UNIVERSIDADE FEDERAL DE SÃO CARLOS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Thiago Luís Lopes Siqueira

SB-INDEX: UM ÍNDICE ESPACIAL BASEADO EM BITMAP PARA DATA WAREHOUSE GEOGRÁFICO

São Carlos

Agosto/2009

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

SB-Index: Um Índice Espacial Baseado em Bitmap para Data Warehouse Geográfico

THIAGO LUÍS LOPES SIQUEIRA

Orientador: Ricardo Rodrigues Ciferri

São Carlos
Agosto/2009

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

S618si

Siqueira, Thiago Luís Lopes.
SB-Index : um índice espacial baseado em bitmap para data warehouse geográfico / Thiago Luís Lopes Siqueira. -- São Carlos : UFSCar, 2009.
118 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2009.

1. Banco de dados. 2. Data warehouse. 3. Indexação. I. Título.

CDD: 005.74 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"SB-index: Um Índice Espacial Baseado em Bitmap para Data Warehouse Geográfico"

THIAGO LUÍS LOPES SIQUEIRA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Ricardo Rodrigues Ciferri

Prof. Dr. Ricardo Rodrigues Ciferri
(Orientador - DC/UFSCar)

Marilde Santos

Profa. Dra. Marilde Terezinha Prado Santos
(DC/UFSCar)

Prof. Dr. Caetano Traina Junior
(ICMC/USP)

Profa. Dra. Claudia Maria Bauzer Medeiros
(IC/UNICAMP)

São Carlos
Agosto/2009

Dedico e consagro esta dissertação ao Rei dos reis: Jesus.

Em memória do Prof. Dr. Mauro Biajiz.

Agradecimentos

Digno és, Senhor, de receber glória, e honra, e poder; porque tu criaste todas as coisas, e por tua vontade são e foram criadas [Ap 4:11]. Porque desde a antiguidade não se ouviu, nem com ouvidos se percebeu, nem com os olhos se viu um Deus além de ti que trabalha para aquele que nele espera [Is 64:4].

À minha abençoada família, que me apoiou e incentivou desde sempre, e que com amor me motivou a superar os obstáculos.

À minha querida namorada Cris, pelo companheirismo, pelo carinho e pela compreensão.

A todas as pessoas que estiveram envolvidas nesta jornada, com maior ou menor intensidade. Seria injusto enumerar apenas algumas delas. Obrigado pela amizade, que certamente é uma benção [Mc 2, 1-12].

Ao estimado Prof. Dr. Ricardo Rodrigues Ciferri, pela oportunidade de trabalharmos, pela confiança, pelo carisma e pelo investimento neste que ingressou como aluno e agora termina como professor.

Às competentíssimas docentes Cristina Dutra de Aguiar Ciferri, Valéria Cesário Times e Marilde Terezinha Prado Santos, por todas as orientações e conselhos, que me proporcionaram grande crescimento profissional.

Ao DC/UFSCar por estabelecer um ambiente de trabalho muito agradável.

Ao GestaFUV/Unesp, pela primeira chance de mostrar o meu trabalho, e por serem os primeiros a acreditar em minha carreira acadêmica.

Ao IFSP pelo acolhimento e pelo estabelecimento de um horário de trabalho flexível, o qual foi determinante para concluir esta dissertação com qualidade.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, ao Instituto Nacional de Estudos e Pesquisas Educacionais e ao Projeto Web-PIDE, pelo auxílio financeiro.

Resumo

O *Data Warehouse Geográfico* (DWG) tornou-se uma das principais tecnologias de suporte à decisão, pois promove a integração de data warehouses, *On-Line Analytical Processing* e Sistemas de Informações Geográficas. Por isso, um DWG viabiliza a análise espacial aliada à execução de consultas analíticas multidimensionais envolvendo enormes volumes de dados. Por outro lado, existe um desafio relativo ao desempenho no processamento de consultas, que envolvem janelas de consulta espaciais *ad-hoc* e várias junções entre tabelas. Claramente, mecanismos para aumentar o desempenho do processamento de consultas, como as estruturas de indexação, são essenciais. Nesta dissertação, propõe-se um novo índice para DWG chamado SB-index, baseado no Índice Bitmap de Junção e no Retângulo Envolvente Mínimo. O SB-index herda todo o legado de técnicas do Índice Bitmap e o introduz no DWG. Além disso, ele provê suporte a hierarquias de atributos espaciais predefinidas. Este índice foi validado por meio de testes experimentais de desempenho. Comparações entre o SB-index, a junção estrela auxiliada pela R-tree e a junção-estrela auxiliada por GiST indicaram que o SB-index diminui significativamente o tempo de resposta do processamento de consultas *roll-up* e *drill-down* relacionadas aos predicados espaciais "intersecta", "está contido" e "contém", promovendo ganhos de 76% a 96%. Mostrou-se ainda que a variação do volume de dados não prejudica o desempenho do SB-index. Esta dissertação também investiga a seguinte questão: "como a redundância de dados espaciais afeta um DWG?". Foram comparados os esquemas de DWG redundante e não-redundante. Observou-se que a redundância de dados espaciais acarreta prejuízos ao tempo de resposta das consultas e aos requisitos de armazenamento do DWG. Então, visando melhorar o desempenho do processamento de consultas, introduziu-se o SB-index no esquema de DWG redundante. Os ganhos de desempenho obtidos a partir desta ação variaram de 25% a 99%. Por fim, foi proposta uma melhoria sobre o SB-index a fim de lidar especificamente com a questão da redundância de dados espaciais. A partir desta melhoria, o ganho mínimo de desempenho tornou-se 80%.

Palavras-chave: Data warehouse geográfico, Índice Bitmap, Indexação

Abstract

Geographic Data Warehouses (GDW) became one of the main technologies used in decision-making processes and spatial analysis since they provide the integration of Data Warehouses, On-Line Analytical Processing and Geographic Information Systems. As a result, a GDW enables spatial analyses together with agile and flexible multidimensional analytical queries over huge volumes of data. On the other hand, there is a challenge in a GDW concerning the query performance, which consists of retrieving data related to ad-hoc spatial query windows and avoiding the high cost of star-joins. Clearly, mechanisms to provide efficient query processing, as index structures, are essential. In this master's thesis, a novel index for GDW is introduced, namely the SB-index, which is based on the Bitmap Join Index and the Minimum Bounding Rectangle. The SB-index inherits the Bitmap Index' legacy techniques and introduces them in GDW, as well as it enables support for predefined spatial attribute hierarchies. The SB-index validation was performed through experimental performance tests. Comparisons among the SB-index approach, the star-join aided by R-tree and the star-join aided by GiST indicated that the SB-index significantly improves the elapsed time in query processing from 76% up to 96% with regard to queries defined over the spatial predicates of intersection, enclosure and containment and applied to roll-up and drill-down operations. In addition, the impact of the increase in data volume on the performance was analyzed. The increase did not impair the performance of the SB-index, which highly improved the elapsed time in query processing. Moreover, in this master's thesis there is an experimental investigation on "how does the spatial data redundancy affect query response time and storage requirements in a GDW?". Redundant and non-redundant GDW schemas were compared, concluding that redundancy is related to high performance losses. Then, aiming at improving query performance, the SB-index performance was evaluated on the redundant GDW schema. The results pointed out that SB-index significantly improves the elapsed time in query processing from 25% up to 99%. Finally, a specific enhancement of the SB-index was developed in order to deal with spatial data redundancy. With this enhancement, the minimum performance gain observed became 80%.

Keywords: Geographic data warehouse, Bitmap Index, Indexing

Lista de Ilustrações

Lista de Tabelas

Lista de Siglas

a3DR-tree – *aggregate* 3DR-tree

aR-tree – *aggregate* R-tree

aRB-tree – *aggregate* RB-tree

BBC – *Byte-aligned Bitmap Code*

DW - *Data Warehouse*

DWG – *Data warehouse* Geográfico

EQ – *Exact Query*

ERQ – *Enclosure Range Query*

gcc – GNU *Compiler Collection*

IBJ – Índice Bitmap de Junção

IRQ – *Intersection Range Query*

JC – janela de consulta

MAM – Método de Acesso Multidimensional

MBR – *Minimum Bounding Rectangle* (retângulo envolvente mínimo)

OLAP – *On-Line Analytical Processing*

OLTP – *On-Line Transaction Processing*

RID – *Row identifier* (identificador de tupla)

RLE – *Run-lenght encoding* (codificação por comprimento da corrida)

ROLAP – OLAP Relacional

ROT – Relação de Ordem Total

RQ – *Range Query*

SB-index – *Spatial Bitmap Index*

SGBD – Sistema gerenciador de banco de dados

SIG – Sistemas de Informações Geográficas

SOLAP – *Spatial* OLAP (OLAP Espacial)

SSB – *Star Schema Benchmark*

SSD – Sistemas de Suporte à Decisão

TIGER/Line – *Topologically Integrated Geographic Encoding and Referencing system*

WAH – *Word-Aligned Hybrid code*

S U M Á R I O

1. INTRODUÇÃO

Esta dissertação é propõe uma nova estrutura de indexação para *data warehouse* geográfico. A estrutura proposta, denominada SB-index (*Spatial Bitmap Index*), visa diminuir o tempo de resposta de consultas analíticas multidimensionais com um dos seguintes predicados espaciais: **intersecta**, **está contido** e **contém**.

Instituições com as mais variadas finalidades precisam cada vez mais da manipulação de dados que apontem tendências, descrevam o comportamento dos seus negócios e colaborem na elaboração de estratégias. Por isso, a correta organização e manipulação dos dados proporcionam o **suporte à decisão**. O crescente volume dos dados (não integrados) e a necessidade de baixo tempo de resposta a consultas analíticas foram fatores que forçaram a separação dos ambientes operacional e informacional. Assim, foi necessário isolar os dados e as operações do dia-a-dia daqueles dados e consultas analíticas de suporte a decisão. Em suma, dissociou-se OLTP (*On-Line Transaction Processing*) de OLAP (*On-Line Analytical Processing*), originando uma base de dados histórica, integrada, consolidada e intrinsecamente muito volumosa. Esta base de dados recebeu o nome de *data warehouse* (DW) (CHAUDHURI; DAYAL, 1997; KIMBALL; ROSS, 2002; INMON, 2002; RIZZI, 2007). Sobre o DW incidem consultas analíticas multidimensionais, viabilizadas pelas ferramentas OLAP. São estas consultas que favorecem o suporte à decisão, alicerçando a elaboração de estratégias.

Da mesma forma que no passado os Bancos de Dados Espaciais e os Sistemas de Informações Geográficas foram popularizados e empregados também no suporte à decisão, observa-se desde o final da década passada, a inclusão de dados espaciais aos DW (STEFANOVIC; HAN; KOPERSKI, 2000). Os Sistemas de Informações Geográficas (SIG) são ferramentas usadas na manipulação, consulta, análise e visualização de bancos de dados espaciais (CÂMARA et al., 1996; FERRARI, 1997; RIGAUX; SCHOLL; VOISARD, 2002; CASANOVA et al., 2005). Portanto, embora possuam as suas particularidades, DW, OLAP e SIG convergem em um aspecto: são empregados no suporte à decisão.

Este fato motivou a integração das três tecnologias mencionadas, originando o *data warehouse* geográfico (DWG) (STEFANOVIC; HAN; KOPERSKI, 2000; MALINOWSKI; ZIMÁNYI, 2004; 2008; FIDALGO et al., 2004; BIMONTE; TCHOUNIKINE; MIQUEL, 2005). Neste sentido, incentivou-se também o desenvolvimento de ferramentas SOLAP (*Spatial* OLAP), para explorar o DWG por meio de consultas analíticas multidimensionais com predicado espacial (FIDALGO; TIMES; DE SOUZA, 2001; 2004; BIMONTE; TCHOUNIKINE; MIQUEL, 2006; 2007; MALINOWSKI; ZIMÁNYI,

2008). Um exemplo de consulta SOLAP é: “obter o total das receitas ganhas pelos fornecedores localizados em cidades que estão contidas em uma região arbitrária, comercializando produtos de uma determinada marca, agrupando os resultados por ano e por marca”.

Um DW pode ser representado logicamente por um esquema estrela, o qual é composto por uma tabela central de fatos que referencia várias tabelas de dimensão (CHAUDHURI; DAYAL, 1997; KIMBALL; ROSS, 2002; INMON, 2002). No DWG, adicionalmente, existem tabelas de dimensão com atributos espaciais que mantêm geometrias (um conjunto de coordenadas) (STEFANOVIC; HAN; KOPERSKI, 2000; MALINOWSKI; ZIMÁNYI, 2004; 2008; FIDALGO et al., 2004; BIMONTE; TCHOUNIKINE; MIQUEL, 2005; SAMPAIO et al., 2006). O processamento de consultas sobre este esquema exige custosas operações de junção e agrupamento, envolvendo uma enorme quantidade de tuplas. Soma-se ao custo desse processamento, ainda, o cômputo de um ou mais predicados espaciais (GAEDE; GÜNTHER, 1998). Claramente, os desafios para prover um bom desempenho no processamento de consultas SOLAP sobre DWG residem na realização de múltiplas junções envolvendo grande quantidade de dados, e na avaliação de predicados espaciais. Desta forma, mecanismos para reduzir o tempo de resposta das consultas são fundamentais.

Os citados desafios motivaram a realização do presente trabalho, visto que os métodos mais conhecidos atualmente para aumentar o desempenho no processamento de consultas sobre o DW compreendem:

a) a materialização de visões (HARINARAYAN; RAJARAMAN; ULLMAN, 1996; GOLFARELLI; MANIEZZO; RAO, et al., 2003; RIZZI, 2004);
b) a fragmentação horizontal ou vertical dos dados em um *site* ou em diversos *sites* de um ambiente distribuído (GOLFARELLI; MAIO; RIZZI, 2000; CIFERRI, C. et al. 2002; 2007; COSTA; MADEIRA, 2004);
c) o particionamento dos dados entre diversos processadores visando o processamento paralelo (DATTA; MOON; THOMAS, 1998; FURTADO, 2004); e
d) o uso de estruturas de indexação (O’NEIL, P.; GRAEFE, 1995; SARAWAGI, 1997; JOHNSON; SHASHA, 1997; JÜRGENS; LENZ, 1999; PAPADIAS et al., 2001; SIQUEIRA et al., 2009a).

Logo, índices que aumentem o desempenho no processamento de consultas são de vital importância ao DWG, sendo a dimensionalidade o grande obstáculo à indexação de dados do DW. O Índice Bitmap é uma estrutura de indexação para DW cujo desempenho não degenera em grande escala mesmo quando há várias tabelas de dimensão no esquema estrela

(O'NEIL, P.; GRAEFE, 1995; SARAWAGI, 1997; O'NEIL, P.; QUASS, 1997; GOYAL; ZAVERI; SHARMA, 2006; O'NEIL, E.; O'NEIL, P.; WU, K., 2007). Sua vantagem reside em manipular vetores que usam a menor unidade de informação, o bit, por meio de operações lógicas que são auxiliadas pelo *hardware*. Como cada elemento do domínio de um atributo indexado exige a criação de um vetor, prevê-se que atributos com um vasto domínio possam degenerar o desempenho do Bitmap. Isto realmente ocorre. Em outras palavras, atributos com alta cardinalidade deterioram a eficiência do Bitmap. Por outro lado, as técnicas de *binning*, compressão e codificação são propostas justamente para atenuar os efeitos negativos da alta cardinalidade (WU, M.-C.; BUCHMANN 1998; CHAN; IOANNIDIS, 1999; STOCKINGER; WU, 2007; WU, K.; STOCKINGER; SHOSHANI, 2008). Embora o Índice Bitmap constitua um método de acesso promissor, não foi encontrado, na literatura consultada, nenhum registro de seu uso em DWG.

Além disso, o único índice para DWG identificado na literatura consultada foi a aR-tree (PAPADIAS et al., 2001). Este índice usa o método de particionamento do espaço da R-tree (GUTTMAN, 1984) para criar **hierarquias *ad-hoc* implícitas entre os objetos espaciais**. Estas hierarquias favorecem a execução de consultas SOLAP que promovem a agregação de dados. Estas hierarquias colaboram para que o percurso sob a árvore seja reduzido durante uma consulta, pois parte da resposta pode ser encontrada em nodos próximos da raiz. Em contraste com a aR-tree, inúmeras aplicações de DWG utilizam hierarquias predefinidas, tais como região $\prec$ nação $\prec$ cidade $\prec$ endereço. Questiona-se, portanto, a ausência de um índice eficiente para DWG que provenha suporte a **hierarquias de atributos espaciais predefinidas** e que seja capaz de lidar adequadamente com a questão da **multidimensionalidade**.

Esta dissertação propõe o SB-index, um novo índice para DWG cujos principais diferenciais são:

a) Introdução do Índice Bitmap no contexto do DWG;
b) Tratamento de hierarquias de atributos espaciais predefinidas; e
c) Realização de consultas analíticas multidimensionais com predicado espacial.

O SB-index foi validado por meio de testes de desempenho experimentais usando DWG criados a partir de dados descritivos sintéticos e dados geográficos reais. Em particular, os dados sintéticos foram gerados a partir do *Star Schema Benchmark*. Um dos testes avaliou os custos de construção do índice, coletando o tempo decorrido para construí-lo, o número de acessos a disco e também a quantidade de espaço de armazenamento. O outro teste avaliou o processamento de consultas no tocante ao número de acessos a disco

necessários e ao tempo decorrido. Foram avaliadas consultas SOLAP com agregação de dados a partir de diferentes predicados espaciais. Os resultados obtidos foram comparados com o processamento de consultas usando os recursos disponibilizados atualmente pelos gerenciadores de banco de dados. O SB-index proporcionou altos ganhos de desempenho, superando 90% em todos os casos.

Além de apresentar uma nova estrutura de indexação, esta dissertação discorre ainda sobre o impacto causado pela redundância de dados espaciais sobre o DWG. Os mesmos testes de desempenho mencionados foram realizados sobre um DWG redundante. Foram então confrontados os resultados obtidos nos esquemas redundante e não-redundante de DWG. Esta comparação revelou que a redundância de dados espaciais afeta negativamente o armazenamento e o processamento de consultas, ao contrário do que ocorre no DW convencional. Tal comparação ainda como motivou a aplicação do SB-index sobre o esquema redundante de DWG, a qual demonstrou resultados foram excelentes, pois os ganhos de desempenho variaram de 25% a 95% no processamento de consultas. Analisando estes resultados foi possível identificar e desenvolver uma melhoria sobre o SB-index, especificamente para lidar com a redundância de dados espaciais. De posse desta melhoria, os experimentos foram repetidos, obtendo-se o ganho de desempenho mínimo de 80%.

1.1 Organização da Dissertação

O restante desta dissertação está organizado em cinco capítulos:

- **Capítulo 2:** resume os conceitos e fundamentos teóricos sobre *data warehouse* geográfico, os quais são indispensáveis para a compreensão do índice proposto;
- **Capítulo 3:** descreve as principais características do Índice Bitmap e da aR-tree, caracterizando os trabalhos correlatos;
- **Capítulo 4:** propõe o SB-index e são especifica todos os testes realizados sobre o mesmo, discutindo os resultados obtidos;
- **Capítulo 5:** aborda os efeitos da redundância de dados espaciais sobre o DWG, bem como são detalhados os testes realizados e os resultados obtidos; e
- **Capítulo 6:** conclui o trabalho, apresentando as contribuições esperadas e as sugestões para trabalhos futuros.

A proposta do SB-index, os resultados alcançados a partir de sua aplicação e ainda a investigação sobre o impacto da redundância de dados espaciais no DWG foram descritos também em artigos publicados pelo autor desta dissertação: Siqueira et al. 2008, Siqueira et al. 2009a e Siqueira et al. 2009b.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são resumidos os fundamentos teóricos necessários para a compreensão desta dissertação. A Seção 2.1 discorre sobre *data warehouse* e a tecnologia de consultas analíticas multidimensionais para o apoio à decisão (OLAP). A Seção 2.2 descreve os dados e as consultas espaciais, enquanto a Seção 2.3 descreve os principais conceitos sobre *data warehouse* geográfico e SOLAP (OLAP espacial). Na Seção 2.4 existe uma discussão acerca de estruturas de indexação, direcionada aos métodos de acesso multidimensionais com ênfase sobre a R-tree e suas variantes. Por fim, na Seção 2.5 são feitas as considerações finais.

