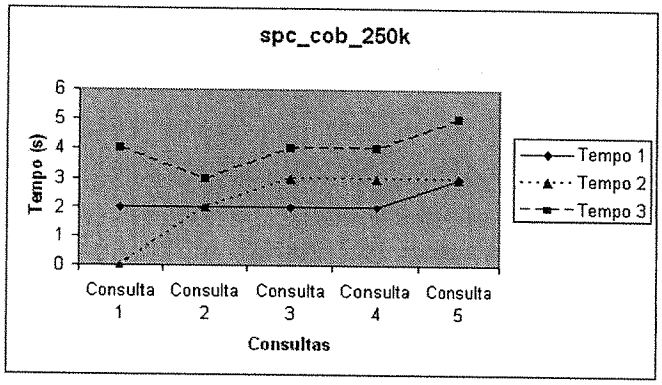

Figura 6.4: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_cob_250k*.

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	3945	3	1	19
$Consulta_2$	1272	4	4	24
$Consulta_3$	3872	4	3	4
$Consulta_4$	6465	4	4	4
$Consulta_5$	28162	5	5	6

Tabela 6.5: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_cob_350k*.

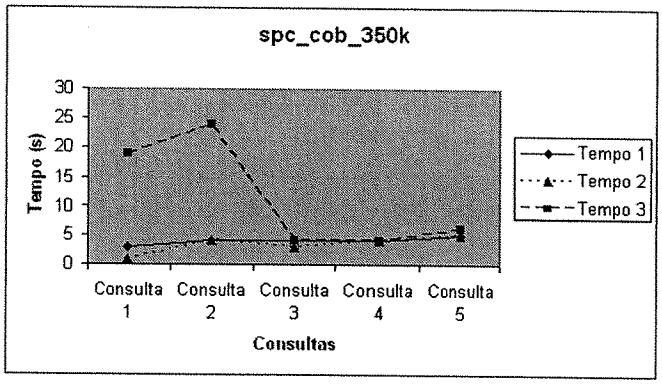

Figura 6.5: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_cob_350k*.

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	5110	4	1	25
$Consulta_2$	1670	16	5	4
$Consulta_3$	4992	4	5	5
$Consulta_4$	8331	6	8	5
$Consulta_5$	36221	6	4	15

Tabela 6.6: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_cob_450k*.

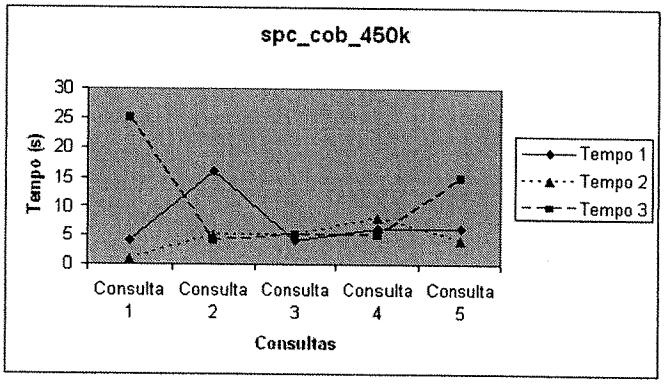

Figura 6.6: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_cob_450k*.

6.1.2 Resultados dos testes usando o Conteúdo Complexo

O índice criado possui quatro dimensões. Segue a lista das regiões definidas por cada consulta efetuada.

- $Consulta_1$:

  Expressao:valorExpressao«00000.400000000

- $Consulta_2$:

  Expressao:valorExpressao[=00000.30000,00000.400000=]

- $Consulta_3$:

  Expressao:valorExpressao[=00000.100000000,00000.1230000=]

- $Consulta_4$:

  Expressao:valorExpressao[=00000.1200000,00000.1700000=]

- $Consulta_5$:

  Expressao:valorExpressao[=00000.2500000000,00000.2600000000=]

Tempo de criação do índice

A tabela 6.7 mostra o tempo gasto para a criação do Índice Integrador sobre o CONTEÚDO complexo. Devido ao fato do eixo "Espaços"não estar em escala, o gráfico apresentou uma distorção nas pontas, mas os dados mostram que o tempo de criação é linear.

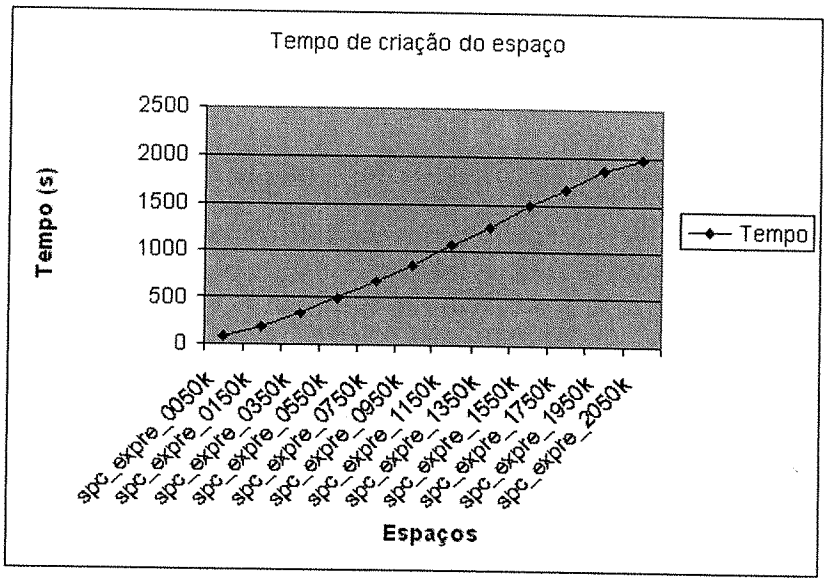

Figura 6.7: Gráfico que mostra o tempo de demora de criação de cada espaço.