2.1 Data warehouse e OLAP

A descoberta de tendências que auxiliem a melhor compreensão do andamento dos negócios é de vital importância para a elaboração de estratégias em uma instituição. Neste sentido, os **sistemas de suporte à decisão** provêm uma adequada organização e correta manipulação dos dados que podem embasar as estratégias. Contudo, o crescente volume dos dados em geral não integrados, e a necessidade de baixo tempo de resposta para consultas analíticas, foram fatores que determinaram a separação das operações do dia-a-dia de uma instituição de suas operações de decisão estratégica.

On-Line Transaction Processing (OLTP) trata do ambiente operacional. Nele, as operações sobre os dados são predominantemente simples e repetitivas, atômicas e isoladas. O grande volume de operações acessa poucos registros de dados por vez, executando modificações, remoções e inserções, além de consultas. Tais operações são executadas de forma concorrente enquanto o ambiente operacional está *on-line*. Preza-se por consistência e tolerância a falhas, e procura-se maximizar o desempenho de consultas e transações, as quais incidem sobre dados atuais (CHAUDHURI; DAYAL, 1997; KIMBALL; ROSS, 2002; ELMASRI; NAVATHE, 2005; RIZZI, 2007).

Por outro lado, as operações de tomada de decisão são efetuadas por aplicações OLAP (*On-Line Analytical Processing*). Estas operações consistem apenas em leituras sobre dados históricos, resumidos e consolidados, dispensando consultas sobre registros individuais detalhados. Tais consultas são mais complexas, quando comparadas com consultas OLTP, e podem acessar milhões de registros por vez, realizando muitas varreduras, junções e agregações. Sobretudo, objetiva-se maximizar o desempenho das consultas, ao invés do desempenho das transações (CHAUDHURI; DAYAL, 1997; KIMBALL; ROSS, 2002; ELMASRI; NAVATHE, 2005; RIZZI, 2007).

O *data warehouse* (DW) (CHAUDHURI; DAYAL, 1997; WU, M.; BUCHMANN, 1997; INMON, 2002; KIMBALL; ROSS, 2002; RAMAKRISHNAN; GEHRKE, 2002; GOLFARELLI; RIZZI.; IURIS, 2004; RIZZI, 2007) surgiu como uma solução para armazenar os dados estratégicos do ambiente informacional e prover suporte a aplicações OLAP. A Figura 1 exibe o DW como o protagonista em uma arquitetura de *data warehousing*. Os provedores de informação compreendem desde SGBD relacionais, orientados a objetos e objeto-relacionais, até bases de conhecimento, sistemas legados (tais como sistemas hierárquicos e de rede), além de planilhas e arquivos de dados, entre outros. Os dados oriundos dos provedores recebem um tratamento visando a sua integração (KIMBALL; CASERTA, 2004; ADZIC; FIORE; SISTO, 2007): a extração a partir dos provedores, a tradução para o formato adequado e a limpeza de elementos irrelevantes.

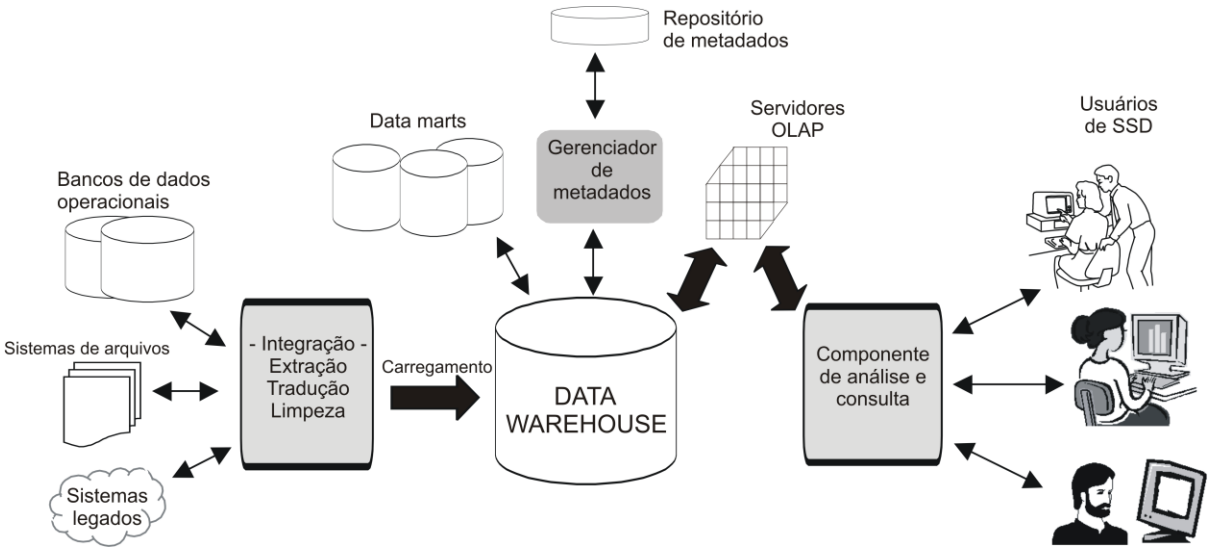


Figura 1 – Ambiente típico de *data warehousing*.

Os metadados conservam os mapeamentos entre esquemas de diferentes provedores e entre dados semanticamente diferentes, além de controlar a versão dos dados históricos (WU, M.; BUCHMANN, 1997). Um *data mart* pode constituir um protótipo para um DW, ou uma solução na qual o escopo de aplicação é limitado se comparado ao escopo do DW. Seus dados compartilham as mesmas características dos dados do DW. O servidor OLAP (HYDE, 2008; MONDRIAN, 2008; OLAP COUNCIL, 2008; SAS OLAP, 2008) atua entre o DW mantido por um SGBD relacional e os componentes acessíveis aos usuários. Ele é composto por um *middleware* com suporte a consultas multidimensionais, as quais permitem aos usuários analisar o negócio sob variadas perspectivas.

Além disso, o servidor OLAP manipula uma linguagem de consulta multidimensional, a MDX (***M****ulti****d****imensional E****x****pressions*) (WHITEHORN; ZARE;

PASUMANSKY, 2005), e efetua o correspondente mapeamento para a SQL. Realizando este mapeamento e submetendo a consulta ao SGBD, o servidor OLAP explora a escalabilidade e as funcionalidades do SGBD. Vale ainda ressaltar que o servidor OLAP também é responsável por identificar e materializar as visões relevantes. O componente de análise e consulta viabiliza aos usuários as consultas analíticas, sem que estes tenham acesso aos detalhes da arquitetura.

Por fim, é importante enfatizar que o DW é também caracterizado por (CHAUDHURI; DAYAL, 1997; INMON, 2002; KIMBALL; ROSS, 2002; KIMBALL; CASERTA, 2004; GOLFARELLI; RIZZI; IURIS, 2004; ADZIC; FIORE; SISTO, 2007; RIZZI, 2007):

a) armazenar os dados somente após eliminar as diferenças semânticas e de modelo dos dados dos provedores de informação, por meio de tradução, filtragem e integração;
b) manter dados orientados ao assunto, históricos e não-voláteis;
c) viabilizar consultas analíticas diretamente sobre si, dispensando o acesso aos provedores de informação originais;
d) proporcionar rapidez e eficiência na busca; e
e) receber atualizações incrementais periódicas, a fim de refletir as atualizações ocorridas nos provedores de informação.

A modelagem de dados multidimensional permite que as aplicações OLAP manipulem os dados do DW como se estivessem organizados em um hipercubo (CHAUDHURI; DAYAL, 1997; CIFERRI, C., 2002; RIZZI, 2007). As células do hipercubo armazenam eventos que ocorreram no domínio de negócio. Assim, o fato analisado é quantificado por medidas e caracterizado por dimensões. As dimensões do hipercubo definem o contexto sob o qual os fatos são avaliados. O valor de uma medida é um ponto no espaço multidimensional, definido e influenciado pelos valores das dimensões.

Um exemplo no domínio de uma aplicação de varejo aplica estes conceitos: um fato típico são as vendas, descritas pela medida quantidade de itens vendidos e caracterizadas pelas dimensões item, fornecedor e data. O hipercubo correspondente é mostrado na Figura 2. O exemplo do hipercubo com três dimensões facilita a visualização, embora seu projeto possa incluir tantas dimensões quanto o domínio do problema demandar. Medidas e hierarquias são aspectos estáticos do modelo multidimensional. Os atributos das dimensões, embora omitidos no hipercubo da Figura 2, podem estar relacionados por meio de uma hierarquia de relacionamento.

No exemplo em questão, uma hierarquia determina que cidade é um nível de granularidade abaixo de nação. Estas hierarquias é que viabilizam a agregação de dados em níveis mais detalhados ou mais resumidos. Portanto, uma agregação detalhada por cidade origina uma agregação mais resumida por nação, considerando todas as cidades de cada nação. O mesmo se observa com os atributos item e categoria, e os atributos mês, trimestre e ano. Tais características do modelo multidimensional enfatizam a agregação de valores das medidas, segundo uma ou mais dimensões (CHAUDHURI; DAYAL, 1997; POURABBAS; RAFANELLI, 1999; ARIGON; TCHOUNIKINE; MIQUEL, 2006). Por exemplo, além dos valores das medidas para cada célula, o hipercubo da Figura 2 mantém as funções de agregação originadas ao longo das dimensões: soma das (ou total de) vendas por localidade, ou total de vendas por ano, por exemplo.

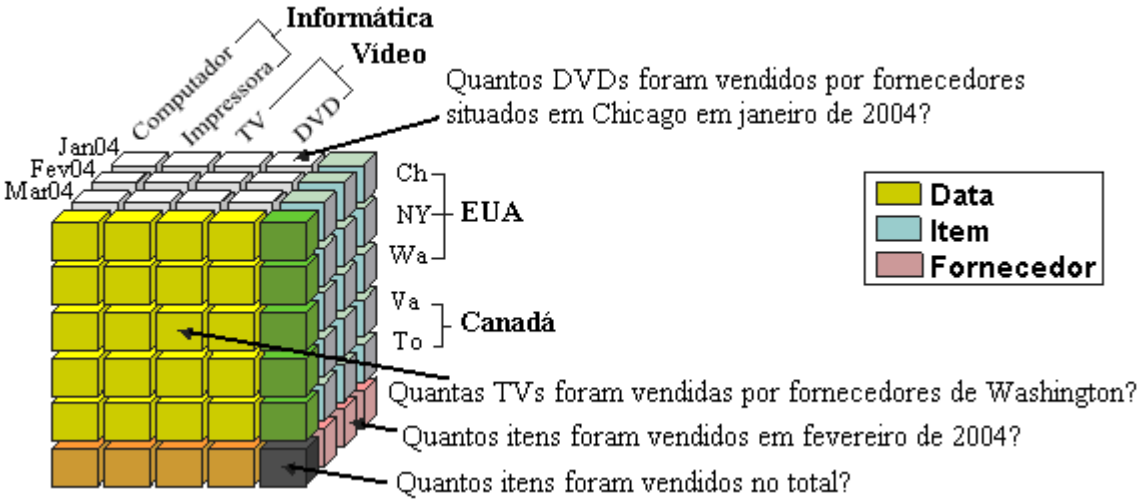


Figura 2 – Cubo de dados multidimensional referente ao exemplo de varejo, e algumas consultas analíticas.

A agregação de dados é ainda mais realçada com as operações dinâmicas sobre o hipercubo. Na operação *drill-down*, os dados são agregados progressivamente no sentido mais detalhado. Por outro lado, na operação *roll-up* a agregação ocorre no sentido mais resumido. A Figura 3 exemplifica as operações *roll-up* e *drill-down*. Outras operações dinâmicas muito conhecidas são *pivoting*, *slice-and-dice* e *drill-across* (WU, M.-C.; BUCHMANN, 1997; POURABBAS; RAFANELLI, 1999; KIMBALL; ROSS, 2002; CIFERRI, C. 2002).

+ resumido ***Drill-down*** + detalhado	Encontre o total de vendas de TV dos fornecedores situados nas Américas Encontre o total de vendas de TV dos fornecedores situados nos EUA. Encontre o total de vendas de TV dos fornecedores situados em Chicago.	+ resumido ***Roll-up*** + detalhado

Figura 3 – Exemplos das operações *drill-down* e *roll-up*.

Um hipercubo d-dimensional pode ser representado por uma treliça (ou reticulado), a qual consiste em um grafo dirigido que possui todas as combinações possíveis de agregações sobre as dimensões. Se o hipercubo possui d dimensões, então existem 2^d combinações de subconjuntos destas dimensões. Os níveis com mais dimensões possuem dados mais detalhados (menor granularidade), enquanto que os níveis com menos dimensões possuem dados mais agregados (resumidos).

A Figura 4 exibe a treliça para o exemplo de varejo exposto. A sua estruturação mostra que muitos vértices compartilham dimensões. Por exemplo, ambos os vértices DFI quanto DF retratam as dimensões D e F. Um mecanismo computacional para produzir *group-bys* pode explorar estes relacionamentos, dividindo a treliça em níveis de agregação. Não é necessário computar todos os níveis, uma vez que o ancestral e um ou mais dos descendentes pode ser capaz de compartilhar alguma porção da agregação de dados. Assim, a resposta para uma consulta pode ser encontrada nos níveis resumidos.

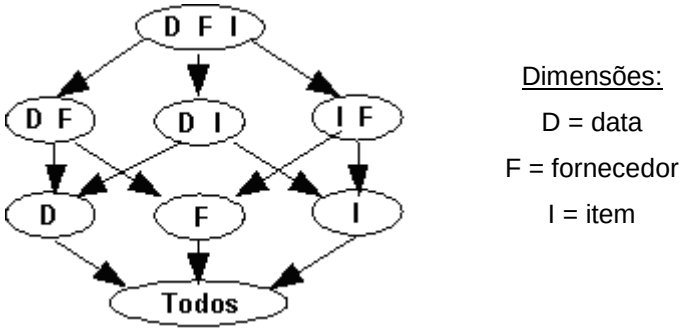


Figura 4 – Treliça para o exemplo de varejo.

Segundo Harinarayan et al. (1996), uma treliça é composta por um conjunto L de visões e uma relação de dependência $\preceq$, isto é: $\langle L, \preceq \rangle$. Sejam Q_1 e Q_2 duas consultas. Então, $Q_1 \preceq Q_2$ se e somente se Q_1 pode ser respondida usando apenas os resultados de Q_2. Na Figura 4, por exemplo, IF $\preceq$ DFI, pois toda consulta envolvendo os atributos item (I) e fornecedor (F) pode ser respondida por consultas envolvendo data (D), fornecedor (F) e item (I). Algumas consultas não satisfazem à relação de dependência. Logo, são falsas as declarações: I $\preceq$ DF e F $\preceq$ D.

Para dois elementos *a* e *b* de uma treliça, $a \prec b$ significa que $a \preceq b$ e $a \neq b$. Os ancestrais e os descendentes de um elemento são definidos como: *ancestral*(*a*) = $\{b \mid a \preceq b\}$ e *descendente*(*a*) = $\{ b \mid b \preceq a \}$, respectivamente. Logo, cada elemento é ancestral e descendente de si mesmo. Os ancestrais imediatos (ou diretos) de um elemento *a* são obtidos pela função *next*(*a*) = $\{ b \mid a \prec b$, não existe *c* tal que $a \prec c$, $c \prec b \}$. Já os descendentes

imediatos (ou diretos) de um elemento a são obtidos pela função $previous(a) = \{ b \mid b \prec a$, não existe c tal que $b \prec c, c \prec a\}$.

A representação lógica de um modelo multidimensional pode ser baseada no modelo relacional. Nesta abordagem, denominada ROLAP (*Relational OLAP*), as relações podem estar definidas sob o esquema estrela ou sob o esquema floco-de-neve. No primeiro, existem uma tabela para cada dimensão e uma única tabela de fatos. Toda tupla na tabela de fatos mantém uma chave estrangeira para cada tabela de dimensão. Os atributos de uma tabela de dimensão podem se relacionar por meio de uma hierarquia. A Figura 5 exibe o esquema estrela para o exemplo de varejo e indica a quantidade de tuplas por tabela.

Cada tupla na tabela de fatos, portanto, corresponde a uma célula do hipercubo. Além disso, cada vértice da treliça pode ser expresso por um esquema-estrela distinto que contemple as dimensões envolvidas. Por exemplo, a tabela de fatos do vértice DFI manterá chaves estrangeiras para as tabelas de dimensão Data, Fornecedor e Item. Já a tabela de fatos do vértice D manterá apenas a chave estrangeira para a tabela Data.

O atributo *Mês* da tabela de dimensão Data reforça a atenção com a dimensão temporal. Os tipos padrão *timestamp* e *date*, de SQL, não são adequados para a tomada de decisão estratégica. Por isso, para resumir adequadamente os dados provenientes de operações de negócio, informações como trimestre fiscal, se um dia é ou não feriado, entre outras, podem ser mantidas para cada data (RAMAKRISHNAN; GEHRKE, 2003).

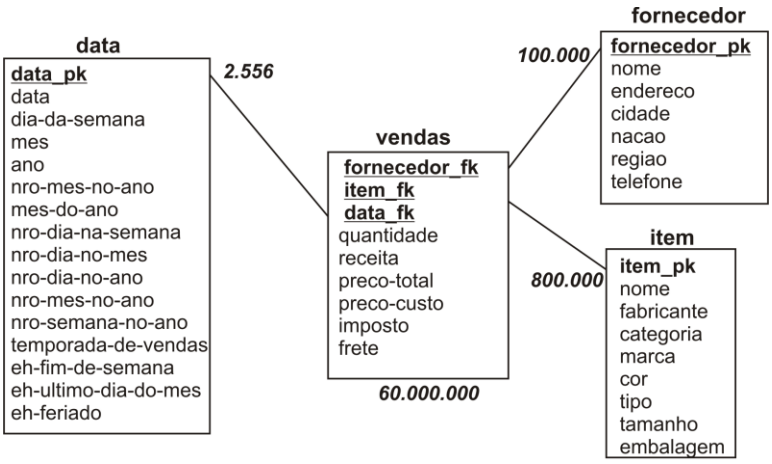


Figura 5 – Esquema estrela para o exemplo de varejo.

Todas as consultas exibidas nas Figuras 2 e 3 podem ser escritas em SQL, usando as tabelas de dimensão e a tabela de fatos. Porém, uma consulta como "encontre os 5 produtos mais vendidos" exige um esforço extra. Deve-se obter o conjunto dos produtos vendidos e classificá-lo segundo o número de vendas, extraindo os 5 mais vendidos. Portanto, embora diversas consultas possam ser respondidas com SQL, algumas exigem muito esforço.

Para representar explicitamente a hierarquia de atributos, emprega-se o esquema floco-de-neve promovendo a normalização das tabelas de dimensão. Embora a estrutura normalizada torne explícita a hierarquia, ela introduz novas junções entre as tabelas. Este é um obstáculo ao bom desempenho no processamento de consultas, tornando o esquema estrela preferível ao esquema floco-de-neve (KIMBALL; ROSS, 2002).