Nome do Espaço	Quantidade de Registros	Tempo (Segundos)
spc_expre_0050k	50.000	91
spc_expre_0150k	150.000	186
spc_expre_0350k	350.000	331
spc_expre_0550k	550.000	495
spc_expre_0750k	750.000	684
spc_expre_0950k	950.000	849
spc_expre_1150k	1.150.000	1078
spc_expre_1350k	1.350.000	1259
spc_expre_1550k	1.550.000	1499
spc_expre_1750k	1.750.000	1657
spc_expre_1950k	1.950.000	1858
spc_expre_2050k	2.050.000	1984

Tabela 6.7: Tabela comparativa de tempo de criação do índice por número de registros

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	2039	0	0	21
$Consulta_2$	528	0	0	53
$Consulta_3$	104	0	0	25
$Consulta_4$	256	0	0	36
$Consulta_5$	51	0	0	22

Tabela 6.8: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_expre_0050k*.

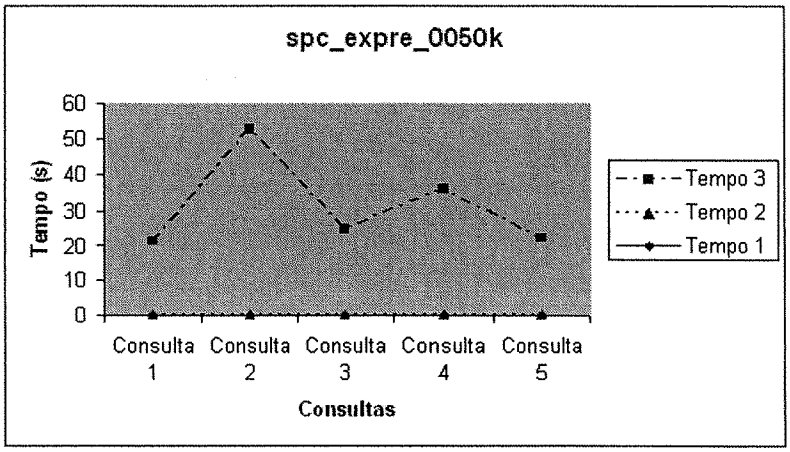

Figura 6.8: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_expre_0050k*.

Tempo de execução das consultas

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	5925	0	1	22
$Consulta_2$	1465	0	0	56
$Consulta_3$	318	0	0	40
$Consulta_4$	768	1	0	69
$Consulta_5$	148	0	0	29

Tabela 6.9: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_ expre_ 0150k.*

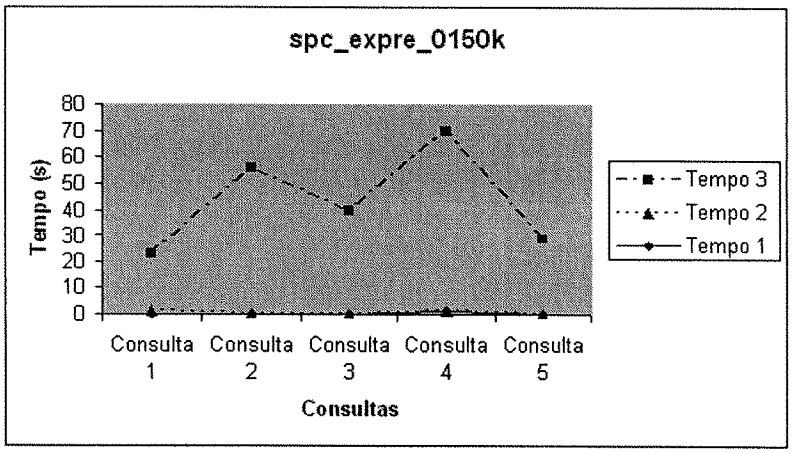

Figura 6.9: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_ expre_ 0150k.*

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	13832	1	0	24
$Consulta_2$	3451	0	0	246
$Consulta_3$	767	0	0	69
$Consulta_4$	1707	0	0	131
$Consulta_5$	349	0	0	42

Tabela 6.10: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_ expre_ 0350k.*

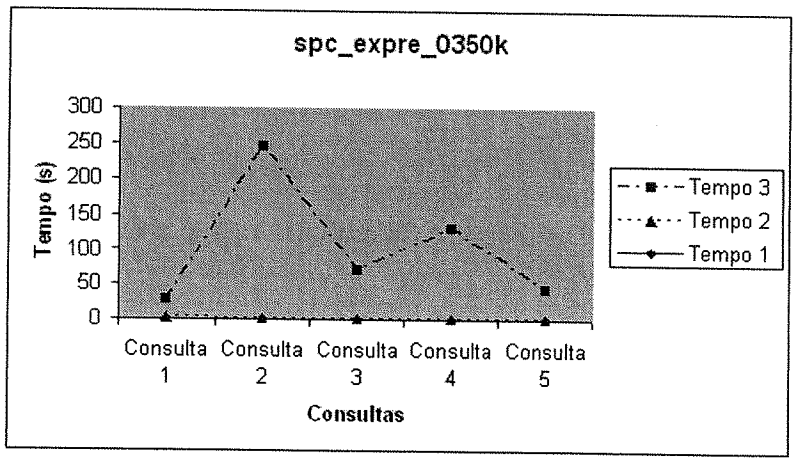

Figura 6.10: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_expre_0350k*.

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	21783	1	0	26
$Consulta_2$	5412	0	0	378
$Consulta_3$	1232	1	0	109
$Consulta_4$	2687	0	0	195
$Consulta_5$	556	0	0	55

Tabela 6.11: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_expre_0550k*.

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	29673	2	0	28
$Consulta_2$	7400	1	0	508
$Consulta_3$	1676	0	0	129
$Consulta_4$	3614	0	0	256
$Consulta_5$	759	0	0	69

Tabela 6.12: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_expre_0750k*.