Por fim, é importante enfatizar que diversas são as formas de obter um bom desempenho no processamento de consultas sobre o DW, frente à existência de operações custosas de junção entre tabelas e *group-bys*. São elas:

a) a materialização de visões (HARINARAYAN; RAJARAMAN; ULLMAN, 1996; GOLFARELLI; MANIEZZO; RIZZI, 2004; RAO, et al., 2003);
b) a fragmentação horizontal ou vertical dos dados em um *site* ou em diversos *sites* de um ambiente distribuído (GOLFARELLI; MAIO; RIZZI, 2000; CIFERRI, C. et al. 2002; 2007; COSTA; MADEIRA, 2004);
c) o particionamento dos dados entre diversos processadores visando o processamento paralelo (DATTA; MOON; THOMAS, 1998; FURTADO, 2004); e
d) o uso de estruturas de indexação (O'NEIL, P.; GRAEFE, 1995; SARAWAGI, 1997; JOHNSON; SHASHA, 1997; JÜRGENS; LENZ, 1999; PAPADIAS et al., 2001; SIQUEIRA et al., 2009a).

2.2 Dados e Consultas Geográficos

Os fenômenos do mundo real que possuem uma localização no globo terrestre podem ser representados por abstrações em forma de feições geográficas em SGBD espaciais. Esta representação corresponde ao georreferenciamento. Os dados georreferenciados possuem elementos não espaciais para descrever nominalmente o fenômeno geográfico ("vulcão", "bairro", "bacia petrolífera" etc.), e componentes espaciais em diferentes níveis de abstração.

Estes componentes informam a localização associada às propriedades geométricas e topológicas do fenômeno. Por exemplo, uma cidade pode ser representada tanto pelo seu perímetro (nível mais detalhado), quanto pelo seu ponto central (nível com alta abstração). Ainda, poderá existir um componente temporal que retrate a data da coleta do dado. Para um estudo detalhado sobre bancos de dados geográficos e sistemas de informações geográficas, recomenda-se a leitura de Câmara et al. (1996), Ferrari (1997), Rigaux, Scholl e Voisard (2002), e Casanova et al. (2005).

Um ponto é a menor unidade possível para representar um objeto espacial, e caracteriza a inexistência de extensão espacial. Em geral, pontos representam localizações

discretas, como uma ocorrência policial num mapa. Uma linha é uma seqüência de pontos conectados de forma retilínea, enquanto que uma linha poligonal não tem os pontos dispostos desta forma. Em ambas, cada par de pontos conectados corresponde a um segmento de linha. Linhas e linhas poligonais são usadas para representar objetos espaciais lineares, como rios, estradas, ferrovias, e redes de infra-estrutura.

Um polígono é formado por uma seqüência fechada de linhas ou de linhas poligonais, ou seja, seu último ponto coincide com o primeiro. Polígonos e linhas poligonais podem ser empregados de forma semelhante, mas em SGBD espaciais podem também diferir em questões de armazenamento de atributos, como área e perímetro, por exemplo. Os polígonos complexos, por sua vez, podem permitir buracos ou consistir de diversas partes disjuntas. Polígonos ou linhas poligonais fechadas são usados para representar objetos bidimensionais, como a área ocupada por um bairro.

Para obter a documentação completa da especificação dos padrões de dados geográficos, pode-se consultar o sítio do *Open Geospatial Consortium* (OPENGIS, 2007). As formas mencionadas são ilustradas na Figura 6.

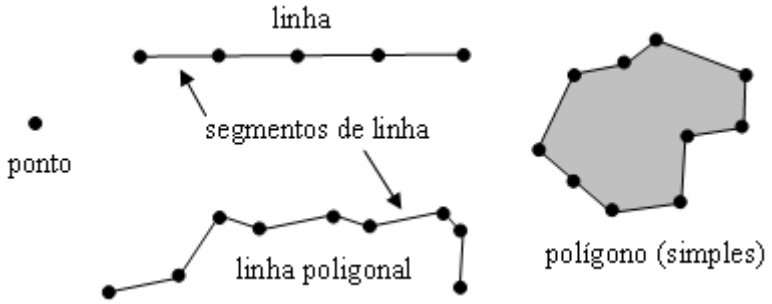


Figura 6 – Tipos de objetos espaciais.

Todo objeto espacial tem pelo menos um atributo que descreve suas extensão e localização no espaço Euclidiano d-dimensional (E^d), ou em algum sub-espaço deste. Tal atributo é denominado atributo espacial, expresso por **o.G**, e consiste num conjunto de coordenadas. Cada uma delas é representada no espaço bidimensional por (x,y), e se refere a algum tipo de dado espacial. As notações $\mathbf{o.G^o}$ e $\mathbf{o\partial G}$ denotam, respectivamente, o interior e a fronteira de **o.G**.

O espaço multidimensional é adequado para a disposição de coordenadas georreferenciadas, pois representa intuitivamente a dimensão espacial e assim viabiliza o uso das propriedades geométricas tais como ângulo, área, perímetro, etc. O armazenamento de um objeto espacial de dimensão não-zero exige a alocação de grande porção de espaço em

memória secundária, para acomodar os diversos pares de coordenadas que descrevem a geometria exata do objeto.

É possível realizar uma variedade de consultas sobre objetos espaciais dispostos no espaço multidimensional. São tratadas agora as consultas espaciais necessárias para a compreensão desta dissertação. A **range query** (RQ), também conhecida como **window query** ou **rectangle query**, busca por todos os objetos que satisfaçam um certo relacionamento topológico com o retângulo d-dimensional **R**. Ou seja, θ(**o.G**, **R**) = {intersecta, está contido, contém}. **R** tem lados paralelos aos eixos de suas respectivas dimensões, consiste na janela de consulta (JC), e ainda é representado por um conjunto de intervalos d-dimensionais fechados $I=\{ [l_1, u_1], ..., [l_k, u_k] \}$. Neste conjunto, $[l_i, u_i]$ descreve a extensão ao longo da dimensão i, para $1 \leq i \leq k$. São analisadas a seguir as três subclasses de RQ, e exemplificadas na Tabela 1. Todas partem da premissa $\mathbf{R} \subseteq \mathbf{E}^d$.

a) A **intersection range query** (IRQ) trata do relacionamento topológico intersecta em RQ. São encontrados todos os objetos que tenham pelo menos um ponto em comum com **R**. Ou seja: $IRQ\ (R, dataset) = \{ o \mid o \in dataset \wedge o.G \cap R \neq \emptyset\}$;
b) Quando dentre os dados só existem pontos, a IRQ se assemelha a **containment range query** (CRQ). Esta, por sua vez, trata o relacionamento topológico está contido em uma RQ. Desta forma, a CRQ encontra todos os objetos **o** contidos em **R**. Isto é:
   $CRQ\ (R, dataset) = \{o \mid o \in dataset \wedge o.G \cap R = o.G\} \equiv \{o \mid o \in dataset \wedge o.G \subseteq R\}$;
c) A **enclosure range query** (ERQ), por fim, é uma RQ com relacionamento topológico contém. Nela, são encontrados todos os objetos que englobam **R**:

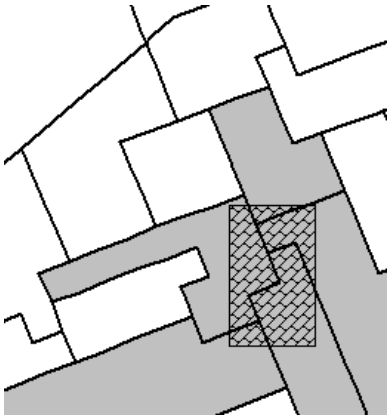
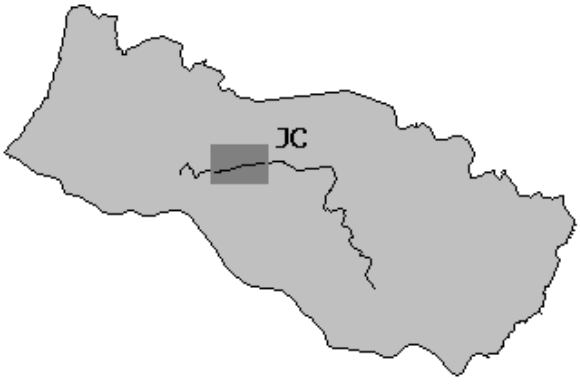
$ERQ\ (R, dataset) = \{ o \mid o \in dataset \wedge o.G \cap R = R \} \equiv \{ o \mid o \in dataset \wedge o.G \supseteq R \}$.

2.3 Data warehouse geográfico e SOLAP

Os Sistemas de Informações Geográficas (SIG) são ferramentas poderosas usadas na manipulação, consulta, análise e visualização de bancos de dados espaciais (CÂMARA et al., 1996; FERRARI, 1997; RIGAUX; SCHOLL; VOISARD, 2002; CASANOVA et al. 2005). A união de SIG aos conceitos de DW/OLAP inspira um modelo de dados específico para suporte à decisão, com flexibilidade e agilidade de análises multidimensionais sobre um grande volume de dados, e com o poder da visualização de mapas e da execução de consultas espaciais (FIDALGO, 2005; BÉDARD; RIVEST; PROULX, 2007). Todavia, a integração de DW para análise sob os contextos multidimensional e espacial não é tema consolidado. Prova disso são as inúmeras propostas

de modelos para DWG: Stefanovic et al. (2000), Malinowski e Zimányi (2004), Bimonte et al. (2005), Fidalgo (2005) e Sampaio et al. (2006).

Tabela 1 – Consultas espaciais comentadas e exemplificadas.

Consulta / Exemplo	Figura
Intersection range query Fornecida uma região retangular com a finalidade de construir um parque, identifique todos os bairros que terão terras desapropriadas. → A região retangular entrelaçada (janela de consulta) é onde se pretende construir o parque. Os polígonos são os bairros, sendo que os polígonos na cor cinza intersectam a janela de consulta e fazem parte da resposta. Os polígonos na cor branca não intersectam a janela de consulta, portanto não são parte da resposta.	
Containment range query Fornecida uma região retangular sobre uma cidade, recupere as instituições de ensino nela contidas, bem como os bairros. → A região retangular cinza tem coloridas em preto as instituições nela contidas. O único bairro contido por ela está destacado com listras. As instituições que não fazem parte da resposta têm cor cinza, e os bairros não recebem cor de destaque (exceto onde a janela de consulta os sobrepõe).	
Enclosure range query Fornecida uma região retangular que intersecta um rio, encontre a bacia hidrográfica a que ele pertence. → A janela de consulta (JC) retangular (cinza escuro) intersecta um rio (linha), e por isso toda a bacia hidrográfica é colorida em um tom de cinza mais claro.	

Apesar das peculiaridades de cada um, OLAP e SIG convergem quando o foco é suporte à decisão. Esta interseção é especialmente relevante porque existem muitos dados no ambiente de DW/OLAP que possuem natureza geográfica, mas não são devidamente

tratados dentro de um contexto espacial. Já os SIG, intrinsecamente, não provêm suporte para análise multidimensional de dados, o que acrescentaria mais qualidade às decisões a serem tomadas no âmbito de aplicações geográficas. Embora a integração de SIG, OLAP e DW pareça natural, na prática isto não ocorre porque cada um deles foi criado para tarefas diferentes, e porque possuem dados, metadados e serviços que não são diretamente integrados (FIDALGO, 2005; BÉDARD; RIVEST; PROULX, 2007).

A integração SIG, OLAP e DW pode promover consultas analíticas sobre o comportamento do negócio, segundo o ponto de vista geográfico. Desta forma, é possível responder com qualidade a questões do tipo "o que?", "quem?" e "onde?". São exemplos de consultas analíticas:

a) "qual o total da receita obtida por ano, por nome do fornecedor, e por categoria do produto para aqueles fornecedores situados dentro de uma área definida *ad-hoc*?";

b) "qual o melhor local para abrir uma nova filial de fornecimento, levando em consideração os endereços dos meus 500 melhores clientes nos semestres e trimestres de 2003?";

c) "qual a receita total obtida em províncias cruzadas por pelo menos um rio?";

d) "quem são os meus cem melhores clientes num raio de 5 km da matriz?";

e) "onde moram os clientes que gastaram mais de R$5.000,00 por fornecedor por ano?".

Um DW que armazena dados espaciais em uma ou mais dimensões ou em pelo menos uma medida é denominado *Data Warehouse* Geográfico (DWG). Ele possui forte ênfase em dados geográficos, os quais se apresentam sob a forma de dimensões ou fatos (STEFANOVIC et al., 2001; MALINOWSKI; ZIMÁNYI, 2004; FIDALGO et al., 2004; RIZZI et al., 2006; BÉDARD; RIVEST; PROULX, 2007).

Esta dissertação usa os conceitos descritos por Stefanovic et al. (1997; 2000), Fidalgo et al. (2004), Malinowski e Zimányi (2005). O esquema estrela é considerado uma boa opção para modelar um DWG, porque provê uma estrutura de armazenamento concisa e organizada, além de facilitar as operações SOLAP (STEFANOVIC et al., 1997; 2000). Para alcançar um bom desempenho em SOLAP, tais autores propõem extensões ao esquema estrela tradicional: as tabelas de dimensão e as medidas podem ser espaciais ou não-espaciais.

Uma **tabela de dimensão espacial não-geométrica** mantém dados geográficos descritivos sobre a localização de um objeto espacial, mas nenhum dado geográfico possui geometria associada. Por exemplo, descrições nominais de endereços, nomes de cidades, nomes de nações, etc. Por sua vez, a **tabela de dimensão estritamente espacial** define todos os níveis da hierarquia de atributos espaciais como feições geométricas. Por exemplo, pontos expressando endereço, polígonos expressando cidades, nações e regiões.

As **medidas numéricas** contêm apenas dados numéricos, conforme já discutido (tais como população e área). Já as **medidas espaciais** são constituídas de uma coleção de ponteiros para objetos geográficos (Stefanovic et al., 1997; 2000). Esta dissertação não enfoca medidas espaciais, mas sim o tratamento de dados espaciais nas tabelas de dimensão de um esquema estrela. Para melhor conhecer o tratamento de medidas espaciais, recomenda-se a leitura de Stefanovic, Han e Koperski (2000), Malinowski e Zimányi (2004), Bimonte et al. (2005), Silva et al. (2008).

Nesta dissertação, foram consideradas apenas **hierarquias espaciais simples simétricas** (MALINOWSKI; ZIMÁNYI, 2005), com o relacionamento espacial **está contido**. Assim, os objetos espaciais dos níveis de granularidade mais altos contêm aqueles objetos dos níveis mais baixos. Nesta classe de hierarquia, e com este relacionamento espacial, um objeto pertencente a um nível de granularidade inferior não pode estar contido em mais que um objeto de um nível superior. Todo objeto que não é do nível mais baixo de granularidade contém pelo menos um objeto de um nível inferior. Um exemplo de hierarquia desta classe determina Nação como sendo o nível superior, Cidade como sendo um nível intermediário e Endereço como sendo o nível inferior. É muito importante salientar que as hierarquias espaciais estão intimamente ligadas às operações *roll-up* e *drill-down* espaciais, ou seja, operações SOLAP de agregação de dados. Para conhecer outras classes de hierarquias, recomenda-se a leitura de Malinowski e Zimányi (2005).

2.3.1 Modelos existentes para data warehouse geográfico

Esta seção aborda os principais modelos conceituais e lógicos propostos para DWG e presentes na literatura. O primeiro *framework* para DWG foi proposto por Stefanovic et al. (1997; 2000), provendo suporte a tabelas de dimensão e medidas espaciais. Foram ainda desenvolvidos algoritmos eficientes para construir os cubos de dados geográficos. A solução fundamentou-se na materialização seletiva do conjunto de objetos espaciais, levando em consideração a freqüência de acesso desse conjunto, o custo de armazenamento, e o benefício que o mesmo trará para a construção de outros cubos geográficos derivados desse conjunto. Sampaio et al. (2006) reusaram este framework para a proposição de um modelo lógico de DWG, posteriormente validado no SGBD Oracle.

Malinowski e Zimányi (2004) estenderam o modelo de Stefanovic et al. (1997; 2000), e criaram um modelo multidimensional conceitual baseado no modelo Entidade-Relacionamento. Nele, as dimensões espaciais podem existir mesmo na ausência de alguns níveis da hierarquia de relacionamento de atributos espaciais. Isto é, *cidade* pode ser uma

dimensão espacial sem qualquer outra subdivisão geográfica. No mapeamento para o modelo lógico, podem ser obtidos esquemas similares ao esquema floco-de-neve, mas que mantém atributos espaciais. Os esquemas resultantes podem compartilhar dimensões espaciais, pois no nível conceitual de abstração é possível reusar hierarquias e evitar repetições desnecessárias das representações dos objetos geográficos. Por outro lado, o processamento de consultas tem o custo da computação de várias junções somado ao do cálculo do predicado espacial.

Bimonte, Tchounikine e Miquel (2005) propuseram e formalizaram um modelo multidimensional espacial contemplando a existência de medidas espaciais, de relacionamentos N:N entre tabelas fatos e tabelas de dimensão, e de suporte a funções de agregação *ad-hoc*. Uma medida espacial é modelada como uma entidade complexa e composta por um conjunto de atributos espaciais e alfanuméricos. Logo, a medida atua não como um valor quantitativo atômico e independente, e sim como objeto complexo com atributos. A dependência semântica entre atributos determina que a mudança de uma função de agregação para um atributo implique na mudança da agregação para os demais dependentes. Por exemplo, se união geométrica determina soma do número de endereços de fornecedores, interseção geométrica pode implicar a diferença.

Bimonte, Tchounikine e Miquel (2006) trouxeram melhoramentos ao modelo anteriormente proposto. Um deles foi o tratamento simétrico de dimensões e medidas. Isto tornou viável transformar uma dimensão em uma medida e vice-versa, alterando a semântica da análise. Formalizou-se também uma álgebra que redefine os operadores OLAP mais comuns para navegação pelo hipercubo, tais como *roll-up* e *slice*.

O uso de medidas espaciais é contestado por Fidalgo, Times e de Souza (2004) e Fidalgo (2005). Ambos refutam a sua adoção, pois argumentam que em todo DW cada medida é um valor quantitativo, e não uma coleção de ponteiros. A implementação desta coleção em forma de texto delimitado exigiria um processamento adicional, podendo inclusive aumentar a complexidade de computação para obter os valores da medida. Argumenta-se ainda que resultados semelhantes podem ser obtidos pelo uso de dimensões espaciais, dispensando as medidas. Além disso, tais autores argumentam que a redundância dos dados espaciais pode afetar negativamente a capacidade de armazenamento. Os mencionados autores, contudo, não realizaram experimentos a fim de atestar este fato.

Claramente, há controvérsias sobre a representação do hipercubo espacial no modelo relacional. Uma proposta de uma estrutura de indexação para DWG deve atentar para este fato, e assim selecionar a representação adequada para a qual proverá suporte. Outros trabalhos propostos não definem um verdadeiro DWG, porque a manipulação de dados

convencionais é feita no DW, enquanto os dados espaciais são manipulados no SIG. Este último, muitas vezes, armazena os dados em arquivos de padrão proprietário: SHP no ESRI ArcGIS (SHAPEFILE, 1998) e MIF no MapInfo (MAPINFO, 2005). Outros trabalhos consideram como DWG um volumoso banco de dados geográficos, ou quando é requerida a integração ou agregação de dados geográficos, ou no emprego de dados geográficos para suporte à decisão. Porém, tais usos não são admitidos no contexto desta dissertação.

2.4 Estruturas de indexação

Índices são estruturas de acesso auxiliares usadas para aumentar o desempenho da recuperação de registros na resposta a certas condições de busca. Uma estrutura de indexação oferece caminhos alternativos de acesso a registros sem alterar a organização física dos dados (ELMASRI; NAVATHE, 2005), e possibilita selecionar e recuperar os registros que satisfazem às condições de consulta segundo uma chave de busca (RAMAKRISHNAN; GEHRKE, 2002), a qual consiste num conjunto arbitrário de atributos. Um índice permite encontrar um registro consultando apenas uma pequena fração de todos os registros possíveis (GARCIA-MOLINA; ULLMAN; WIDOW, 2000), pela imposição de uma ordem de acesso sem de fato reordenar os registros (FOLK; ZOELLICK, 1992). Quaisquer atributos de uma relação podem ser usados para criar um índice.