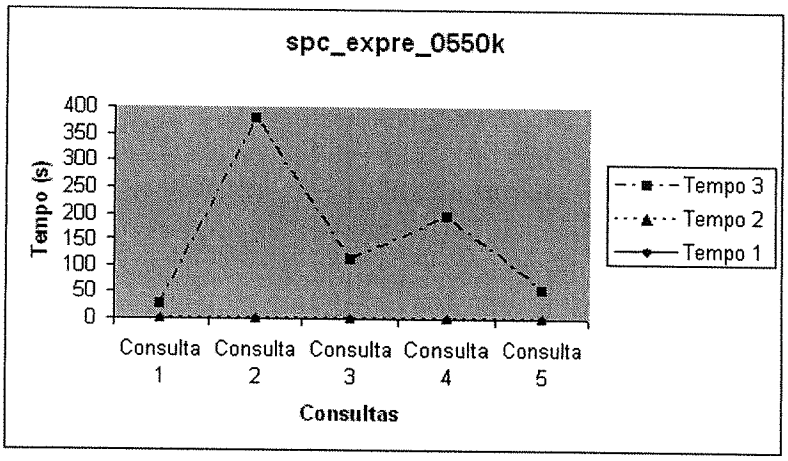

Figura 6.11: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_ expre_ 0550k*.

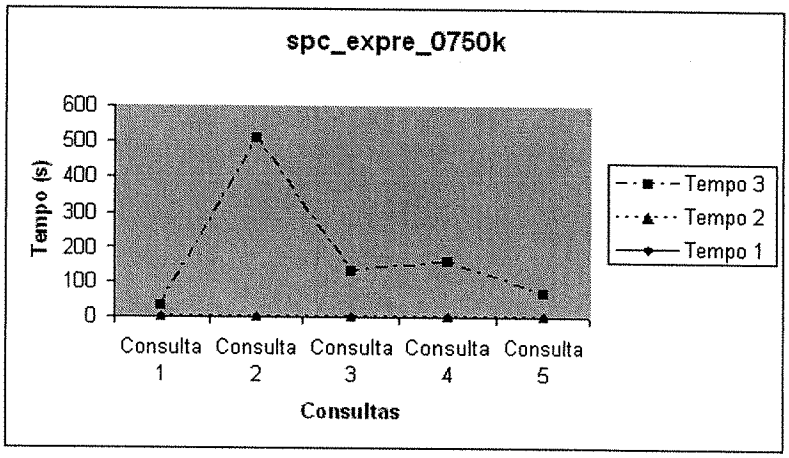

Figura 6.12: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_ expre_ 0750k*.

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	37647	5	1	29
$Consulta_2$	9365	0	0	845
$Consulta_3$	2131	0	0	111
$Consulta_4$	4585	0	0	325
$Consulta_5$	953	0	0	77

Tabela 6.13: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_ expre_ 0950k*.

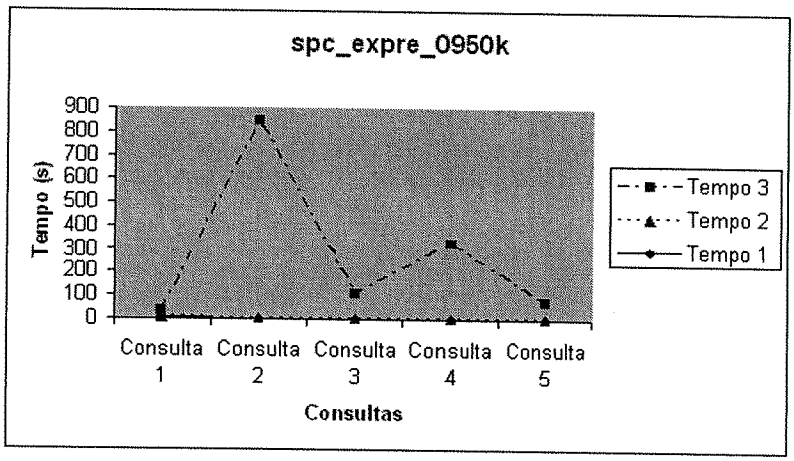

Figura 6.13: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_ expre_ 0950k*.

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	41606	2	1	31
$Consulta_2$	10408	0	0	937
$Consulta_3$	2356	0	0	176
$Consulta_4$	5049	0	0	353
$Consulta_5$	1037	1	0	88

Tabela 6.14: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_ expre_ 1050k*.

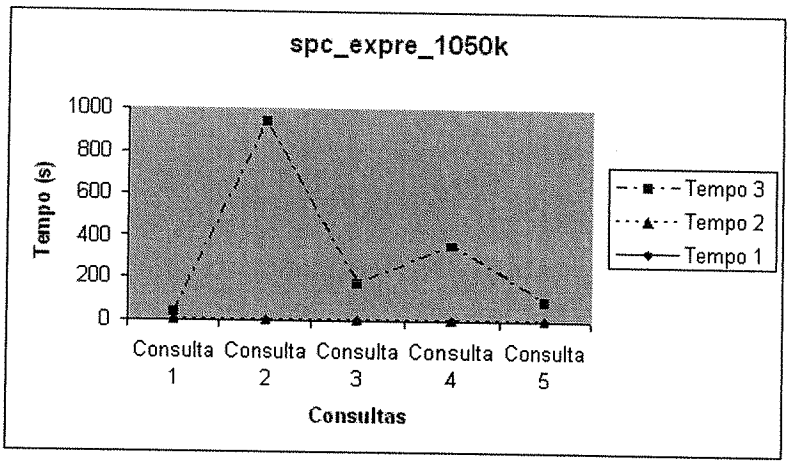

Figura 6.14: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_ expre_ 1050k*.

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	45537	2	1	31
$Consulta_2$	11423	1	0	1155
$Consulta_3$	2560	1	0	190
$Consulta_4$	5537	1	0	385
$Consulta_5$	1142	0	1	95

Tabela 6.15: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_ expre_ 1150k*.