Comumente as estruturas de indexação crescem em volume, exigindo que seu armazenamento seja feito em memória secundária. Freqüentemente isto se realiza em disco. Os custos no armazenamento secundário estão relacionados à busca de uma determinada localização em disco. Uma vez posicionada, a cabeça de leitura pode processar um fluxo contíguo de bytes rapidamente. É uma combinação de busca lenta e transferência de dados rápida (FOLK; ZOELLICK, 1992).

Geralmente, os registros de um índice são menores que aqueles do arquivo de dados, porque possuem menos atributos (normalmente um valor de chave, um ponteiro e nenhum ou poucos outros atributos relevantes para a busca), mas podem também ocupar diversos blocos. Sendo assim, mesmo com estratégias de busca eficientes, podem ser necessários vários acessos a disco para obter o registro desejado. A modificação dos dados por meio de inserções, eliminações e atualizações implica, em certos casos, em reorganização custosa do índice e no gerenciamento do espaço extra concedido ao armazenamento (blocos de *overflow*).

As estruturas de indexação convencionais, como B-tree e Hash (FOLK; ZOELLICK, 1992; GARCIA-MOLINA; ULLMAN; WIDOW, 2000; RAMAKRISHNAN;

GEHRKE, 2002; ELMASRI; NAVATHE, 2005) assumem uma única chave de busca e auxiliam a recuperação de registros de acordo com os valores dela. Índices assim caracterizados possuem apenas uma dimensão, seja a chave de busca um único atributo ou uma concatenação deles. Aplicações OLAP e geográficas exigem a visualização de dados em espaços com duas ou mais dimensões. Por isso, se diferenciam dos SGBD tradicionais pelo suporte a certos tipos de consultas.

O espaço multidimensional é o domínio utilizado pelos métodos de acesso multidimensionais (MAM) (GAEDE; GÜNTHER, 1998) para indexar elementos com dimensionalidade. Como antecipado na Seção 2.2, o armazenamento de todos os pares de coordenadas de um objeto espacial é custoso. Por outro lado, existem **aproximações** (abstrações) dos objetos espaciais para representar suas geometrias. As aproximações asseguram, por sua propriedade conservativa, que nenhum dos objetos espaciais que satisfaz a um certo relacionamento espacial seja desconsiderado na resposta da consulta.

A aproximação mais utilizada é o retângulo envolvente mínimo (MBR – *Minimum Bounding Rectangle*), que consiste no menor retângulo d-dimensional com lados paralelos aos eixos, e que contém completamente o objeto espacial. Em um espaço multidimensional, esta aproximação consiste em quatro pares de coordenadas: $(X_{mín}, Y_{mín})$, $(X_{mín}, Y_{máx})$, $(X_{máx}, Y_{mín})$ e $(X_{máx}, Y_{máx})$, nas quais $X_{mín}$ e $Y_{mín}$ são os menores valores das coordenadas X e Y do retângulo, respectivamente, e $X_{máx}$ e $Y_{máx}$ são os maiores destes valores. A área vazia que não faz parte da geometria, mas que pertence à aproximação, chama-se *dead space*. Se esta área for considerada no teste de um relacionamento espacial, o objeto espacial correspondente é um falso candidato, porque não satisfaz ao citado relacionamento espacial.

Devido a esta imprecisão no conjunto de respostas, o processamento de consultas espaciais nos MAM exige a filtragem e o posterior refinamento das respostas. Na primeira, objetos que não satisfazem a um relacionamento espacial são descartados e, devido ao uso de aproximações, surgem falsos candidatos. Esta fase é pouco custosa, graças à manipulação de geometrias simples. No refinamento, cada objeto candidato passa pela verificação que testa se o mesmo obedece ao relacionamento espacial, descartando assim os falsos. Como trata das geometrias exatas dos objetos espaciais, esta fase é mais custosa, pois, embora o volume de dados seja menor, os cálculos são mais complexos e requerem a recuperação de dados armazenados em disco (BRINKHOFF et al., 1994).

Portanto, a filtragem deve reduzir drasticamente a quantidade de objetos a serem analisados na fase de refinamento, para que se obtenham ganhos expressivos em desempenho. Aproximações progressivas, que consistem em subconjuntos de pontos do

objeto espacial, podem ser usadas para validar candidatos numa fase intermediária entre a filtragem o refinamento, dispensando que alguns dos candidatos verdadeiros sigam ao refinamento. Existem ainda aproximações mais precisas que o MBR, embora mais custosas em armazenamento, tais como o *convex hull* e o 5C (BRINKHOFF; KRIEGEL; SCHNEIDER, 1993).

A Seção 2.4.1 discorre sobre um MAM popularizado dentre os SGBD com suporte a objetos espaciais e de reconhecida eficiência: R-tree.

2.4.1 R-tree

A **R-tree** (GUTTMAN, 1984) é um MAM baseado na B-tree que provê suporte a consultas no espaço multidimensional, tais como *range queries* (RQ). A R-tree constrói uma hierarquia com os MBR dos objetos espaciais. A ineficiência dos demais MAM com relação ao tratamento de paginação em memória secundária, e ao desempenho na manipulação de grandes bancos de dados, foram os fatores que motivaram a proposição desta estrutura e que também justificaram sua popularização. A R-tree é usada para a indexação de objetos espaciais armazenados em memória secundária, e cada nodo corresponde a exatamente uma página de disco.

A R-tree possui dois tipos de nodos: folha e interno. A sua estrutura balanceada assegura que os nodos folha estejam sempre no mesmo nível. Todo nodo, independentemente de seu tipo, possui espaço para M entradas e armazena obrigatoriamente um número mínimo m de entradas (com $m \geq M/2$). O parâmetro m pode ser ajustado com o intuito de melhorar o desempenho da R-tree. O nodo raiz possui pelo menos dois filhos (exceto se for folha), sendo o único que pode ter menos que m entradas. A

Como se vê na Figura 7, a interseção de MBR diferentes em um mesmo nível da árvore pode não ser vazia. Quando isto ocorre, diz-se que os nodos correspondentes estão indexando a mesma "porção" do espaço naquela região da intersecção. Este é o caso do par de nodos internos (R_3, R_4), e do par de nodos folha (r_2, L_6), por exemplo.

Tabela 2 demonstra as características dos nodos internos e folhas. Exibe-se um exemplo de R-tree na Figura 7. Sua estrutura de dados é ilustrada na Figura 8.

Como se vê na Figura 7, a interseção de MBR diferentes em um mesmo nível da árvore pode não ser vazia. Quando isto ocorre, diz-se que os nodos correspondentes estão indexando a mesma "porção" do espaço naquela região da intersecção. Este é o caso do par de nodos internos (R_3, R_4), e do par de nodos folha (r_2, L_6), por exemplo.

Tabela 2 – Caracterizações dos nodos internos e folhas de uma R-tree.

Nodos internos	Nodos folha
Possui entre m e M filhos, exceto se for raiz. Para cada entrada <ptr, MBR>, MBR é o menor retângulo que envolve os retângulos do nodo filho apontado por ptr.	Para cada entrada <id, MBR>, MBR é o menor retângulo que envolve o objeto espacial identificado por id.

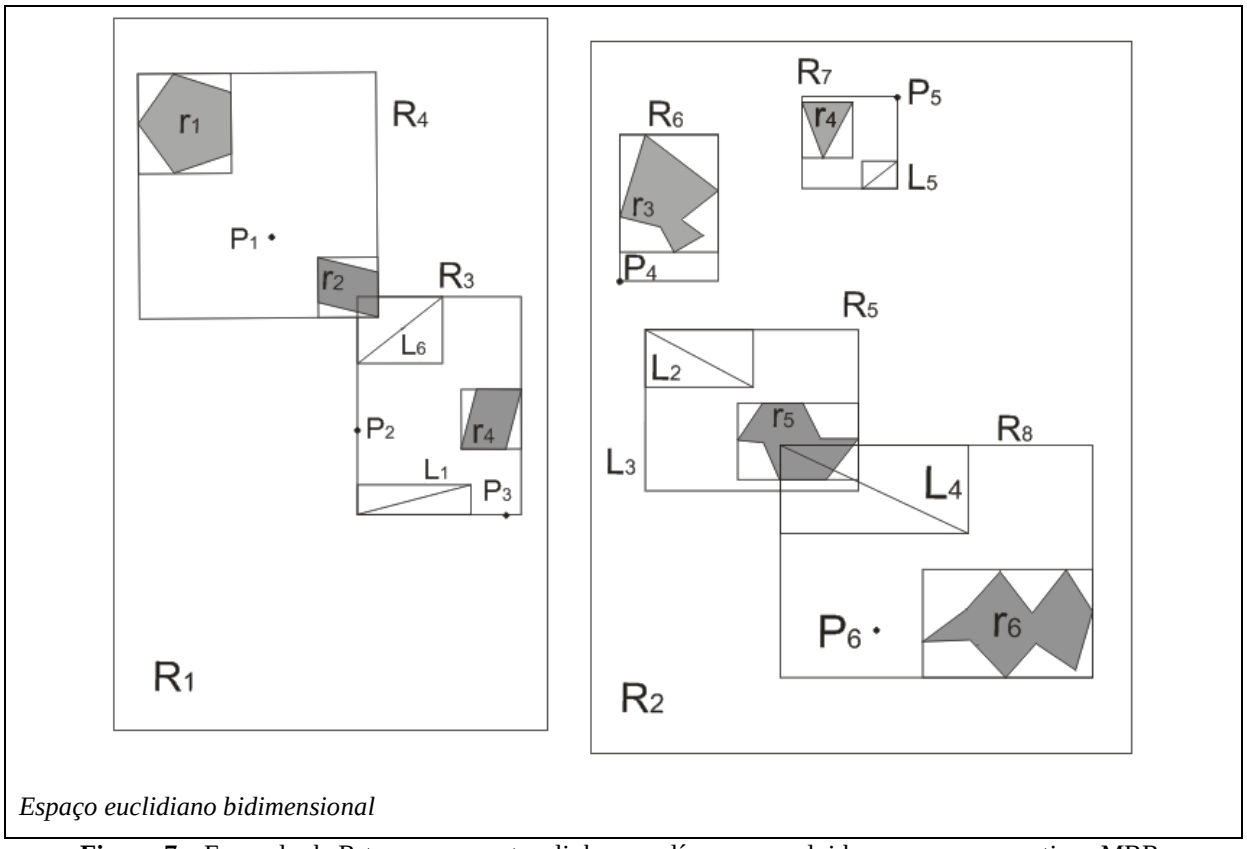


Figura 7 – Exemplo de R-tree, com pontos, linhas e polígonos envolvidos por seus respectivos MBR.

Nas duas próximas subseções são tratados os algoritmos de inserção e de busca sobre a R-tree. Considerando que a presente dissertação propõe uma estrutura de indexação para DWG, e que os dados num DWG não sofrem eliminações, não será abordado aqui o algoritmo de remoção da R-tree.

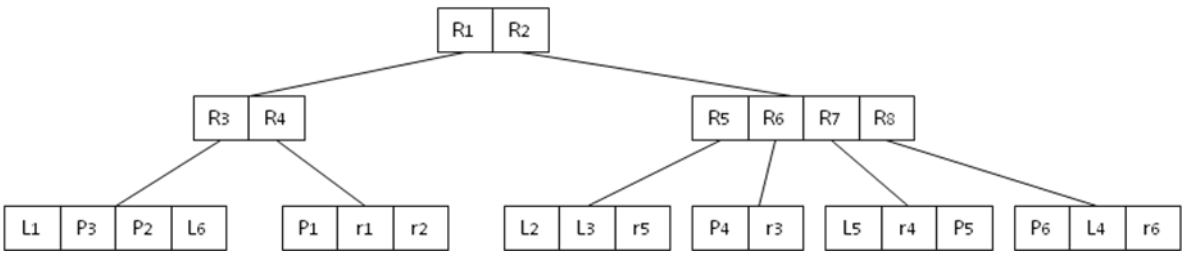


Figura 8 – Estrutura de dados da R-tree exemplificada na Figura 7.

2.4.1.1 Algoritmo de inserção

Similarmente à B-tree, todo novo objeto deve ser inserido nas folhas da R-tree, e a raiz deve emergir. O objetivo é minimizar a área total ocupada pelos MBR das entradas de cada nodo interno, a fim de reduzir a possibilidade de interseção da janela de consulta com os MBR das entradas de um nodo interno. Esta medida visa prevenir a ramificação do percurso inicial de busca.

Considerando uma árvore que possui apenas o nodo raiz, os objetos são inseridos até que a raiz fique totalmente cheia. Uma inserção com a raiz cheia causará a divisão da raiz em dois nodos. Os objetos são re-arranjados nestes dois nodos, respeitando o parâmetro `m`. Em seguida, calcula-se o MBR destes dois novos nodos. Os dois MBR mais um ponteiro para cada um dos respectivos nodos são inseridos em uma nova raiz, a qual emerge.

Depois que a raiz foi dividida, sempre que novos objetos são inseridos, um algoritmo de três passos guia o processo. No primeiro passo, a estrutura é percorrida a partir da raiz, seguindo pelos nodos internos cujos MBR sofrerão o menor aumento de área caso um de seus nodos folha venha a receber o novo objeto. Em caso de empate, escolhe-se a entrada cujo MBR tem a menor área. Este processo é realizado até atingir um nodo folha.

Se o nodo folha já mantiver um máximo de `M-1` entradas, então há espaço para alocar o objeto no nodo. O segundo passo consiste em fazer a verificação e a alocação. Se não houver o espaço, o nodo deve ser particionado. O particionamento produz uma nova folha e distribui as `M+1` entradas entre os dois nodos, de modo a respeitar o parâmetro `m`.

No terceiro passo, propagam-se as atualizações dos MBR das entradas aos nodos de níveis superiores e os efeitos do particionamento da folha, se este ocorrer. Logo, percorre-se o caminho inverso na árvore. A criação de uma nova folha implica na criação de uma nova entrada no nodo imediatamente superior. Este nodo poderá ter sua capacidade excedida, exigindo o seu particionamento. Isto requer atualizar os MBR das entradas dos nodos superiores. Sucessivamente, tal processo pode atingir até a raiz.

Para alcançar o objetivo de minimizar a área total ocupada pelos MBR das entradas de cada nodo interno, Guttman (1984) propôs três algoritmos distintos para executar o particionamento. O primeiro, exaustivo, investiga todas as possibilidades de agrupamento e encontra a melhor distribuição possível, mas tem custo exponencial em `M`. O segundo, quadrático, encontra uma área total pequena, mas não a menor possível, e tem complexidade quadrática. O último, linear, possui complexidade linear no número de entradas e de dimensões, mas nem sempre assegura encontrar uma pequena área total.

2.4.1.2 Algoritmos de busca

O algoritmo básico de busca da R-tree é voltado ao suporte de *intersection range query* (IRQ). O percurso no índice é *top-down*, similar ao da B-tree. Para todo nodo interno visitado, cada entrada passa por um teste que verifica se seu MBR intersecta a janela de consulta (JC). O percurso procede pelos nodos em que este relacionamento é satisfeito, visitando os nodos filhos. Assim, várias sub-árvores podem ser visitadas, gerando ramificações no percurso inicial. Alcançado um nodo folha, as entradas deste também passam pelo mesmo teste. Então, o percurso é encerrado, possibilitando colecionar um subconjunto do resultado da busca. Este resultado consiste num conjunto de objetos espaciais cujos MBR satisfazem o relacionamento de interseção com a JC.

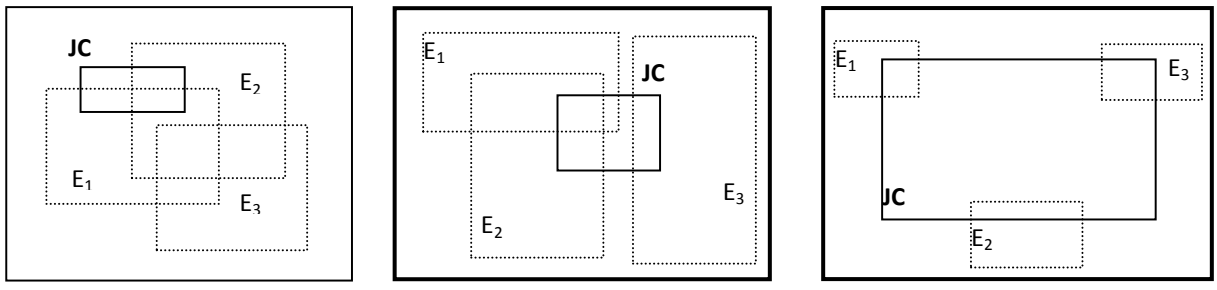
A ramificação do percurso inicial aumenta o número de caminhos de busca e, conseqüentemente, o número de visitas a nodos. Isto constitui forte restrição ao desempenho do algoritmo básico de busca, devido à necessidade de recuperação de diversas páginas de disco. Contudo, os mecanismos de otimização junto dos algoritmos de inserção e remoção de dados da R-tree são capazes de atenuar os prejuízos causados. Essas otimizações possibilitam que o algoritmo de busca torne-se livre de regiões irrelevantes do espaço multidimensional indexado, para que o percurso visite poucos nodos próximos à JC.

Caso a JC intersecte os MBR pertencentes a um grande número de entradas de um determinado nodo interno, a ramificação será acentuada. Isto ocorre sob três circunstâncias (CIFERRI, R., 2002):

a) na sobreposição entre os MBR das entradas de um nodo interno (Figura 9a);

b) no armazenamento de MBR grandes em relação ao espaço total indexado (Figura 9b); e

c) na escolha de uma JC abrangente em relação ao espaço total indexado (Figura 9c).



(a) Sobreposição de MBR (b) Armazenamento de MBR grandes (c) Janela de consulta abrangente

Figura 9 – Casos que levam à ramificação do percurso na R-tree.

O primeiro caso (Figura 9a) leva à ramificação do percurso caso a JC englobe parte de uma área onde há intersecção de MBR referentes a entradas da R-tree. A ramificação

ocorre somente para as entradas cujos MBR pertencem à área comum. Em especial, quanto maior o número de entradas participantes da área comum, mais degradado será o desempenho. Portanto, um alto número de sobreposições e uma grande área sobreposta aumentam a possibilidade de ramificação. Como a R-tree é uma estrutura dinâmica, é impossível evitar a sobreposição entre os MBR das entradas de nodos internos. Este caso é o mais comum, e tem particular relevância em bancos de dados que armazenam objetos de tamanhos variados (pequenos e grandes). Na Figura 9a, a JC intersecta a área comum de duas entradas: E_1 e E_2.

A segunda circunstância, sobre armazenar MBR grandes em relação ao espaço total indexado (Figura 9b), proporciona ramificação. A intersecção da JC e o MBR da entrada de um nodo interno é influenciada pelo tamanho deste MBR. Logo, quanto maior a área ocupada pelo MBR, maior a chance desta intersecção ocorrer. Um MBR grande pode ser gerado inserindo-se objetos espaciais extensos ou devido à alocação imprópria de um ou mais objetos em um mesmo nodo folha. Esta circunstância já havia sido antecipada por Guttman (1984), o qual aprimorou os pontos de otimização do algoritmo de inserção no sentido de reduzir a área total representada por um ou mais MBR.

Na terceira situação, quando há uma JC abrangente, fica praticamente inevitável a ocorrência de várias intersecções entre a JC e os MBR das entradas de um nodo interno. Deste modo, os pontos de otimização são suprimidos. Isto é ilustrado na Figura 9c.

Por meio de adaptações, o algoritmo básico de busca passa a prover suporte a CRQ. Logo, a intersecção da JC com o MBR de um nodo interno significa que pode ser verdadeiro o relacionamento espacial **está contido** entre a JC e o MBR de algum objeto espacial alcançável a partir da entrada analisada. Portanto, o percurso deve obrigatoriamente se estender até os nodos folha. Conforme mostrado na Figura 10a, só é possível determinar se o_1 e o_3 estão contidos na JC checando-os no nodo-folha. O mesmo acontece com o_1 na Figura 10b. Por outro lado, conclui-se que o relacionamento **intersecta** colabora decisivamente para eliminar os testes sobre o_2 na Figura 10a, e o_2 e o_3 na Figura 10b.