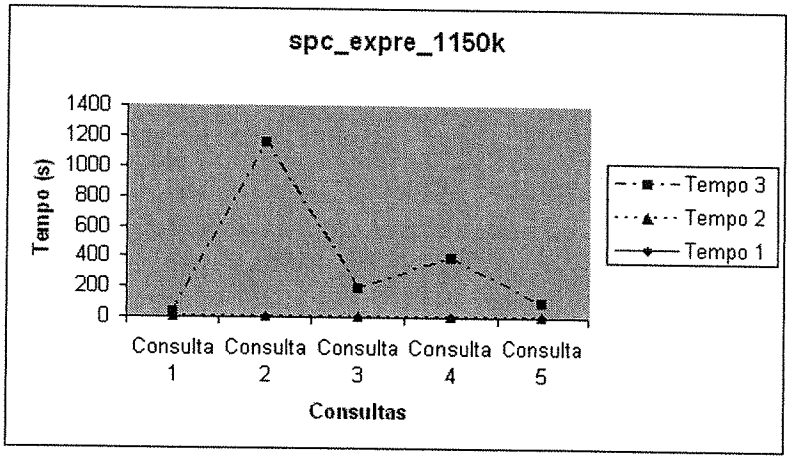

Figura 6.15: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_ expre_ 1150k.*

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	53434	7	1	33
$Consulta_2$	13448	0	0	1207
$Consulta_3$	2998	0	0	219
$Consulta_4$	6563	0	1	573
$Consulta_5$	1339	0	0	112

Tabela 6.16: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_ expre_ 1350k.*

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	61560	2	1	34
$Consulta_2$	15418	3	0	1377
$Consulta_3$	3496	0	0	252
$Consulta_4$	7523	1	1	684
$Consulta_5$	1534	0	0	123

Tabela 6.17: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_ expre_ 1550k.*

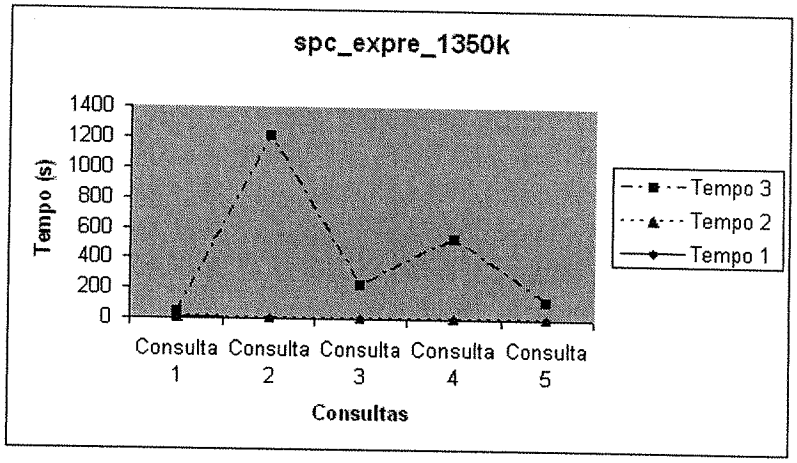

Figura 6.16: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_expre_1350k*.

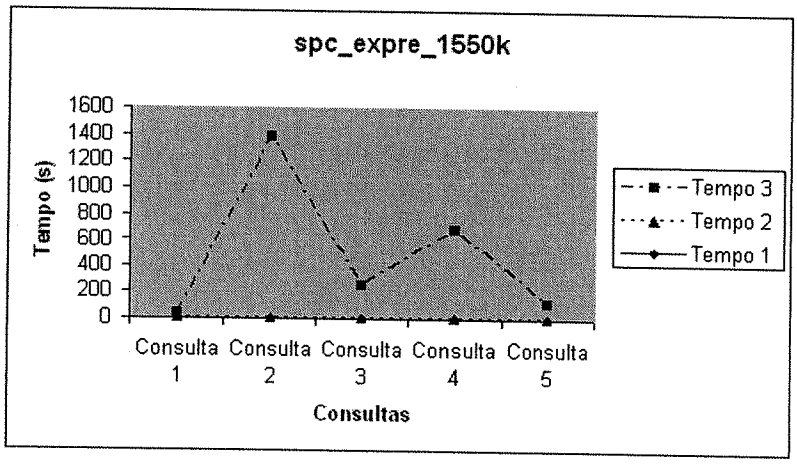

Figura 6.17: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_expre_1550k*.

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	69492	2	2	36
$Consulta_2$	17402	0	0	1554
$Consulta_3$	3926	1	0	279
$Consulta_4$	8523	4	0	771
$Consulta_5$	1738	0	0	136

Tabela 6.18: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_expre_1750k*.

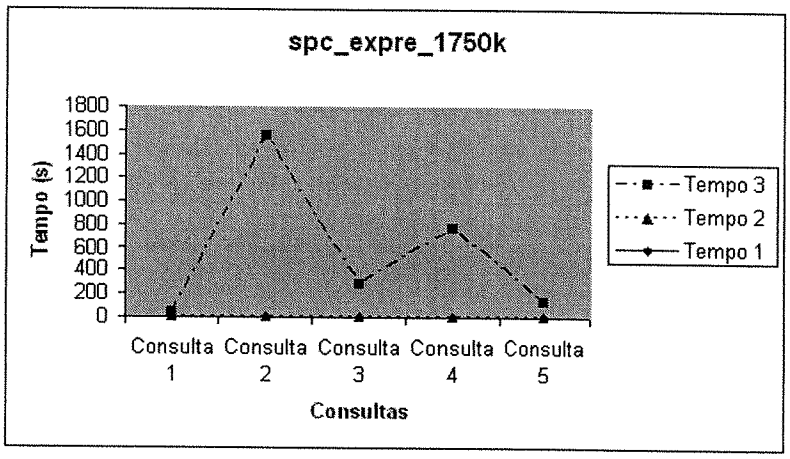

Figura 6.18: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_expre_1750k*.

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	77664	3	1	37
$Consulta_2$	19437	1	0	1741
$Consulta_3$	4365	0	0	310
$Consulta_4$	9518	1	0	966
$Consulta_5$	1955	0	0	151

Tabela 6.19: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_expre_1950k*.

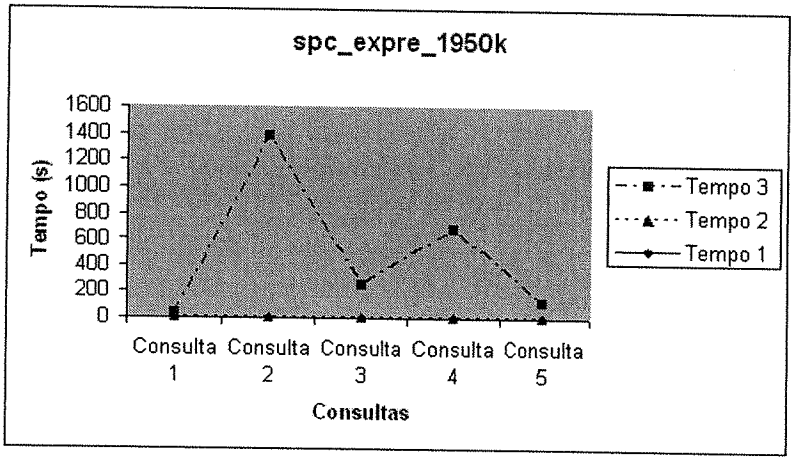

Figura 6.19: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_ expre_ 1950k.*

	Qtde de Registros Recuperados	$Tempo_1$	$Tempo_2$	$Tempo_3$
$Consulta_1$	81570	3	1	39
$Consulta_2$	20474	1	0	1829
$Consulta_3$	4588	0	0	321
$Consulta_4$	10018	1	0	901
$Consulta_5$	2062	1	0	157

Tabela 6.20: Esta tabela mostra o tempo gasto de cada consulta usando o espaço *spc_ expre_ 2050k.*

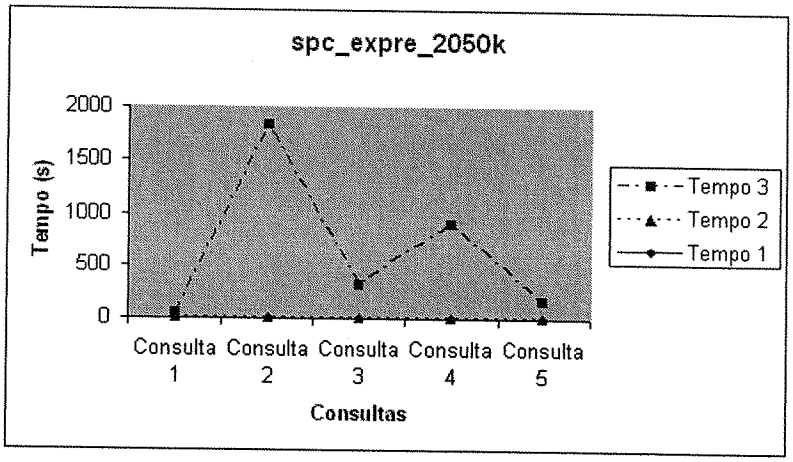

Figura 6.20: Gráfico que mostra o tempo de demora de cada consulta para o espaço *spc_expre_2050k*.

6.2 Análise

O desempenho das estruturas de índices nos testes em PC foram semelhantes aos testes na SUN para índices com quantidade pequena de registros (entende-se por pequeno índices com menos que 750.000 registros e menos que 3 dimensões). Isso pode ser justificado considerando o fato de que tanto o Índice Integrador com o Índice por Conteúdo utilizarem apenas memória RAM.

A seguir estão detalhadas as análises dos testes sobre os dois tipos de CONTEÚDOS.

6.2.1 Análise dos testes para o CONTEÚDO simples

As tabelas e gráficos apresentados na seção 6.1.1 referem-se aos tempos obtidos na execução dos testes para o CONTEÚDO simples.

O gráfico 6.2 mostra o desempenho das consultas num espaço com 50.000 registros. É possível notar que existe um ganho pequenho de desempenho (na ordem de segundos). Já no gráfico 6.6, nota-se que o aumento de volume de dados provoca a diminuição do tempo ganho. Isto evidencia que para CONTEÚDOS SIMPLES, este índice tem apenas o papel de integrador de dados.

O gráfico 6.1 mostra que o tempo gasto na criação do Índice Integrador é proporcional a quantidade de registro e a quantidade de dimensões. Fato que comprova a viabilidade do processo de criação.

O uso do Índice Integrador não representa um ganho significativo de tempo nas consultas em CONTEÚDOS simples, contudo, se mostrou como uma ótima alternativa para integração de bancos de dados.

O uso do Índice por Conteúdo não alterou significativamente o tempo das consultas mas sua importância aumenta de acordo com o crescimento do número de dimensões e do volume de dados. A árvore do Índice por Conteúdo armazena as regiões já consultadas.

6.2.2 Análise dos testes para o CONTEÚDO complexo

Através das tabelas e dos gráficos de tempo de execução das consultas dos CONTEÚDOS complexos apresentados na seção 6.1.2 é possível verificar que o Índice Integrador melhora significativamente o tempo de consulta em relação a consulta direta sobre os bancos de dados, mesmo com a otimização das consultas nos bancos de dados operacionais. Isso pode ser comparado nas análises das colunas $tempo_1$ e $tempo_3$ de cada consulta $Consulta_i$.

Também é possível verificar que a camada de Índice por Conteúdo, medido pelo $tempo_2$, melhora o tempo de consulta, apesar de não ser tão significativo quanto ao ganho de performance obtido com o Índice Integrador.

Em função do escopo do protótipo proposto nesse trabalho, um fator limitante dessa implementação é o fato do Índice Integrador ser implementado em memória. Isso limita o tamanho do índice, tanto em relação a quantidade de registros como em relação ao número de dimensões. No $Ambiente_2$, foi constatado que o maior índice possível de ser criado possui 8 dimensões e 750.000 registros. Já no $Ambiente_1$, o maior índice possível de ser criado possui 15 dimensões e 1.500.000 registros.

Pela análise do grágico 6.7, o tempo de criação do Índice Integrador cresce linearmente com o aumento da quantidade de registros e de dimensões.

O uso do Índice Integrador representa um ganho significativo de tempo nas consultas em CONTEÚDOS complexos e se mostrou como uma ótima alternativa para integração de bancos de dados.

O uso do Índice por Conteúdo representa um ganho no número de operações sobre as árvores do ESPAÇO a medida que, se a região da consulta já estiver indexada pela árvore do Índice por Conteúdo, os identificadores dos objetos dos bancos de dados primários já estarão pré-selecionados.

Capítulo 7

Conclusão

O propósito desse trabalho foi de criar uma camada de índice para a integração de bancos de dados e indexação de conteúdo de consultas. Para definição destes conceitos foram feitos levantamentos bibliográficos de índices para bancos de dados orientado a objetos (capítulo 2), índices para espaços vetoriais (capítulo 3) e índices para espaços métricos (capítulo 4).