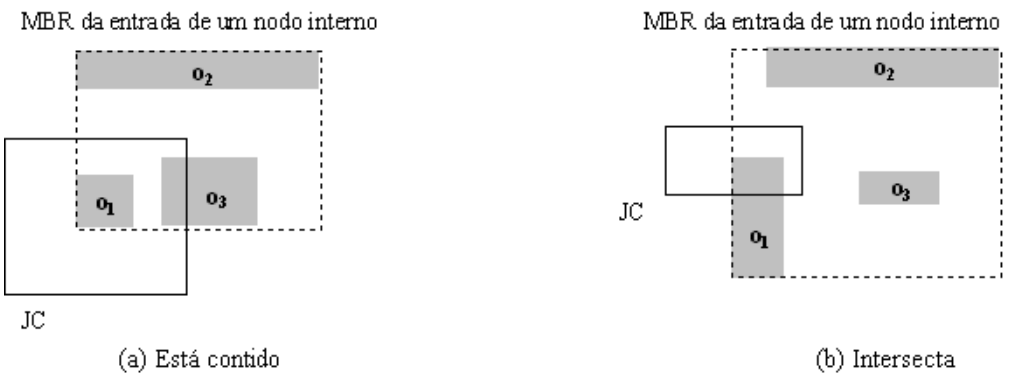


Figura 10 – Relacionamentos de (a) inclusão e (b) interseção para nodos internos da R-tree.

Da mesma forma, pode-se prover suporte à ERQ. Nela, em geral, é desnecessário descer até os nodos folha. O fato do MBR da entrada de um nodo interno não englobar a JC elimina qualquer possibilidade de um MBR de objeto espacial alcançável a partir desta entrada conter a JC. Por outro lado, se o MBR de um nodo interno engloba a JC, não obrigatoriamente haverá um MBR que **contém** a JC. Logo, pode ser necessário descer até os nodos folha. Na Figura 11 observa-se que o MBR da entrada de um nodo interno: (a) contém a JC e também possui um MBR de objeto espacial que contém a JC; (b) não possui nenhum MBR que contém a JC, apesar da entrada conter a JC; (c) não contém a JC e, portanto, não mantém nenhum MBR cujo objeto espacial possa conter a JC. Em todos os casos, todas as entradas são testadas para verificar quais delas contêm a JC.

O fato da R-tree empregar aproximações exige, para IRQ e ERQ, que a representação exata dos objetos espaciais seja acessada para a verificação da validade do relacionamento espacial. Isto porque alguns objetos espaciais pertencentes à resposta de uma busca podem ser falsos candidatos: a interseção da JC com o MBR do objeto pode acontecer apenas sob a área de *dead space* (em uma IRQ), ou a janela de consulta pode estar contida pelo *dead space* (em uma ERQ).

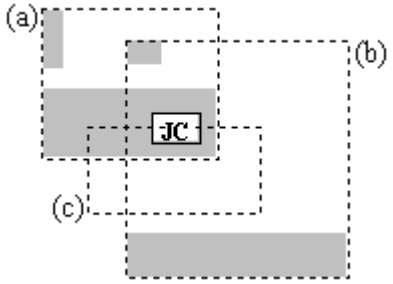


Figura 11 – Diferentes casos do relacionamento de inclusão (contém).

2.5 Considerações Finais

Enquanto para DW o modelo multidimensional conceitual e o modelo lógico são consolidados, o mesmo não se pode dizer sobre o DWG. A integração dos dados espaciais oriundos de SIG sob o contexto multidimensional e relacional tem motivado diferentes trabalhos. Não há consenso sobre a definição do que é uma medida espacial, diferentemente do que ocorre com uma medida em um DW convencional. Por um lado, os modelos conceituais para DWG proporcionam flexibilidade e buscam conferir as mesmas características do OLAP tradicional ao SOLAP. Por outro lado, estes modelos não são imediatamente operacionais, e lhes falta validação por meio de uma implementação adequada. Permanece em aberto a criação de métodos eficientes que possam diminuir o tempo de

resposta no processamento de consultas sobre DWG. Nesta dissertação, o foco foi direcionado à indexação.

B-tree e *hashing* são índices comprovadamente eficientes e empregados pelos SGBD, em OLTP. Todavia, igualmente importante é frisar que ambos não são adequados ao caso espacial. Por isso, foi descrita a R-tree, que consiste na estrutura mais referenciada e utilizada com a finalidade de indexar dados espaciais. Ela indexa dados multidimensionais usando **aproximações** retangulares (MBR) dos objetos espaciais. Ainda, apesar de sua robustez (grande parte dela herdada da B-tree) e de suas vantagens, Beckmann et al. (1990) propuseram a R*-tree visando diminuir a sobreposição de áreas cobertas por nodos da árvore. Outros índices derivados de R-tree foram propostos criando a principal família de índices para dados espaciais: R+-tree (SELLIS, ROUSSOPOULOS; FALOUTSOS, 1987), R*-tree (BECKMANN et al., 1990), Parallel R-tree (KAMEL; FALOUTSOS, 1992) e Hilbert R-tree (KAMEL; FALOUTSOS, 1994), dentre outras. É válido ressaltar que a estrutura da R-tree foi utilizada por Papadias et al. (2001) para propor um índice para DWG denominado aR-tree. Maiores detalhes sobre a aR-tree se encontram na Seção 3.4.

3. TRABALHOS CORRELATOS

O tipo e a complexidade das consultas, o enorme volume de dados armazenado no DW e a predominância de consultas em relação a operações de atualização são fatores que tornam inadequadas para DW as técnicas de indexação projetadas para SGBD tradicionais e ambientes OLTP. Por exemplo, os trunfos do índice B-tree são a sua estrutura dinâmica, além de seu bom desempenho e estabilidade diante de sucessivas operações de atualização. Todavia, estas características são irrelevantes num DW (WU, M.; BUCHMANN, 1998). Por outro lado, esses fatores mencionados enfatizam a importância do eficiente processamento de consultas em um ambiente OLAP, motivando a criação de estruturas de indexação que aperfeiçoem seu desempenho.

Segundo Sarawagi (1997), a estrutura multidimensional do DW e a grande quantidade de atributos nas tabelas de dimensão de um esquema estrela induzem à adaptação da R-tree. Considerando que o hipercubo geralmente é esparso e que suas regiões densas geralmente têm forma de retângulo, a R-tree pode ser construída criando nodos das regiões densas do hipercubo (que contêm mais que um limiar de pontos em seu interior), e nodos para pontos em regiões esparsas. Apenas os limites dos retângulos das regiões densas são armazenados no índice, e os pontos em seu interior ficam num vetor multidimensional à parte.

Encontrar grupos densos é uma tarefa que pode ser realizada pelo próprio administrador do DW, ou pela aplicação de algoritmos de agrupamento (*clustering*) adaptados para reunir retângulos. Cada entrada de grupo denso na R-tree mantém seu retângulo e um ponteiro para um vetor de comprimento variável. Sobre o vetor podem ser aplicadas técnicas de agrupamento para melhorar a sua organização espacial e evitar a disposição linear. Deve-se ressaltar que, originalmente, a R-tree foi projetada para indexar dados espaciais com duas e três dimensões, preferencialmente.

A aR-tree (PAPADIAS et al., 2001) é um índice para DWG baseado na R-tree que mantém em suas entradas os valores gerados pela função de agregação sobre uma medida da tabela de fatos. Este índice se beneficia do algoritmo de particionamento do espaço da R-tree a fim de criar hierarquias *ad-hoc* entre os objetos espaciais. Papadias et al. (2002) modelam as dimensões espacial e temporal de um DWG em uma mesma dimensão combinada no hipercubo. Para tanto, integram a indexação espaço-temporal com pré-agregação de medidas, sob uma implementação de múltiplas árvores B-tree e R-tree. O resultado foi a criação de dois novos índices: a3DR-tree e aRB-tree.

Deve-se frisar que os métodos de acesso multidimensionais são eficientes apenas em determinados tipos de consulta, e têm seu desempenho degenerado se existem numerosas dimensões (por exemplo, para mais que 10 dimensões) e se predicados envolvem múltiplos atributos, requerendo o produto cartesiano das chaves envolvidas (SARAWAGI, 1997). Além disso, a estrutura dinâmica e o bom desempenho e estabilidade diante das operações de atualização, proporcionados pela R-tree, também são irrelevantes no DW.

O Índice Bitmap, por outro lado, é uma estrutura de indexação para DW cujo desempenho não degenera em grande escala mesmo quando há muitas tabelas de dimensão no esquema estrela (O'NEIL, P.; GRAEFE, 1995; SARAWAGI, 1997; O'NEIL, P.; QUASS, 1997; GOYAL; ZAVERI; SHARMA, 2006; O'NEIL, E.; O'NEIL, P.; WU, K., 2007). Sua vantagem reside em manipular vetores que usam a menor unidade de informação, o bit, por meio de operações lógicas que são auxiliadas pelo *hardware*. Porém, atributos cujos domínios possuem muitos valores distintos deterioram a eficiência do Bitmap. No entanto, as técnicas de *binning*, compressão e codificação são propostas justamente para atenuar estes efeitos negativos (WU, K.; OTOO; ARIE, 2006; STOCKINGER; WU, K., 2007; WU, K.; STOCKINGER; SHOSHANI, 2008). Embora o Índice Bitmap constitua um método de acesso promissor, não foi encontrado, na literatura consultada, nenhum registro de seu uso em DWG. Neste Capítulo são tratados índices para DW e para DWG, partindo das premissas anteriores. Nas Seções 3.1, 3.2 e 3.3 são descritos índices para DW: o Índice de Projeção, o Índice Bitmap e o *software* FastBit, respectivamente. Nas Seções 3.4 e 3.5 discorre-se sobre índices para DWG: aR-tree, a3DR-tree e aRB-tree. Na Seção 3.6 são feitas considerações finais.

3.1 Índice de Projeção

Um Índice de Projeção (O'NEIL, P.; QUASS, 1997) sobre a coluna **X** de uma tabela **T** consiste em uma seqüência de valores de **X** ordenadas segundo a numeração das tuplas em **T**. Espaços vagos podem existir se números não são utilizados, e valores duplicados podem ocorrer na estrutura. O Índice de Projeção difere do particionamento vertical porque conserva a ordem das tuplas e não elimina tuplas replicadas, já que pode ser construído para um atributo não chave. Na Figura 12a exibe-se uma fração dos dados da tabela de dimensão *fornecedor*, do esquema estrela exemplificado na Figura 5 (Capítulo 2). Já na Figura 12b está disposto o Índice de Projeção para o atributo *nacao*.

Com as tuplas numeradas de 0 a n, o valor de **X** em uma delas é facilmente calculado a partir de n, e vice-versa. Por exemplo, supondo que uma página de disco pode

alocar 1000 valores de **X**, é possível acessar um valor **X** = *x* em sua página p e *slot* s em disco pelo cômputo de p = n ÷ 1000 e s = n MOD 1000, respectivamente. E, dado um valor **X** = *x* em uma posição do índice, obtém-se a posição da tupla por n = 1000 × p + s. Por *slot*, entende-se uma porção de espaço no disco rígido que pode ser endereçada.

fornecedor_pk	endereco	cidade	nacao	regiao
1	A	Tam Ky	Vietnã	Ásia
2	B	Bordeaux	França	Europa
3	C	Craiova	Romênia	Europa
4	D	Thessa	Argélia	África
5	E	Oran	Argélia	África

(a) Tabela de dimensão: fornecedor.

nacao
Vietnã
França
Romênia
Argélia
Argélia

(b) Índice de Projeção: nacao.

Figura 12 – Uma tabela de dimensão e um Índice de Projeção para um de seus atributos.

Podem-se adiantar, aqui, dois fatos sobre o Índice de Projeção:

a) ele é empregado para armazenar os dados de forma vertical pelo software FastBit, o qual é tratado na Seção 3.3; e
b) sua estrutura foi adaptada para dar origem à estrutura de dados do índice apresentado nesta dissertação, o SB-index.

3.2 Índice Bitmap

A origem do Índice Bitmap não possui relação com DW e OLAP. De fato, ele antecede inclusive os sistemas de banco de dados relacionais. Em seu princípio, era tido como uma forma de arquivo invertido (KNUTH, 1998). Seu nome se popularizou da forma que se conhece atualmente a partir dos trabalhos de O'Neil, P. et al. (1995; 1997). Também foram estes que introduziram o Índice Bitmap para uso em OLAP. Jürgens e Lenz (1999) observaram que este índice tende a superar o desempenho de estruturas baseadas em árvores (como é o caso da R-tree), devido à evolução na tecnologia de discos rígidos.

Um Índice Bitmap sobre a coluna **X** de uma tabela consiste em uma seqüência ordenada de valores de chave, representando os valores distintos que a coluna pode assumir. Cada valor de chave é associado a um vetor de bits, que especifica o conjunto de linhas na tabela em que a coluna **X** assume aquele valor. Todo vetor possui tantos bits quantas são as linhas na tabela. O i-ésimo bit no vetor é fixado em 1 se o valor de **X** na i-ésima tupla da tabela é igual ao valor de chave do vetor associado, e em 0 caso contrário.

Para mapear o valor inteiro que determina a posição de um bit do vetor no endereço efetivo da tupla, usa-se um identificador para ela. O identificador é um número inteiro capaz de indicar a página e a porção de espaço em disco onde a tupla se aloca, de forma análoga ao índice de projeção. Se as tuplas possuem tamanhos variáveis, fixa-se a quantidade de tuplas que podem existir em uma página de disco.

No contexto de um DW, o Índice Bitmap pode ser criado para evitar a computação de junções intrínsecas do esquema estrela, as quais ocorrem entre tabelas de dimensão e a tabela de fatos. Este é o Índice Bitmap de Junção (IBJ) (O'NEIL,P.; GRAEFE, 1995), que relaciona os valores de um ou mais atributos de uma tabela de dimensão às tuplas correspondentes na tabela de fatos (CHAUDHURI; DAYAL, 1997).

Considere-se o esquema estrela da Figura 13a para os exemplos de IBJ nas Figuras 13b e 13c, e que há um relacionamento 1:1 entre e *fornecedor_pk* e *endereco*, bem como *fornecedor_fk* é uma chave estrangeira para o atributo *fornecedor_pk*. Logo:

a) um Índice Bitmap sobre o atributo *fornecedor_fk* é um IBJ para o atributo *fornecedor_pk*, uma vez que *fornecedor_fk* possui uma chave estrangeira para *fornecedor_pk*;
b) o IBJ para o atributo *fornecedor_pk* associa cada tupla da tabela de dimensão fornecedor a um conjunto de tuplas da tabela de fatos;
c) conforme as Figuras 13a e 13b, *fornecedor_pk* = 1 ocorre na primeira e na segunda tupla da tabela de fatos, *fornecedor_pk* = 2 ocorre na terceira tupla da tabela de fatos, e assim por diante;
d) se o fornecedor 1 está relacionado ao endereço A, o fornecedor 2 está relacionado ao endereço B e assim por diante, então o vetor de bits para *fornecedor_fk* = 1 é idêntico ao vetor de bits para *endereco* = 'A', o vetor de bits para *fornecedor_fk* = 2 é idêntico ao vetor de bits para *endereco* = 'B', e assim por diante; e
e) conforme a Figura 13c, *endereco* = 'A' ocorre na primeira e na segunda tuplas da tabela de fatos, *endereco* = 'B' ocorre na terceira tupla da tabela de fatos, e assim por diante.

fornecedor_fk	**item_fk**	receita
1	235	20000
1	512	16700
2	512	22870
3	235	19960
3	512	15710
3	106	21950
4	235	20550
5	106	18740

(a) A tabela de fatos vendas

1	2	3	4	5
1	0	0	0	0
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

(b) IBJ para *fornecedor_pk*

A	B	C	D	E
1	0	0	0	0
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

(c) IBJ para *endereco*

Figura 13 – Construção de IBJ para os atributos *fornecedor_pk* e *endereco*.

Portanto, embora *fornecedor_pk* e *endereco* não estejam presentes na tabela de fatos, é possível indexá-los por um IBJ devido à existência da chave estrangeira em *fornecedor_fk*. Ainda, de acordo com Harinarayan, Rajaraman e Ullman (1996), $Q_1 \prec Q_2$ se, e somente se, é possível responder a Q_1 usando apenas os resultados de Q_2, e $Q_1 \neq Q_2$. Deste modo, é possível indexar *cidade*, *nação* e *região*, uma vez que região $\prec$ nação $\prec$ cidade $\prec$ endereço. Por exemplo, executando uma operação lógica OR bit-a-bit com os vetores de bits

de *endereço* = 'B' e *endereço* = 'C' resulta no vetor de bits para *região* = 'Europa'. Na Figura 14 exibe-se a construção do IBJ para o atributo *região*.

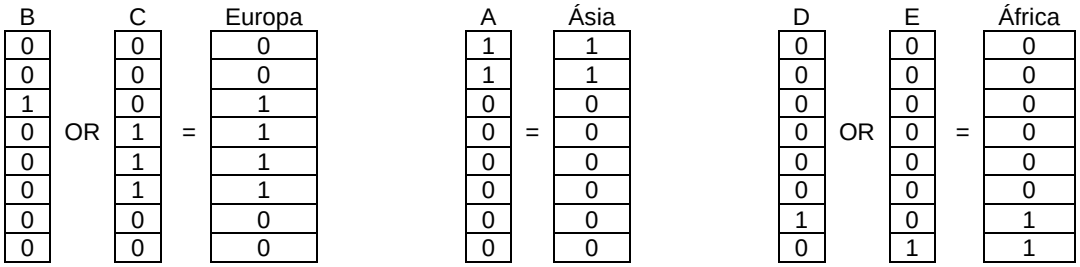


Figura 14 – Construção de IBJ para o atributo *regiao*.

Nitidamente, criar IBJ para as hierarquias de atributos é suficiente para assegurar as operações *roll-up* e *drill-down*. Por exemplo, para resolver o predicado endereço = 'B' ∧ marca = 'MFGR#2239', deve-se realizar uma operação lógica `AND` bit-a-bit entre os vetores de bits criados para os valores B e MFGR#2239, pertencentes ao IBJ criado para *endereço* e *marca*, respectivamente. De forma análoga, pode-se executar um *roll-up* resolvendo os seguintes predicados:

a) cidade = 'Bordeaux' ∧ marca = 'MFGR#2239';

b) nação = 'França' ∧ marca = 'MFGR#2239'; e

c) região = 'Europa' ∧ marca = 'MFGR#2239'.

A complexidade do processamento de uma consulta é função do número de vetores de bits que são acessados e do número de operações lógicas realizados sobre estes vetores. Consultas por valor exato em uma ou mais dimensões podem ser respondidas pela interseção de vetores. Por exemplo: “qual a receita das vendas do item 512 na Europa?”. É requerida uma operação `AND` bit-a-bit sobre os vetores correspondentes. O item 512 está associado ao vetor 01101000, e Europa ao vetor 00111100, como mostram a tabela de fatos da Figura 13 e o vetor para Europa na Figura 14. A operação `AND` bit-a-bit entre os dois vetores resulta em 00101000, apontando a 3ª e a 5ª tuplas como aquelas que devem ter a receita somada para responder à consulta.

As respostas a alguns tipos de consulta por faixa de valores vêm da aplicação da operação `OR` sobre vetores de bits relativos a diferentes valores na mesma dimensão, seguida da operação `AND` entre o resultado da operação `OR` e os vetores de bits construídos para as outras dimensões. Para ilustrar esta situação, é adaptado um exemplo de Garcia-Molina et al. (2000) referente a uma joalheria, e ilustrado na Figura 15. À esquerda, há uma tabela de fatos com as chaves das dimensões Idade e Salário e a medida Venda$. Esta medida representa o montante em milhares em unidade monetária alcançado com as vendas a clientes com salário em milhares e idade em anos. Os vetores Bi indexam Idade, e Bs o Salário.