A abordagem dada para o problema de integração de bancos de dados foi a criação de uma camada sistêmica de índices externos. Essa camada fornece uma linguagem de consultas semelhante ao consagrado SQL, detalhes desta linguagem pode ser visto no apêndice A. O seu núcleo é composto por duas estruturas de índices chamadas Índice Integrador e Índice por Conteúdo. o Índice Integrador cria ESPAÇOS (ver capítulo 5) vetoriais de consultas e as sub-regiões definidas pelos predicados das consultas são indexados pelo Índice por Conteúdo. O sistema está descrito no capítulo 5.

O sistema proposto foi implementado sobre a estrutura de bancos de dados definidos pelo trabalho [BCJFG03]. A metodologia e resultados dos testes podem ser visto no capítulo 6.

Com os resultados obtidos nos testes de performance do Índice Integrador é possível validar sua eficiência na integração de bancos de dados. Foi comprovado que quanto maior a heterogeneidade do ambiente, maior será sua importância. O espaço vetorial definido pelo Índice Integrador é indexado pelo Índice por Conteúdo que armazena hierarquicamente regiões de dados defindos pelos predicados das consultas feitas sobre este ESPAÇO. Issa abordagem permite que as regiões já consultadas sejam respondidas mais rapidamente e permite analisar o comportamento das consultas dos usuários.

7.1 Contribuições

Como contribuição desse trabalho é possível citar:

- A validação do conceito de índices externos para consultas globais. Isto é, a criação de estruturas de dados de indexação especializada fora dos bancos de dados operacionais, permitindo a integração dos bancos de dados heterogêneos e a realização de consultas globais sem o prévio conhecimento das estruturas internas de cada banco de dados. O Índice Integrador se propõe a resolver consultas por regiões em espaços n-dimensionais.

  Na implementação proposta, foi definido um conceito chamado ESPAÇO (ver capítulo 5) que contém n-dimensões. Cada dimensão contém uma árvore-B formada pelos valores que o atributo assume no banco de dados operacional. Nas folhas dessa árvore-B estão os identificadores dos objetos que assumem este valor no seu atributo.

- A introdução do conceito de índice por conteúdo de consulta (Indice por Conteúdo), no qual os dados são agrupados por regiões de acordo com as consultas feitas pelos usuários. O Índice por Conteúdo é construído dinamicamente e indexa os espaços segundo o significado semântico da consulta. Este conceito permite criar um índice que registre as regiões mais consultadas de tal modo que priorize as respostas das consultas feitas nestas regiões.

  A implementação desta estrutura foi baseada na árvore-R, já que os predicados de cada consulta representam regiões definidas no espaço considerado. Os algoritmos de busca e inserção estão descritos na seção 5.3.4. Sua performance foi analisada em 6.2.

7.2 Futuras pesquisas

Como próximos passos do trabalho pode-se listar os seguintes tópicos:

- O projeto definido em [BCJFG03], pressupõe em uma segunda etapa, a introdução do dados de objetos complexos, isto é, objetos como imagens e estruturas de DNA. Como visto no capítulo 4, a partir do momento que se trabalha com objetos complexos, o tipo de consulta mais comum é por similaridade. Portanto, existe a necessidade de modificar o Índice Integrador e o Índice por Conteúdo para atender à esses tipos de pesquisas eficientemente.

- Como as consultas são indexadas de acordo com os predicados estipulados pelos usuários, a árvore gerada não é balanceada, fato que pode comprometer a performance das consultas, necessitando de re-arranjos periódicos.

- O Índice integrador não é dinâmico, isto é, os dados são obtidos no momento da criação do índice e não se modificam de acordo com atualizações nos bancos de dados. Isto pode ser contornado de duas formas: recriar o índice periodicamente ou criar mecanismos de atualização, refletindo modificações nos bancos de dados. A recriação do índice pode ser viável para espaços com que envolvam poucos registros, mas quando se trata de espaços com muitas dimensões de grande quantidade de registros, deve-se criar um mecanismo de atualização, vinculado com o banco de dados.

- Outro trabalho na área de inferência de comportamento das consultas é utilizar a árvore gerada pelo Índice por Conteudo para determinar comportamentos de consultas. Pode-se acrescentar o fator tempo nos nós destas árvores tornar possível analise da sazonalidade do comportamento das consultas.

Apêndice A

Comandos do sistema

A.1 Comandos de sistema

São comandos que alteram o comportamento do sistema.

- **BEGIN** : inicia um comando. É empregado na edição de comandos longos, com múltiplas linhas.
- **END** : finaliza um comando BEGIN, para indicar que o comando acabou.
- **LOAD [ALL|<nome espaço>]** : carrega a configuração do sistema descrito no arquivo ListIndex.cfg, a árvore e a lista de consultas já efetuadas. O arquivo ListIndex.cgf descreve todos os conteúdos existentes e os espaços criados.
- **SAVE [ALL|<nome espaco>]** : Grava a configuração do sistema no arquivo ListIndex.cfg, a árvore, a lista de consultas já efetuadas e os espaços abertos.
- **SHOW [STATISTICS|RESULTS|TREETRACE] [ON|OFF]** : indica ao sistema se deve mostrar ou não as informações sobre tempos de consultas (STATISTICS), os resultados do comando SELECT (RESULTS) e manipulação da árvore, como procurar e inserir (TREETRACE).

A.2 Comandos de manipulação de estrutura

São comandos de criação, alteração e remoção das estruturas dos objetos.

- **CLEAR [TREE|SELECT|ALL]** : inicializa estrutura interna como a árvore (TREE) e a lista de consultas já efetuadas (SELECT)
- **CLOSE <nome do espaco>** : tira da memória o espaço especificado.
- **CREATE CONCEPT <nome> ON [<conteúdo>:<atributo>]*** : cria o conteúdo
- **CREATE SPACE <nome> ON [<conteúdo>:<atributo>]*** : cria o espaço especificado com os conteúdos/atributos especificados como dimensão.
- **DROP SPACE <nome>** : remove o espaço especificado.
- **LIST CONCEPT** : lista todos os conteúdos do sistema.
- **LIST METAMODEL** : lista todos os meta-modelos[1] do sistema.
- **LIST SELECT** : lista todas as consultas já efetuadas no sistema.
- **LIST SPACE** : lista todos os espaços do sistema.
- **LIST TREE** : lista os nós da árvore.
- **OPEN <nome do espaço>** : carrega em memória o espaço especificado.
- **REFRESH [ALL|<nome do espaco>]** : atualiza o espaço especificado (ou todos) com informações dos bancos de dados.
- **USE METAMODEL** : ativa o meta-modelo especificado.