Idade	Sal	Venda$	Bi_{25}	Bi_{45}	Bi_{50}	Bi_{70}	Bs_{60}	Bs_{75}	Bs_{100}	Bs_{120}
25	60	120	1	0	0	0	1	0	0	0
45	60	140	0	1	0	0	1	0	0	0
50	75	200	0	0	1	0	0	1	0	0
50	100	210	0	0	1	0	0	0	1	0
50	120	340	0	0	1	0	0	0	0	1
70	120	290	0	0	0	1	0	0	0	1

Figura 15 – Índices Bitmap para a tabela de fatos do DW da joalheria.

Se o objetivo é calcular o montante de vendas para os compradores de jóias com idade na faixa 45-55 e salário entre 70-150, então o critério idade toma os bitmaps Bi_{45} = 010000 e Bi_{50} = 001110 e efetua OR bit-a-bit entre eles, o que resulta em 011110. Já para salário, os bitmaps Bs_{75}, Bs_{100} e Bs_{120} se submetem à disjunção lógica, logo 001000 OR 000100 OR 000011 resulta em 001111. O próximo passo é realizar AND bit-a-bit nos resultados de cada dimensão: 011110 AND 001111 = 001110. Portanto, os montantes de vendas a serem somados são os da terceira, quarta e quinta tupla, totalizando 750.

A Figura 14 também exemplifica como o Índice Bitmap lida com a redundância de dados, intrínseca em um esquema estrela. Por exemplo, no vetor de bits criado para *região* = 'Europa', nota-se que existem 4 bits indicando as quatro tuplas da tabela de fatos onde *região* = 'Europa'. Logo, ao invés de armazenar quatro vezes este valor descritivo textual, o Índice Bitmap assinala quatro bits com o valor "1" no vetor correspondente.

Dentre as vantagens do Índice Bitmap, Sarawagi (1997) e O'Neil, E., et al. (2007) destacam:

a) o acesso aos dados está agrupado, já que a organização de um vetor de bits corresponde àquela em que os dados estão fisicamente armazenados;
b) todas as dimensões são tratadas simetricamente, e dados esparsos podem ser manipulados da mesma forma como são os densos;
c) se necessário, os dados podem ser recuperados segundo uma classificação, pelo percurso do vetor de bits em ordem arbitrária. Porém, o agrupamento do item (a) é perdido; e
d) uma vez que o *hardware* provê suporte a operações lógicas bit-a-bit como OR, AND e NOT, os predicados SQL podem ser avaliados de forma bastante veloz.

Dentre as desvantagens, destacam-se:

a) o número de operações OR pode ser muito grande para certas consultas, já que cada valor de uma dimensão em uma faixa de valores implica em uma operação destas, aumentando o custo do processamento da consulta. Por outro lado, as operações AND são limitadas ao número de dimensões, que não é excessiva na maioria dos casos;

b) atualizações em lote podem também ser custosas, uma vez que todos os vetores de bits devem ser modificados mesmo se houver uma única inserção de tupla. Ainda, Stockinger e Wu, K. (2007) apontam que soluções eficientes para a questão da atualização podem ser a chave para melhor difundir a adoção do Bitmap em aplicações comerciais; e
c) especificamente no caso do IBJ, se existem numerosas colunas usadas para as restrições de junção, então o número de IBJ necessários para combinar restrições arbitrárias a partir de cada tabela de dimensão é um produto do número de colunas em cada dimensão. Ocorre então uma explosão combinatória, em termos do número de colunas (O'NEIL, P.; QUASS, 1997). Este problema é análogo ao de armazenar os diversos níveis de agregação da treliça.

Em dissonância está a questão da cardinalidade do domínio do atributo indexado. Considerando que N é o número de tuplas de uma tabela T, e que $|\mathbf{X}|$ denota a cardinalidade da coluna **X,** é simples concluir que cada Índice Bitmap, como os das Figuras 13, 14 e 15, necessitam de $|\mathbf{X}|$ vetores de N bits. No pior caso **X** apresenta apenas valores distintos, então $|\mathbf{X}| = N$. Conseqüentemente, tal Índice Bitmap requer N^2 bits. Num conjunto de dados grande, com milhões ou bilhões de tuplas, este índice pode ser até mesmo maior que a tabela indexada.

Sarawagi (1997) e O'Neil, E., O'Neil, P. e Wu, K. (2007) alegam que para altas cardinalidades há grande perda de desempenho associada ao armazenamento de todos os vetores. Argumentam ainda que, se por um lado técnicas de compressão trazem ganhos em espaço de armazenamento, as operações sobre o índice comprimido são mais lentas com implementação complicada. Stockinger e Wu, K. (2007) e Wu, K., Stockinger, Shoshani (2008), por sua vez, afirmam e demonstram que o Índice Bitmap pode ser eficiente se aliado a técnicas de *binning*, compressão e codificação. Estas técnicas são tratadas nas Seções 3.2.1, 3.2.2 e 3.2.3, respectivamente.

Sarawagi (1997) afirma que o Índice Bitmap pode ser implementado como uma B-tree na qual, ao invés de serem armazenados identificadores de tuplas para cada valor de chave nos nodos folha, armazena-se um vetor de bits. Porém, Stockinger e Wu, K. (2007), consideram que a B-tree é empregada apenas no arranjo de chaves, sendo os vetores de bits armazenados separadamente em arquivos. Eles ainda argumentam que o Índice Bitmap, desta forma, é independente da B-tree.

O SB-index, proposto nesta dissertação, visa introduzir o Índice Bitmap em DWG, mais especificamente usando os recursos do IBJ. Além disso, foi empregada uma técnica de compressão sobre os IBJ criados.

3.2.1 Binning

A idéia básica de *binning* é construir um vetor de bits para uma caixa (chamada de *bin*), ao invés de criá-lo para cada valor de atributo. Deste modo, o número de vetores torna-se independente da cardinalidade do atributo. Uma vantagem clara desta abordagem é que permite controlar o tamanho da estrutura. Porém, uma consulta pode ter como resposta falsos candidatos, se apenas o "índice encaixotado" for consultado. Neste caso, é indispensável que o resultado passe por uma fase de refinamento, envolvendo o custoso acesso aos dados originais.

A Figura 16 exibe um exemplo de um Índice Bitmap. O atributo **X**, que representa a receita total em milhões obtidas pelos fornecedores situados na América, tem valores entre 0 e 100, mostrados na segunda coluna a partir da esquerda. A faixa de possíveis valores de **X** tem as caixas [0;20), [20;40), [40;60), [60;80) e [80;100). Um bit fixado em "1" num vetor de bits B_i significa que o valor de **X** está associado a uma caixa específica. Se fixado em "0", indica que o valor do atributo não pode ser atribuído àquela caixa.

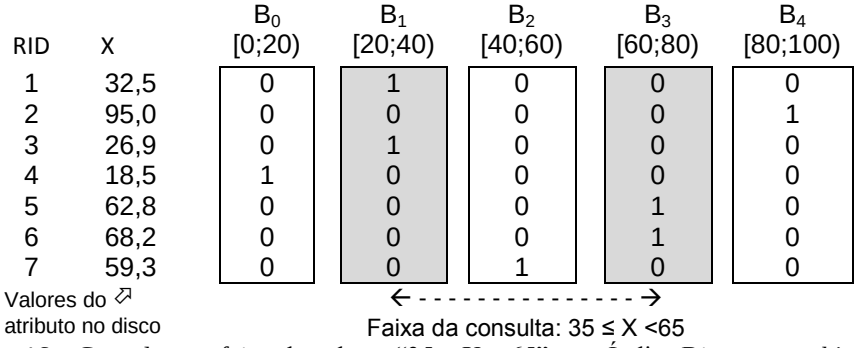


Figura 16 – Consulta por faixa de valores "$35 \leq X < 65$" num Índice Bitmap com *binning*.

No processamento da consulta "conte o número de linhas em que $35 \leq \mathbf{X} < 65$", a resposta correta deve ser 2 (tuplas 5 e 7). Vê-se que a faixa na consulta sobrepõe as caixas B_1, B_2 e B_3. Certamente, todas as tuplas que "caem" na caixa B_2 definitivamente são respostas. Por outro lado, tuplas que "caem" nas caixas B_1 e B_3 podem ser falsas candidatas. Neste caso, as caixas B_1 e B_3 são ditas fronteiras. As tuplas que "caem" sobre B_1 e B_3 são apenas tuplas candidatas, como as tuplas 1 e 3 da caixa B_1 e 5 e 6 da caixa B_3. Elas devem obrigatoriamente passar pelo refinamento, que confere nos dados originais se os valores realmente satisfazem ou não à consulta. Assim, após o refinamento, as tuplas 1, 3 e 6 são descartadas da resposta final da consulta por estarem fora do intervalo 35 a 65. Num conjunto de dados grande, o refinamento pode requerer a leitura de muitas páginas de disco, fator determinante sobre o tempo de processamento total de uma consulta OLAP.

Binning tem se destacado como a mais promissora das técnicas para aumentar o desempenho do Índice Bitmap em domínios com alta cardinalidade. O trabalho de Wu, K.,

Stockinger e Shoshani (2008), por exemplo, propõe uma técnica de *binning* capaz de indexar eficientemente atributos com altíssima cardinalidade. Nos experimentos dos referidos autores, foram usados atributos com cardinalidade próxima de 25 milhões de valores distintos, associados ao estudo de explosão estelar supernova.

3.2.2 Compressão

A compressão é mais uma estratégia para reduzir o tamanho do Índice Bitmap. Como cada vetor de bits da estrutura deve ser usado separadamente dos demais, a compressão é tipicamente aplicada individualmente sobre cada vetor. As operações no processamento das consultas, contudo, são mais lentas em vetores comprimidos. Os métodos mais relevantes são *Byte-aligned Bitmap Code* (BBC) (ANTOSHENKOV, 1995; JOHNSON, 1999) e *Word-Aligned Hybrid code* (WAH) (WU, K.; OTOO; ARIE, 2006).

Vetores de bits comprimidos com BBC são ligeiramente maiores, mas têm operações mais velozes que os vetores comprimidos pelos métodos de compressão de propósito geral. Evidencia-se o equilíbrio que deve existir entre tempo de resposta e espaço de armazenamento. WAH, por sua vez, aguça este equilíbrio: seus vetores comprimidos são maiores que os de BBC, todavia seu tempo de resposta a consultas é muito menor que o de BBC. Isto porque as operações lógicas em vetores comprimidos por WAH são mais ágeis que aquelas sobre os vetores em BBC. Apenas WAH será abordada neste trabalho.

Considere-se que uma corrida é uma seqüência de bits. A técnica WAH se baseia na codificação por comprimento da corrida (RLE: *run-lenght encoding*), em que os bits consecutivos idênticos são representados com seu valor 1 ou 0, e o comprimento da corrida (*run*). Ou seja, para a seqüência 00000 tem-se a representação 0101: uma seqüência de zeros com $(101)_2 = (5)_{10}$ elementos. Na técnica WAH, uma corrida consiste em um preenchimento (*fill*) e uma cauda (*tail*). O primeiro é o conjunto de bits idênticos consecutivos, representado pelo comprimento e seus valores (1 ou 0). A segunda é um conjunto de 0’s e 1’s misturados, representado literalmente dados sem compressão. A idéia chave de WAH é definir os preenchimentos e as caudas de forma que possam ser armazenadas em palavras (unidades operacionais de hardware).

Um exemplo, adaptado de Stockinger, Wu, K. (2007) é exibido na Figura 17. Assumindo que uma palavra de máquina tem 32 bits, o exemplo mostra como uma seqüência de 5456 bits (vide Figura 17a), que denota um vetor de bits, é repartida em duas corridas e codificada em três palavras. Conceitualmente, a seqüência de bits é antes dividida em grupos de 31 bits cada (Figura 17b). Depois, os grupos vizinhos com bits idênticos são fundidos

(Figura 17c). Finalmente, os três grupos são codificados como palavras de máquina de 32 bits (Figura 17d).

A primeira corrida do exemplo contém um preenchimento de comprimento 0 e uma cauda, não apresentando nenhuma palavra de preenchimento, e sim apenas uma palavra literal (literais não são comprimidos) que representa a cauda de 31 bits para esta corrida. A segunda corrida contém um preenchimento de comprimento 174 (representa 174 grupos de 31 bits cada) e uma cauda, requerendo uma palavra para o preenchimento e outra para a cauda. O primeiro bit de uma palavra de preenchimento a identifica como tal, e o segundo bit mantém o valor do preenchimento: 0 no caso deste exemplo. Os 30 bits restantes armazenam o comprimento do preenchimento, $(10101110)_2 = (174)_{10}$ neste caso.

O tempo de resposta a consultas por faixas de valores usando vetores de bits codificados por WAH cresce linearmente no número de acertos, como nas árvores B+ e B*. Porém, não há como combinar estas árvores de forma eficiente a fim de responder a tais consultas *ad-hoc* no âmbito multidimensional, ao contrário do Índice Bitmap (STOCKINGER; WU, K., 2007).

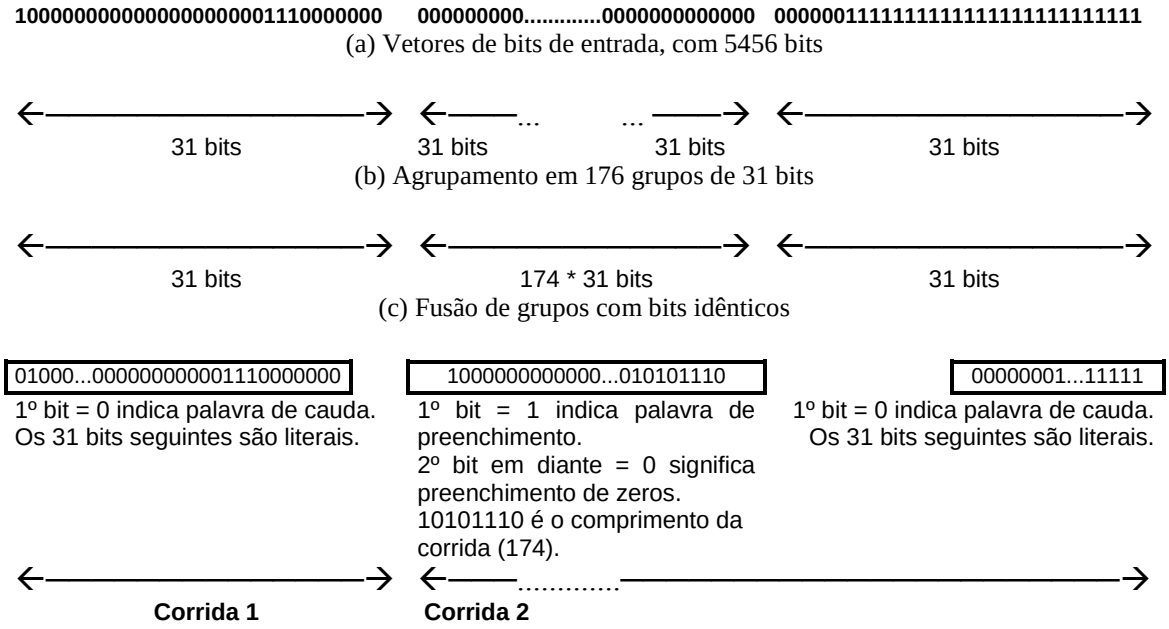


Figura 17 – Exemplo da codificação WAH de uma seqüência de 5456 bits em uma máquina de 32 bits.

Goyal, Zaveri e Sharma (2006) propõem uma nova estratégia para usar o Índice Bitmap, que pode ser aplicada a qualquer esquema de compressão baseado em RLE e suas variantes. Considerando que longas cadeias de 0's e 1's são desejáveis neste método, classifica-se a tabela segundo a coluna que se almeja indexar, e então cria-se um Índice

Bitmap de agrupamento sob a referida coluna. Porém, apenas um exemplar deste índice é permitido por tabela indexada.

A classificação da tabela agrega um custo extra em tempo, e por isso deve ser realizada enquanto o DW estiver *offline*. Por outro lado, os ganhos em termos de tempo de processamento de consultas e de redução de espaço de armazenamento do índice justificam sua existência. A escolha da coluna que norteará a classificação também ocorre no período de inatividade. Se por alguma razão for escolhida outra coluna, o DW só estará indexado após novo carregamento.

Os testes foram realizados por Goyal, Zaveri e Sharma (2006) para as consultas EQ (*Exact Query*) e RQ, sob os esquemas de compressão WAH e BBC. Os resultados apontam que, com a estratégia de classificação, a compressão e o desempenho de WAH e BBC são melhores que os obtidos sem ela. Isto vale para todas as combinações de cardinalidade de colunas e tabelas, e para consultas por valor exato e por faixas de valores.

3.2.3 Codificação

O Índice Bitmap básico discutido na Seção 3.2 é também chamado de Bitmap codificado por igualdade, já que cada vetor de bits indica se o valor de um atributo é ou não igual à chave. Esta estratégia é a mais eficiente para consultas por valores exatos, como "umidade = 32". Chan e Ioannidis (1999) desenvolveram a codificação por faixa. Ela visa o processamento de consultas por faixa de valores com um operador, como "pressão < 1".

Na Figura 18, adaptada de Stockinger e Wu, K. (2007), vê-se a codificação para o valor 2 da coluna **X** da relação **R**. Na codificação por igualdade (Figura 18b), o vetor E_2 é fixado em "1", e os demais bits na mesma posição horizontal em "0". Na codificação por faixa (Figura 18c), todos os bits entre R_2 e R_8 são fixados em "1", e os outros em "0". Este método é bastante eficiente. Por exemplo, na consulta "$X \leq 4$", no máximo um vetor, R_4, deve ser acessado e percorrido no processamento da consulta. Todos os bits que são fixados em "1" em R_4 obedecem ao critério da consulta. Por outro lado, usando a codificação por igualdade, E_0, E_1, E_2, E_3 e E_4 devem passar pela operação lógica `OR`, resultando no acesso a 5 vetores, número bem superior ao único que é percorrido no caso da codificação por faixa.

De modo simplificado, a codificação por faixa requer no máximo um acesso a um vetor para avaliar consultas por faixas de valores. Enquanto isso, o método por igualdade requer em média acesso a `N/2`, em que `N` é o número total de vetores de bits. Embora haja benefícios neste sentido, não se observa redução considerável na quantidade de vetores armazenados: `N-1` são necessários para a codificação por faixa, pois o vetor que contém

apenas 1's pode ser dispensado, como R_9. Logo, supondo que o domínio de **X** é D = {0, ..., 9}, ou seja |**X**|=10 , cria-se os vetores $E_d \mid d \in D$ para a codificação por igualdade. Na codificação por faixa são criados $R_d \mid d \in D-\{9\}$.

RID	$\Pi_X(R)$	E_9	E_8	E_7	E_6	E_5	E_4	E_3	E_2	E_1	E_0	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	3	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1	1	0	0	0
2	2	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	0	0
3	1	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	0
4	2	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	0	0
5	8	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
6	**2**	0	0	0	0	0	0	0	**1**	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0
7	9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
9	7	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
10	5	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0
11	6	0	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
12	4	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0	0	0	0

(a) Projeção (b) Bitmap codificado por igualdade (c) Bitmap codificado por faixa

Figura 18 – Comparação entre Bitmaps codificados por igualdade e por faixa, dado o Índice de Projeção.

Diante disso, há necessidade de técnicas mais apuradas para controlar a quantidade de vetores do Índice Bitmap, a fim de tornar viável o seu emprego sobre atributos com alta cardinalidade. A codificação binária empregada pelos índices **Bitmap codificado** e **Bit-sliced**, que serão abordados a seguir, é a que produz a menor quantidade de vetores de bits: $\lceil \log_2 |\mathbf{X}| \rceil$. Esta codificação requer bem menos vetores que a codificação por faixa. Por outro lado, enquanto esta acessa apenas um vetor na resposta a uma consulta por faixa de valores, a codificação binária acessa todos eles. Conclui-se, então, que é preciso existir equilíbrio entre os requisitos tempo de resposta e espaço de armazenamento.

O'Neil, P. e Quass (1997) alcançaram melhor desempenho no processamento de consultas usando o Índice de Projeção e o Índice Bit-sliced. O Índice Bit-sliced emprega a codificação binária, sendo constituído por um conjunto de "fatias de bitmaps", as quais são "ortogonais" aos dados mantidos num Índice de Projeção. Em uma tabela T, cada valor da coluna C é mapeado para um número binário de N+1 bits. Define-se então a função D(n,0), com i = 0, ..., N, para a n-ésima tupla da tabela T da seguinte maneira:

D(n,0) = 1 se o bit de posição 2^0 for 1 no valor codificado da n-ésima tupla de C.

D(n,1) = 1 se o bit de posição 2^1 for 1 no valor codificado da n-ésima tupla de C.

E assim por diante. Ou seja:

D(n,i) = 1 se o bit de posição 2^i for 1 no valor codificado da n-ésima tupla de C.

Para cada valor i=0,...,N, e D(n,i) > 0 em uma tupla de T, o vetor B_i vai sendo definido pela atribuição de D(n,i) ao n-ésimo bit deste. A condição D(n,i) > 0 evita a representação de um vetor inteiro de zeros. A Figura 19b mostra a construção dos índices de

projeção e a Figura 19c do bit-sliced, ambos para a coluna Quantidade da tabela vendas (Figura 19a).

A primeira linha (índice 0) da coluna Quantidade tem valor 5, ou $(101)_2$. Logo, D(0,0) = 1, então $B_0[0]=1$; D(0,1) =0, então $B_1[0]=0$; e D(0,2)=1, então $B_2[0]=1$. Observa-se que índices Bitmap básicos para a coluna Quantidade exigiriam um total de vetores menor que o número necessário para bit-sliced (5 < 6). Este fato, porém, nem sempre é observado em domínios de dados reais.

Nome	Mês	Cidade	Quantidade
Tênis X	Jan	Analândia	5
Cama Y	Jan	Itirapina	3
Sofá Z	Jan	Ipeúna	0
Câmera W	Jan	São Carlos	60
Perfume U	Jan	Araraquara	57
Livro L	Jan	Descalvado	5

(a) Fragmento de dados da tabela Vendas

Quantidade
5
3
0
60
57
5

(b) Projeção

B_5	0	0	0	1	1	0
B_4	0	0	0	1	1	0
B_3	0	0	0	1	1	0
B_2	1	0	0	1	0	1
B_1	0	1	0	0	0	0
B_0	1	1	0	0	1	1
	5	*3*	*0*	*60*	*57*	*5*

(c) Bit-sliced

Figura 19 – Índices de projeção e bit-sliced exemplificados.

Uma consulta SELECT SUM (Quantidade) FROM Vendas WHERE *condição* possui diferentes processamentos conforme os índices apresentados, supondo a existência do vetor B_F para representar o conjunto de valores que atendem à cláusula WHERE. No caso de uma B-tree, existem vetores B_v em seus nodos folha para os valores não-nulos v. Para cada resultado de (B_v AND B_F) são contadas as ocorrências do bit "1", e então o total desta contagem é multiplicado por v. Este produto é adicionando à soma que se obtivera para cada B_v antes do atual.

Se B_F é um vetor, pode ser alocado em memória, tornando mais ágil o processamento da consulta. Isto não é observado se B_F for uma lista-RID, ou seja, uma lista de identificadores de tuplas (*row identifier*) que especificam a posição da tupla em disco, pois um identificador tem tamanho superior a um bit. Freqüentemente, cada B_v é implementado como lista-RID, e a iteração sobre tais listas é uma tarefa custosa, como discutem O'Neil, P. e Quass (1997).

O cálculo da referida soma com um Índice Bit-sliced sobre um atributo contendo os vetores B_i (i = 0, ..., N, com N = 5) acontece inicializando-a com zero e adicionando à mesma, para i=0 até i=N, 2^i multiplicado pela contagem dos bits "1" presentes no vetor resultante de (B_i AND B_F). Considerando que o domínio do atributo é vasto, exigindo muitos vetores para um Índice Bitmap básico, e que pode ser mapeado em números binários que requerem poucos vetores na abordagem Bit-sliced, então Bit-sliced executa menos contagens e operações AND que a abordagem básica.

Os autores também discutem a combinação adequada de função de agregação (além de SUM) e índice. Eles analisam ainda o desempenho do processamento de consultas por

faixas de valores, propondo um algoritmo que realiza este tipo de consulta em Bit-sliced. Concluiu-se que as faixas de valores pouco abrangentes favorecem o uso de um índice como a B-tree. Aumentando-se a abrangência, Bit-sliced se torna a melhor opção.

A existência dos índices de Projeção ou Bit-sliced sobre todos os atributos das dimensões envolvidas em uma consulta OLAP dispensa a necessidade de uma junção explícita, e os faz atuar como índices de junção neste caso. Para computar *group-bys*, por exemplo, assume-se que existem Índices de Projeção na tabela de fatos para cada uma das colunas do *group-by* e para aquelas envolvidas com a função de agregação. Então, cada tupla do resultado da cláusula WHERE é classificada em uma célula de *group-by* por meio da leitura do Índice de Projeção apropriado, e os valores a serem agregados (que também estão num Índice de Projeção) são lidos. Efetua-se a função de agregação sobre os valores, e seu resultado é por fim armazenado na célula adequada.

Diante das novas estruturas e algoritmos que propuseram, O'Neil, P. e Quass (1997) argumentam que um DW deve possuir mais de um índice disponível sobre cada coluna, a fim de viabilizar a escolha do mais adequado de acordo com a consulta a ser respondida. Contudo, não trataram questões concernentes à atualização das estruturas que apresentaram.

Objetivando reduzir o espaço necessário para armazenar um Índice Bitmap e ainda alcançar bom desempenho no processamento de consultas, Wu, M. e Buchmann (1998) propuseram uma extensão para este índice. A idéia central é a de codificar em binário o domínio do atributo indexado. A Figura 20, adaptada do referido trabalho, exibe as construções dos índices Bitmap básico e codificado para o domínio {*a*, *b*, *c*} do atributo **X**.

Os componentes do Índice Bitmap com codificação binária são: o conjunto de vetores, uma tabela de mapeamento para o domínio da coluna indexada, e as funções *booleanas* (detalhadas em seguida). Observa-se que o valor *a* é codificado como 00, *b* como 01, e *c* como 10. O vetor $\mathbf{B_i}$ contém o i-ésimo bit do valor codificado do atributo **X**, partindo do menos significativo para o mais significativo. Ou seja, $\mathbf{B_1}[0]\mathbf{B_2}[0] = 00 = a$ conforme determina a tabela de mapeamento, mantendo a relação de ordem total (ROT) do domínio.

Tabela **T**

...	**X**	...
	a	
	b	
	c	
	b	
	a	

Índice Bitmap básico

$\mathbf{B_a}$	$\mathbf{B_b}$	$\mathbf{B_c}$
1	0	0
0	1	0
0	0	1
0	1	0
1	0	0

Índice bitmap codificado

$\mathbf{B_1}$	$\mathbf{B_0}$
0	0
0	1
1	0
0	1
0	0

Tabela de mapeamento

a	00
b	01
c	10

Figura 20 – Exemplo do Índice Bitmap com codificação binária.

Visto que a cardinalidade do atributo **X** é 3, são requeridos três vetores de bits na abordagem simples. Já na abordagem codificada, são $\lceil \log_2 |\mathbf{X}| \rceil = \lceil \log_2 3 \rceil = 2$ vetores, mais a tabela de mapeamento. Se um valor v_0 é codificado como b_1b_0 ($b_i \in \{0,1\}$, i = 0,1), então a **função de recuperação** para v_0 é definida como x_1x_0, em que $x_i = \mathbf{B_i}$ se b_i=1. Caso contrário, x_i é a negação de $\mathbf{B_i}$, ou seja, $\mathbf{B_i'}$. No exemplo, as funções de recuperação para *a*, *b* e *c* são f_a=$\mathbf{B_1'B_0'}$, f_b=$\mathbf{B_1'B_0}$, f_c=$\mathbf{B_1'B_0}$, respectivamente.

Considerando que *x*' denota a negação da variável *x*, *xy* denota (*x* AND *y*), e *x*+*y* denota (*x* OR *y*), se a consulta procura por dados em que **X**=*a* ou **X**=*b*, então é possível aplicar o operador OR sobre f_a e f_b, isto é: $f_a + f_b = \mathbf{B_1'B_0'} + \mathbf{B_1'B_0}$, cuja redução é $\mathbf{B_1'}$. Logo, para recuperar tuplas em que **X**=*a* ou **X**=*b*, basta usar o inverso do vetor $\mathbf{B_1}$ e recuperar as tuplas com 1's, que satisfazem a condição de seleção.

Além da vantagem relacionada ao menor espaço de armazenamento requerido, as consultas podem se tornar mais ágeis pelo acesso a alguns (e não todos) os vetores. A tabela de mapeamento não precisa ser acessada nas consultas, pois a codificação determina a construção de cada vetor preservando a ordem em que as tuplas ocorrem na tabela. Isto é, $\mathbf{B_1}[0]\mathbf{B_2}[0] = 00 = a$ ocorre na primeira posição da tabela, bem como nas primeiras posições dos vetores $\mathbf{B_1}$ e $\mathbf{B_2}$. Assim, não constitui custo adicional ao processamento das consultas.

Para minimizar o número de vetores que devem ser acessados, e assim reduzir o tempo de processamento, é necessário criar uma **codificação bem definida**. Descobri-la sem um algoritmo tem complexidade exponencial na cardinalidade do domínio. Por isso, embora desejável, ela não é essencial. Wu, M. e Buchmann (1998) não apresentam nenhum algoritmo para sua descoberta, mas explicam como o Índice Bitmap codificado pode ser aplicado a hierarquias de atributos nas dimensões, na preservação da relação de ordem total (ROT) de um atributo e a atributos com faixas de valores. Tais propriedades reafirmam que o Índice Bitmap codificado é adequado para OLAP.

Com relação ao processamento das consultas, Wu, M. e Buchmann (1998) observaram que a versão codificada é mais bem sucedida, em especial sob consultas envolvendo faixas extensas de valores. Mesmo que a codificação seja realizada de maneira imprópria, ou que a redução lógica das funções de recuperação não seja satisfatória (pior caso), o Índice Bitmap codificado é mais estável e tem desempenho superior ao do Índice Bitmap básico.

Quando o critério de seleção envolve uma seleção de valor **X**=a, por exemplo, o método básico acessa um vetor, enquanto que o codificado acessa dois. Por outro lado, na

consulta por faixa de valores **X** in {a,b} um vetor é acessado pelo codificado, e dois pelo básico. Se b_b e b_c denotam as quantidades de vetores acessados pelos índices Bitmap básico e codificado respectivamente, então $1 \leq b_b \leq |\mathbf{X}|$ e $1 \leq b_c \leq \lceil \log_2 |\mathbf{X}| \rceil$. Na abordagem básica, $b_b = \delta$, em que δ denota o tamanho do intervalo da consulta (δ=2 em **X** in {a,b}). Na abordagem codificada, b_c é uma função de δ, com $1 \leq \delta \leq |\mathbf{X}|$. No pior caso, $b_c \leq \lceil \log_2 |\mathbf{X}| \rceil$. No melhor, o número de vetores descartados pela redução lógica (γ) influencia no resultado, levando a $b_c = \lceil \log_2 |\mathbf{X}| \rceil - \gamma$. Para maiores detalhes, sugere-se a leitura de Wu, M. e Buchmann (1998).

A implementação dos vetores de bits nos nodos folha de B-trees demonstrou a degradação do desempenho, à medida que a cardinalidade se tornou muito alta. Sob esta circunstância, tal implementação teve desempenho parecido com o da própria B-tree. Wu, M. e Buchmann (1998) não propuseram nenhuma solução neste sentido, apesar de terem levantado outras questões pendentes. Dentre elas, classificaram como desejáveis:

a) o uso de valores "irrelevante" na codificação, para aprimorar o desempenho; e
b) a existência de um modelo para avaliar a viabilidade da reconstrução dos índices em domínios de aplicação cujo conjunto de predicados de seleção pré-definidos se altera com o passar do tempo.

As atualizações sobre o DW podem introduzir novos valores ao domínio de um atributo **X**, aumentando a sua cardinalidade, interferindo na questão (b) mencionada. Se isto ocorrer, e o número de bits usados para codificar o domínio for suficiente para manter o novo elemento, ele é codificado e inserido na tabela de mapeamento. Por exemplo, a inclusão de *d* no domínio do atributo **X** da Figura 20, e *d* sendo codificado como 11. Se o número de bits for insuficiente, um novo vetor de bits é criado e a tabela de mapeamento é atualizada. Por exemplo, a inclusão de *e* subseqüente à inclusão de *d* no domínio do atributo **X** resulta na criação de um novo vetor de bits, e nos mapeamentos: *a* = 000, *b* = 001, *c* = 010, *d* = 011, *e* = 100. A Figura 21 ilustra esta situação.

Tabela T

...	**X**	...
	a	
	b	
	c	
	b	
	a	
	d	
	e	

Índice Bitmap Básico

$\mathbf{B_a}$	$\mathbf{B_b}$	$\mathbf{B_c}$	$\mathbf{B_d}$	$\mathbf{B_e}$
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	1	0	0	0
1	0	0	0	0
0	0	0	1	0
0	0	0	0	1

Índice Bitmap Codificado

$\mathbf{B_2}$	$\mathbf{B_1}$	$\mathbf{B_0}$
0	0	0
0	0	1
0	1	0
0	0	1
0	0	0
0	1	1
1	0	0

Tabela de Mapeamento

a	000
b	001
c	010
d	011
e	100

Figura 21 – Atualização do Índice Bitmap com codificação binária.

A análise dos resultados de Wu, M. e Buchmann (1998) apontou que, em média, a razão (m-1)/m do índice Bitmap básico é esparso, sendo m a cardinalidade do atributo indexado. Para o modo codificado, a razão é ½ (independente de m). Por isso, a sua construção e a sua manutenção são mais econômicas que as da versão básica, à medida que a cardinalidade aumenta.

3.2.4 Combinando as técnicas para melhorar o desempenho

As técnicas de *binning*, compressão e codificação apresentam vantagens e desvantagens, conforme discutido nas seções anteriores. Todas devem ser levadas em consideração no projeto do Índice Bitmap, segundo os objetivos que se deseja alcançar. A combinação destas técnicas visando melhorar o desempenho também é chamada de ajuste fino (*tunning*). Nesta seção, são comentados alguns casos específicos e as soluções viáveis para os mesmos, baseando-se na análise feita por Stockinger e Wu, K. (2007).

Se não for aplicada a codificação binária sobre o Índice Bitmap, deve-se comprimi-lo a fim de beneficiar o armazenamento. A codificação e a quantidade de caixas do *binning* merecem atenção. A melhor técnica de codificação depende intimamente do tipo de consulta que se deseja responder. Por exemplo, a codificação por faixa é a melhor para consultas por faixas de valores, mas pode ter seu desempenho degenerado em atributos de alta cardinalidade.

Quando não há compressão, a codificação por faixa é favorecida. Quando há, o custo do percurso no índice é proporcional ao número de acertos. Isto independe da quantidade de vetores de bits envolvidos em uma estrutura comprimida por WAH e codificada por igualdade. Como a codificação por igualdade tem a sua compressão mais facilitada, pode-se optar por esta compressão.

A escolha da quantidade de caixas do *binning* visa diminuir os candidatos a se submeterem ao refinamento. Logo, quanto mais caixas, menor o custo associado à checagem de candidatos na fase de refinamento. Porém, na codificação por igualdade, o uso de mais caixas aumenta também o custo do percurso no índice. No método por faixa isto não acontece, porque não mais que dois vetores são acessados para responder à consulta.

Outro fator que demanda atenção é a estratégia de *binning*, que tem influência direta sobre a checagem de candidatos. A escolha fica entre criar partições do domínio em caixas de mesmo tamanho ou distribuir igualmente uma quantidade de entradas entre as caixas (STOCKINGER; WU, K. 2007).

3.3 FastBit

Stockinger e Wu, K. (2007) contestam a afirmação de Sarawagi (1997) de que o Índice Bitmap possa ser implementado como uma B-tree cujos nodos folha referenciam vetores de bits. Eles argumentam que que a B-tree é empregada apenas no arranjo de chaves, sendo os vetores de bits armazenados separadamente em arquivos, isto é, independentemente daquela árvore.

O software livre FastBit (O'NEIL, E.; O'NEIL, P.; WU, K, 2007; FASTBIT, 2008) é um exemplo que realça esta independência, pois os vetores de bits são organizados em um arquivo binário. Esta implementação confere uma boa eficiência ao Índices Bitmap construído por FastBit. Em maiores detalhes, são construídos índices de Projeção, sendo um arquivo de índice para cada coluna. Cada arquivo mantém um vetor de valores distintos em ordem crescente, cujas entradas apontam para vetores de bits. Na Figura 22, este vetor é representado como ***F***. Nesta ilustração, foi construído um IBJ usando FastBit, para o atributo *fornecedor_pk* da Figura 12. É importante salientar que este software implementa diversificadas técnicas de *binning*, compressão e codificação. Por isso, constitui uma alternativa extremamente viável para a construção e a manipulação de índices Bitmap. Contudo, FastBit não provê suporte à indexação de DWG.

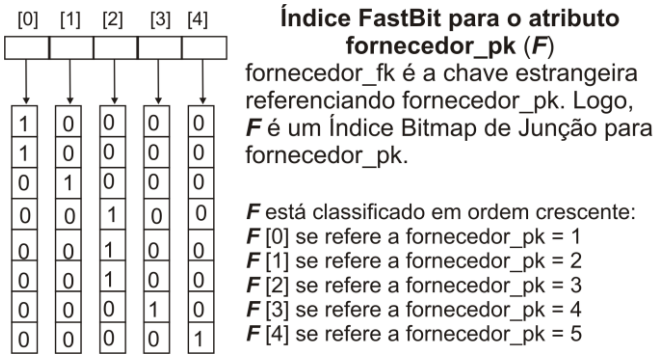


Figura 22 – Um Índice Bitmap de Junção criado usando FastBit.

3.4 aR-tree

A aR-tree (*aggregate* R-tree) (PAPADIAS et al., 2001) é um índice para DWG baseado na R-tree que mantém em suas entradas os valores gerados pela função de agregação sobre uma medida da tabela de fatos. Antes de prosseguir com a sua descrição, é plausível contextualizá-la. Em aplicações OLAP, é possível definir hierarquias de relacionamento de atributos em diversas dimensões. No exemplo de varejo da Seção 2.1., ano $\prec$ mês é uma hierarquia, em que mês constitui o nível de menor granularidade. O usuário pode também submeter consultas com agrupamento por ano. Para tornar ágil a resposta a consultas que envolvem agregação, os resultados das mesmas podem ser pré-calculados e materializados.

São diversas as técnicas que promovem a pré-agregação de medidas em DW convencionais, com o intuito de acelerar o processamento de consultas. Diferentemente do que ocorre em um DW convencional, em um DWG pode haver pouco ou nenhum conhecimento prévio sobre a hierarquia de relacionamento de atributos espaciais, a qual permite o agrupamento dos objetos. As hierarquias de relacionamento de atributos não pré-estabelecidas durante o projeto são chamadas na literatura de hierarquias *ad-hoc*. Ainda, as consultas SOLAP podem requerer agrupamentos baseados em mapas criados arbitrariamente, ao invés de apenas solicitar as regiões geográficas predefinidas nas tabelas de dimensão. Conseqüentemente, os métodos de agregação prévia, como a materialização de visões da treliça, são inadequados para o caso espacial. Isto motivou a proposta de um método que combinasse indexação espacial com resultados pré-agregados.