A.3 Comandos de manipulação de dados

O comando SELECT e SELECTDISTINCT recuperam os valores do sistema. Sua sintaxe é semelhante ao SQL ANSI (padrão da maioria dos bancos de dados relacionais) a não ser pelo fato de ser necessário a definição do ESPACO a ser utilizado ou, no caso de **none**, buscar diretamente do banco de dados. As sintaxes possíveis para os operadors estão em A.4.

- **SELECT** [<conteúdo:atributo>]* FROM <conteúdo>=[none|<nome do espaço>] WHERE <conteúdo:atributo><operador>

[1] Um meta-modelo define um conjunto de CONTEÚDOS e ESPAÇOS

- **SELECTDISTINCT** [<conteúdo:atributo>]* FROM <conteúdo>=[none|<nome do espaço>] WHERE <conteúdo:atributo><operador>

A.4 Operadores definidos para os predicados

Seja $A.a$ representando o atributo **a** do objeto **A** e *v1* e *v2* dois possíveis valores de A.a, os operadores definidos são:

1. A.a <= v1 : significa todos os valores de $A.a$ menores ou iguais a *v1*.
2. A.a « v1 : significa todos os valores de $A.a$ estritamente menores que *v1*.
3. A.a >= v1 : significa todos os valores de $A.a$ maiores ou iguais a *v1*.
4. A.a » v1 : significa todos os valores de $A.a$ estritamente maiores que *v1*.
5. A.a <> v1 : significa todos os valores de $A.a$ diferentes de *v1*.
6. A.a [=v1,v2=] : significa todos os valores de $A.a$ maiores ou iguais a *v1* e menores ou iguais a *v2*.
7. A.a [=v1,v2]] : significa todos os valores de $A.a$ maiores ou iguais a *v1* e estritamente menores que *v2*.
8. A.a [[v1,v2=] : significa todos os valores de $A.a$ estritamente maiores que *v1* e menores ou iguais a *v2*.
9. A.a [[v1,v2]] : significa todos os valores de $A.a$ estritamente maiores que *v1* e estritamente menores que *v2*.

Referências Bibliográficas

[BCJFG03] J. Barrera, R. M. Cesar-Jr, J. E. Ferreira, and M. D. Gubitoso. An enviroment for knowledge discovery in biology. *Journal of Computers in Biology and Medicine*, 2003.

[BER85] D. A. Beckley, M. W. Evens, and V. K. Raman. Multikey retrieval from k-d trees and quad-trees. *ACM SIGMOD*, pages 291–301, 1985.

[Ber93] E. Bertino. *Object-Oriented Database System: Concepts and Architecture.* Addison Wesley, 1993.

[Ber94] E. Bertino. On indexisng configuration in object-oriented databases. *VLDB Journal*, 1994.

[BF95] E. Bertino and P. Foscoli. Index organization for object-orientes database systems. *IEEE Trnsaction on Knowledge and Data Engeneering*, 1995.

[BG93] E Bertino and C Guglielmina. Path-index: An approach to the efficient execution of object-oriented queries. *Data and Knowledge Engeneering*, 1993.

[BK73] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Communications of ACM*, 1973.

[BK89] E. Bertino and W. Kim. Indexing techniques for queries on nested objects. *IEEE Transaction on Knowledge and Data Engeneering*, 1989.

[BR90] P. Bodorik and J. S. Riordon. System integration in multi-databases. *ACM*, 1990.

[Bri95] S. Brin. Near neighbor search in large metric spaces. In *The VLDB Journal*, pages 574–584, 1995.

[BYE99] R. Orlandic B. Yu and M. Evens. Simple qsf-trees: an efficient and scalable spatial access method. In *Proceedings of the eighth international conference on Information and knowledge management*, pages 5–14. ACM Press, 1999.

[Bz97] T. Bozkaya and Z. M. Özsoyoglu. Distance-based indexing for high-dimensional metric space. *ACM-SIGMOD*, 1997.

[Cla97] K. Clarkson. Nearest neighbor queries in metric spaces. *STOC*, 1997.

[CPRZ97] P. Ciaccia, M. Patella, F. Rabitti, and P. Zezula. Indexing metric spaces with m-tree. In *Sistemi Evolui per Basi di Dati*, pages 67–86, 1997.

[DKR00] K. C. Davis, U. T. Kang, and S. Ravishamkar. Indexing inheritence and aggregation. *9th International conference on Information and Knowledge Management*, 2000.

[DMAW85] A. M. Tenenbaum D. M. Arnow and C. Wu. P-trees: storage efficient multiway trees. In *Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 111–121. ACM Press, 1985.

[ea97] C Zaniolo; et al. *Advanced Database Systems.* Morgan Kaufmann Publishers, 1997.

[ECM01] R. Baeza-Yates E. Chaves, G. Navarro and J. L. Marroquin. Searching in metric spaces. *ACMComput. Surv. 33*, pages 273Ũ–322, 2001.

[EN00] R. Elmasri and S. B. Navathe. *Database Systems: Fundamental of Database System.* Addison Wesley, 2000.

[FF00] J. E. Ferreira and M. Finger. . *Controle de Concorrência e Distribuição de Dados: Teoria Clássica, suas Limitações e Extensões Modernas.* XII Escola de Computação, 2000.

[FKL00] C. Fung, K. Karlapalem, and Q Li. Complex object retrieval via structural join index hierarchy mechanisms. *9th International conference on Information and Knowledge Management*, 2000.

[FL95] C. Faloutsos and K. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174, San Jose, California, 22–25 1995.

[Fre87] M. Freeston. The bang file: a new kind of grid-file. *ACM*, 1987.

[FTJF01] R. F. Santos Filho, A. J. M. Traina, C. Traina Jr., and C. Faloutsos. Similarity search without tears: The OMNI family of all-purpose access methods. In *ICDE*, pages 623–630, 2001.

[GBM00] Genbank mirror usp - http://cagedb.ime.usp.br/cage, 2000.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[GSVGM98] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity search in databases. *VLDB conference*, 1998.

[Gun86] O. Gunther. The cell tree: an index for geometric data. *Technical Report - Berkley*, 1986.

[HS03a] G. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5), 2003.

[HS03b] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.*, 28(4):517–580, 2003.

[HSW88] A. Hutflesz, H. Six, and P. Widmayer. Twin grid files: space optimizing access schemes. In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 183–190. ACM Press, 1988.

[ICI01] J. E. Ferreira e O. K. Takai I. C. Italiano. *Aspectos Conceituais em Data Warehouse.* Relatório Técnico, 2001.

[int86] Genbank - www.ncbi.nlm.nih.gov/genbank, 1986.

[int02] Genbank mirror - www.ncbi.nlm.nih.gov/genbank, 2002.

[JNS84] H. Hinterberger J. Nievergelt and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. on Database Systems*, 1984.

[KCK01] K. Kim, K. Cha, and K. Kwon. Optimizing multidimensional index trees for main memory access. *ACM SIGMOD - Management of Data*, 2001.

[Kim82] W. Kim. On optimizing an sql-like nested query. *ACM Trans. Database Syst.*, 7(3):443–469, 1982.

[KKD89] W. Kim, K. Kim, and A. Dale. *In Object-Oriented Concepts, Database and Applications.* Addison-Wesley, 1989.

[KKM92] A. Kemper, C. Kilger, and G. Moerkotte. Access suport relations: An indexing method for object bases. *Information Systems*, 1992.

[KM94] C. Kilger and G. Moerkotte. Indexing multiple sets. *In Proc. 20th International Conference on Very Large Data Bases*, 1994.

[LOL92] C C Low, B C Ooi, and H. Lu. H-trees: A dynamic associative search index for oobd. *ACM SIGMOD International Conference on Management of Data.*, 1992.

[LS90] D. B. Lomet and B. Salzberg. The hb-tree: a multiattribute indexing method with good guaranteed performance. *ACM Trans. Database Syst.*, 15(4):625–658, 1990.

[MS86] D. Maier and J. Stein. Indexing in an object-oriented database. *IEEE Workshop on Object-Oriented DBMSs*, 1986.

[NBS90] R. Schneider N. Beckmann, H. Kriegel and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 322–331. ACM Press, 1990.

[NS98] M. A. Nascimento and J R. O. Silva. Towards historical r-trees. *ACM Symposium on Applied Computing*, 1998.

[PAAV02] O. Procopiuc, P. Agarwal, L. Arge, and J. Vitter. Bkd-tree: A dynamic scalable kd-tree, 2002.

[PCZ97] M. Patella P. Ciaccia and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. *VLDB*, 1997.

[RAS03] A. Evfimievski R. Agrawal and R. Srikant. Information sharing across private databases. *ACM - SIGMOD*, 2003.

[Sam95] H. Samet. Spatial data structures. In *Modern Database Systems: The Object Model, Interoperability and Beyond*, pages 361–385. 1995.

[SBK96] D. A. Keim S. Berchtold and H. Kriegel. The x-tree: An index structure for high-dimensional data. *VLDB*, pages 29–39, 1996.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r^+-tree: A dynamic index for multi-dimensional objects. 1987.

[SS94] B. Sreenath and S. Seshadri. The hcc-tree: An efficient index structure for object oreinted databases. *In Proc. 20th International Conference on Very Large Data Bases*, 1994.

[SWY75] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

[TS82] M. Tamminen and R. Sulonen. The excell method for efficient geometric access to data. In *Proceedings of the nineteenth design automation conference*, pages 345–351. IEEE Press, 1982.

[TS96] Y. Theodoridis and T. Sellis. A model for the prediction of r-tree performance. *ACM - PODS*, 1996.

[TTFF02] C. Traina, Jr., A. Traina, R. S. Filho, and C. Faloutsos. How to improve the pruning ability of dynamic metric access methods. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 219–226. ACM Press, 2002.

[TTSF00] C. Traina Jr., A. Traina, B. Seeger, and C. Faloutsos. Slim-Trees: High performance metric trees minimizing overlap between nodes. *Lecture Notes in Computer Science*, 1777:51–??, 2000.

[Uhl84] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *ACM Trans. on Database Systems*, 1984.

[Val87] P. Valduriez. Join indices. *ACM TRansaction on Database Siystems*, 1987.

[WR84] S. K. M. Wong and V. V. Raghavan. Vector space model of information retrieval: a reevaluation. In *Proceedings of the 7th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 167–185. British Computer Society, 1984.

[WWY92] Z. W. Wang, S. K. M. Wong, and Y. Y. Yao. An analysis of vector space models based on computational geometry. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 152–160. ACM Press, 1992.

[WZRW86] S. K. M. Wong, W. Ziarko, V. V. Raghavan, and P. C. N. Wong. On extending the vector space model for boolean query processing. In *Proceedings of the 9th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 175–185. ACM Press, 1986.

[WZRW87] S. K.M. Wong, W. Ziarko, V. V. Raghavan, and P. C.N. Wong. On modeling of information retrieval concepts in vector spaces. *ACM Trans. Database Syst.*, 12(2):299–321, 1987.

[WZW85] S. K. M. Wong, W. Ziarko, and P. C. N. Wong. Generalized vector spaces model in information retrieval. In *Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 18–25. ACM Press, 1985.

[XH94] Z. Xie and J. Han. Join index hierarchy for supporting efficient navigation in object-oriented databases. *VLDB International conference*, 1994.

[XZY03] J. X. Yu X. Zhou, G. Wang and G. Yu. M+-tree: a new dynamical multidimensional index for metric spaces. In *Proceedings of the Fourteenth Australasian database conference on Database technologies 2003*, pages 161–168. Australian Computer Society, Inc., 2003.

[Yia93] Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1993.