Seja um sistema de supervisão de tráfego que monitora as posições dos veículos numa cidade e o tráfego nas ruas. Nele, os veículos podem estar também em estacionamentos e não apenas nas ruas. Para ele, existem duas R-tree's: uma para armazenar as posições dos carros (R_C), e outra para os segmentos de rua (R_R). Neste sistema, consultas típicas são:

a) "encontre os segmentos das ruas com tráfego mais intenso, e próximos ao centro da cidade";
b) "indique o hospital que pode ser alcançado mais rapidamente, dada a situação atual do trânsito";
c) "encontre o número total de carros dentro de cada bairro";
d) "encontre o segmento de rua com o maior tráfego em um raio de 1 km em torno de cada hospital".

Diante destas suposições, responder à consulta "forneça o tráfego para todo segmento de rua numa área de raio 1 km em torno de cada hospital", requer uma junção espacial entre R_C e R_R. Embora o resultado desta junção possa estar pré-computado e armazenado para cada um dos segmentos, a condição espacial "área de raio 1 km em torno de cada hospital" remete a um MAM. Isto também viabiliza a construção de um DWG por meio de um esquema estrela, sob as diretrizes expostas por Stefanovic et al. (1997; 2000).

3.4.1 Estrutura de dados

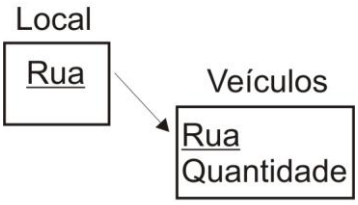
Papadias et al. (2001) propuseram um índice sobre os objetos de menor granularidade da dimensão espacial, o qual consiste em uma R-tree que armazena, para cada MBR, o valor da função de agregação para todos os objetos nele incluídos. Por isso, a

estrutura foi denominada aR-tree. Uma vez que os agrupamentos de objetos espaciais realizados pelo índice criam uma hierarquia implícita, tal hierarquia foi incorporada à treliça de de Harinarayan, Rajaraman e Ullman (1996). Isto possibilita a seleção das agregações apropriadas para a materialização.

Com isso, baseado no nível de menor granularidade espacial de uma tabela de dimensão espacial, a aR-tree pode responder a uma consulta com agregação sem requerer o acesso a todos os objetos incluídos no MBR. Isto porque parte da resposta pode ser encontrada nos nodos intermediários da árvore. A aR-tree é, portanto, uma estrutura de indexação para DWG sob ROLAP. Ela pode lidar com dimensões espaciais, medidas espaciais, ou ambas simultaneamente.

O DWG do exemplo para a supervisão de tráfego da Seção 3.4 é representado por um esquema estrela cuja tabela de fatos deve conter os resultados agregados para a dimensão espacial no menor nível de granularidade. Este nível define a quantidade de carros por segmento de rua. A Figura 23 exibe um DWG com apenas uma tabela de dimensão espacial, para simplificar a compreensão da aR-tree. As ruas r_1, ..., r_5 são aproximadas para seus MBR a_1, ..., a_5.

Neste exemplo, supondo que carros (pontos) e ruas (linhas) são indexados, respectivamente, pelas árvores-R R_C e R_R, o algoritmo de junção espacial de Brinkhoff, Kriegel e Seeger (1993) pode ser aplicado a fim de calcular os valores da função de agregação para os nodos folha da aR-tree. Caso contrário, outros índices de junção específicos podem ser empregados. Desta forma, a árvore pode ser construída de baixo para cima, com o reuso das informações dos níveis mais baixos. Fica delineada a hierarquia *ad-hoc* $A_i \prec a_i$.

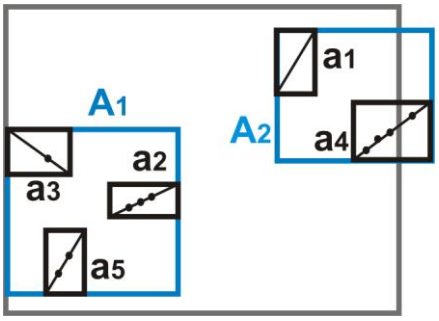


(a) Esquema estrela

Veículos

Rua	Quantidade
r_1	0
r_2	3
r_3	1
r_4	4
r_5	2

(b) Dados na tabela de fatos



(c) aR-tree para a dimensão espacial Local

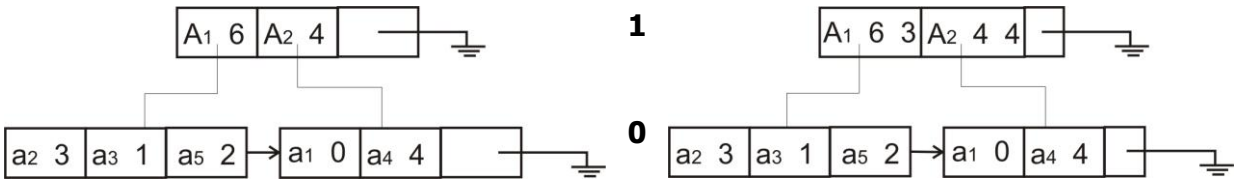
Figura 23 – Um DWG com a dimensão espacial Local e a aR-tree correspondente.

Na estrutura de dados, cada nodo X_i da aR-tree possui as entradas $X_{i,1}$ ... $X_{i,CPi}$ e um ponteiro para o próximo nodo X_{i+1} no mesmo nível, sendo que $CP_i \leq CP$ é a capacidade de X_i. Em cada entrada da forma $< X_{i,j}.MBR, X_{i,j}.ref, X_{i,j}.agr >$, o ponteiro $X_{i,k}.ref$ referencia o nodo X_k

do próximo nível inferior. Logo, a raiz constitui o maior nível da árvore. O resultado pré-agregado para cada entrada é $X_{i,k}$.agr, mantendo todos os valores para cada resultado de função de agregação (COUNT, MAX, AVG etc). Cada objeto no nível n-1 pertence a exatamente um MBR do nível n, permitindo o armazenamento de resultados parciais e a agregação deles num passo posterior. Com isto, são obtidos resultados mais resumidos, ou seja, efetua-se uma operação de *roll-up*.

Seguem, na Figura 24, duas possíveis aR-tree para o exemplo de tráfego. Na Figura 24a, o nodo raiz da aR-tree, X_1, está no nível 1 e mantém as entradas $X_{1,1}$ e $X_{1,2}$, além de um ponteiro nulo (porque não há outro nodo no mesmo nível). Para a entrada $X_{1,1}$ são válidos: $X_{1,1}$.MBR = A_1; $X_{1,1}$.ref consiste na aresta que liga a entrada $X_{1,1}$ ao nodo X_2=[(a_2, 3), (a_3, 1), (a_5, 2)] do nível 0; e $X_{1,1}$.agr=6. O nodo X_2 mantém as entradas $X_{2,1}$, $X_{2,2}$ e $X_{2,3}$, além de um ponteiro para seu nodo irmão X_3, representado pela aresta dirigida que parte de X_2. Como existem 3 carros na rua r_2, há uma entrada (a_2, 3) em um nodo folha da aR-tree. O comentado *roll-up* define que o total de carros em todas as ruas seja calculado pela soma dos resultados materializados para A_1 e A_2, ou seja, 6 + 4 = 10. Isto é possível porque cada a_i (i = 1, ..., 5) pertence a exatamente um A_i, ou seja, há um relacionamento N:1 entre MBR do nível 0 com aqueles do nível 1.

A Figura 24b exemplifica uma aR-tree cujo DWG requer duas funções de agregação: COUNT e MAX. Ou seja, há interesse em armazenar contagem e o número máximo de veículos para cada grupo de segmentos de rua. Portanto, na entrada $X_{1,1}$ valem: $X_{1,1}$.MBR = A_1; $X_{1,1}$.ref consiste na aresta que liga a entrada $X_{1,1}$ ao nodo X_2=[(a_2, 3), (a_3, 1), (a_5, 2)] do nível 0; $X_{1,1}$.agr[0]=6; $X_{1,1}$.agr[1]=3. Ou seja, implementa-se uma lista com os valores das diferentes funções de agregação. Observe-se que $X_{1,1}$ é uma entrada do nodo X_1 e este último está no nível 1. Portanto, $X_{1,1}$.agr[1]=3 significa que o número máximo de carros armazenado em ruas mantidas pelo nodo X_2 é 3, sendo que este é referenciado por $X_{1,1}$.ref e está no nível 0.



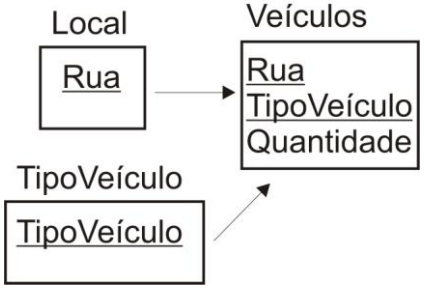
(a) aR-Tree com função de agregação COUNT (b) aR-Tree com funções de agregação COUNT e MAX

Figura 24 – aR-Tree com diferentes funções de agregação.

Quando houver mais de uma tabela de dimensão no esquema estrela, deve-se manter em cada entrada da árvore um ponteiro para um vetor multidimensional. Este vetor indica os valores agregados para todo o domínio das outras dimensões. Por exemplo, seja o esquema estrela da Figura 25a, e um fragmento dos dados de sua tabela de fatos na Figura

25b. Esta tabela de fatos possui valores iguais a zero para a medida em algumas de suas tuplas. Isto não é uma restrição, porque estas tuplas poderiam simplesmente inexistir. Papadias et al. (2001) nada comentam sobre isto. A dimensão TipoVeículo é não-espacial.

A aR-tree correspondente ao fragmento exibido pela Figura 25 é mostrada na Figura 26. Presume-se que a ordem dos dados mantidos na tabela de dimensão TipoVeículo (Figura 25c) é respeitada na disposição dos dados em cada vetor unidimensional que acompanha cada entrada dos nodos da árvore. Esta suposição é feita porque Papadias et al. (2001) apenas mencionam que deve haver um vetor multidimensional para os valores das demais dimensões, mas não explicam sua implementação. No exemplo, a entrada que mantém o MBR A_1 tem o vetor {2, 1, 3}. Isto é, no interior de A_1 há 2 caminhões, 1 carro e 3 motos.



(a) Esquema estrela

Veículos

Rua	TipoVeículo	Quantidade
r_1	Moto	0
r_1	Carro	0
r_1	Caminhão	0
r_3	Moto	1
r_2	Caminhão	2
r_2	Carro	1
r_4	Carro	2
r_4	Moto	2
r_5	Carro	2

(b) Dados na tabela de fatos

TipoVeículo

TipoVeículo
Caminhão
Carro
Moto

(c) Tipos de veículos

Figura 25 – aR-Tree com uma dimensão espacial (Local) e uma não-espacial (Veículos).

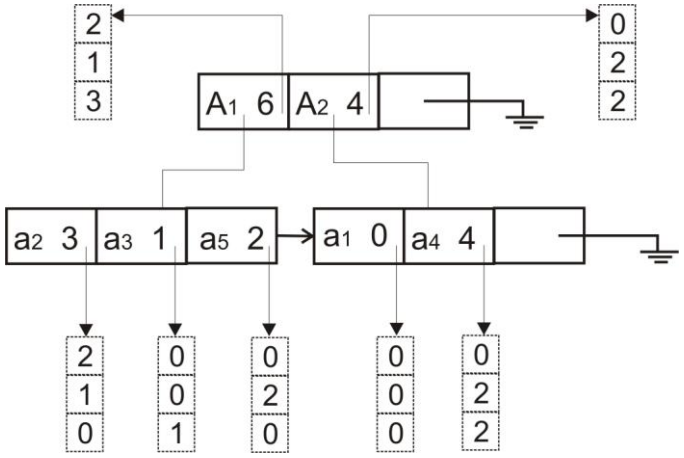


Figura 26 – aR-Tree com diferentes funções de agregação.

Duas considerações devem ser feitas aqui. Primeiramente, a construção de agrupamentos com hierarquias *ad-hoc* no índice, baseada no atributo espacial de menor granularidade, remete ao modelo de DWG de Stefanovic et al. (1997; 2000). Mais especificamente, quer dizer que uma dimensão estritamente espacial pode ser indexada apenas no atributo com menor granularidade. Por exemplo, se existir uma hierarquia de relacionamento de atributos espaciais estado $\prec$ cidade $\prec$ rua, apenas rua pode ser indexado.

A segunda consideração é a de que Papadias et al. (2001) afirmam que a estrutura deste índice pode ser estendida no sentido de permitir medidas espaciais: deve-se substituir $X_{i,j}$.agr por um ponteiro que referencie o objeto agregado. Ainda, tanto medidas numéricas quanto espaciais podem coexistir na mesma árvore. Para viabilizar tais estruturações, o hipercubo deve ser aquele descrito por Stefanovic et al. (1997; 2000).

Retomando a descrição da aR-tree, é importante frisar que ela também define uma hierarquia entre MBR que formam uma treliça, em que E_l é o conjunto de valores materializados para todas as entradas dos nodos do nível *l* da árvore. Se E_i pode ser respondido por E_j, então $E_i \preceq E_j$. Se a altura da árvore é h, então E_h contém apenas um membro: o resultado agregado de todos os objetos. A existência da treliça é assegurada porque o operador $\preceq$ impõe ordem aos conjuntos E_l, e há um elemento cabeça E_0 do qual dependem todos os conjuntos E_l. Na Figura 27 exibe-se a treliça para a árvore da Figura 24a, os conjuntos E_0, E_1, E_2 e as agregações (AG) que os originam. Observa-se que $E_1 = E_{h-1}$ é o conjunto do nodo raiz da árvore, e que cada E_l corresponde a um nível da árvore.

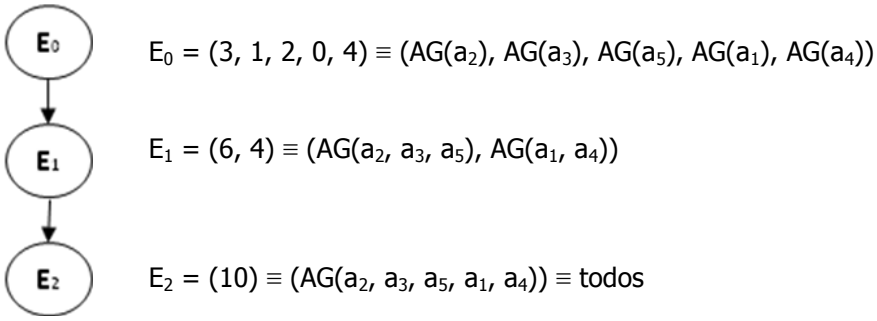
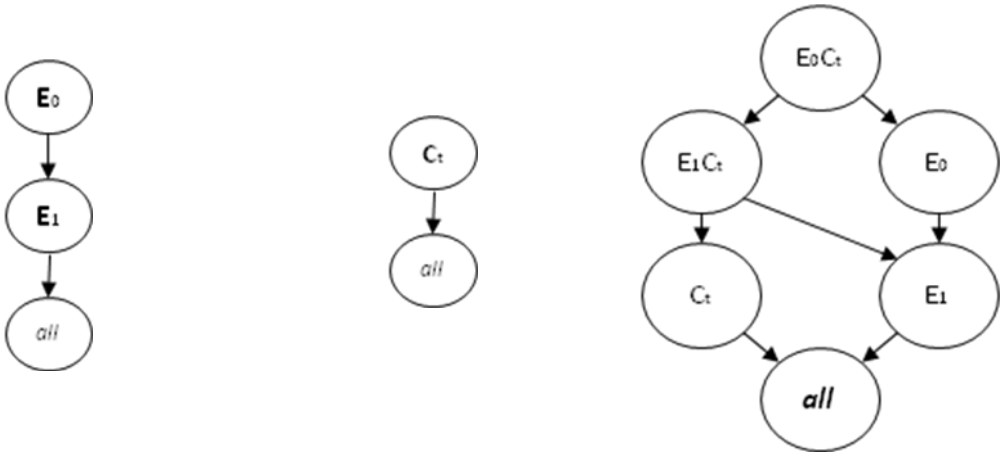


Figura 27 – A treliça para a aR-tree da Figura 24a (PAPADIAS et al., 2001).

Uma vez incorporada à treliça, a aR-tree desfruta de todas as vantagens conferidas aos DW não-espaciais. Uma delas é o suporte a múltiplas dimensões, cuja implementação foi previamente comentada. Supondo a existência de uma dimensão não-espacial todos (*all*)$\prec C_t$ que modela tipos de veículos (vide Figura 25a), a treliça resultante para as dimensões E_0 e C_t é exibido na Figura 28. O nodo E_0C_t corresponde à consulta que retorna o número de veículos em cada rua agrupado por seu tipo, por exemplo. Se houver no esquema estrela mais que uma dimensão espacial, então uma aR-tree distinta deve ser criada para cada uma delas.

Outra vantagem diz respeito à materialização de visões, que consiste na seleção e materialização de conjuntos E_l a fim de acelerar o processamento das consultas. Esta é uma boa iniciativa quando, além de hierarquias *ad-hoc*, existem hierarquias definidas em tempo de projeto (bairro $\prec$rua, por exemplo). Usa-se então a combinação destas hierarquias e alguns

níveis da árvore, para alcançar baixo custo no processamento das consultas. Isto é realizado criando-se uma visão da árvore por meio de redefinições de ponteiros, interligando alguns de seus níveis e dispensando outros intermediários. A construção de uma visão materializada não modifica o índice, pois cria uma estrutura à parte.



(a) Dimensão espacial (b) Dimensão não-espacial (c) Treliça resultante

Figura 28 – Treliça composta de uma dimensão espacial e outra não-espacial (PAPADIAS et al., 2001).

Antes de descrever o processamento das consultas, é válido comentar dois fatos sobre a aR-tree. Primeiramente, para hierarquias conhecidas em tempo de projeto, a materialização é preferível à indexação usando a aR-tree. Logo, o processamento de consultas que envolvam uma dimensão espacial "Local" com a hierarquia de relacionamento de atributos região $\prec$ nação $\prec$ cidade $\prec$ endereço pode ser feito com maior agilidade usando visões materializadas do que usando a indexação com a aR-tree (PAPADIAS et al., 2001). Contudo, as dimensões com hierarquias *ad-hoc* (sistema de supervisão de tráfego exemplificado) ainda carecem deste índice para obter um bom desempenho no processamento de consultas.

O segundo fato se refere à forma de divisão do espaço indexado. Os autores afirmam que, além do método de agrupamento de objetos proposto pela R-tree, outros podem ser utilizados. Por exemplo, o modo proposto pela Quadtree (GAEDE; GÜNTHER, 1998). É importante ressaltar que o modo de agrupamento influencia o estabelecimento de hierarquias *ad-hoc* e a construção da árvore.

3.4.2 Processamento de consultas

O processamento das consultas usando aR-tree é realizado por meio do percurso da árvore a partir de seu nodo raiz, seguindo em direção aos nodos folha. Todas as

entradas dos nodos visitados são testadas em relação à janela de consulta (JC), e uma das seguintes condições ocorre:

a) se o MBR da entrada e a JC são disjuntas (não há interseção), então a entrada não é recuperada;
b) se o MBR da entrada está contido na JC, recupera-se a informação agregada da entrada, evitando acesso ao nodo filho; e
c) se o MBR da entrada sobrepõe parcialmente a JC, o nodo filho desta entrada deve ser percorrido. Se for um nodo folha, então a resposta é estimada ou computada acessando os dados originais.

Há uma diferença fundamental entre este algoritmo de busca e o respectivo algoritmo da R-tree. Na R-tree, se a JC for muito grande, emprega-se a busca seqüencial sobre os objetos ao invés do índice. Já na aR-tree, para consultas que promovem agregação, existem dois casos:

a) a JC é muito grande. Logo, muitos nodos intermediários estarão nela contidos. Então os resultados pré-calculados são usados, evitando-se visitar os objetos individualmente; e
b) a JC é muito pequena. Então, a aR-Tree age como um MAM, permitindo a seleção dos objetos candidatos.

Pode ocorrer o caso em que existem múltiplas janelas de consulta q_i, em que cada uma define uma região de agrupamento sobre a qual se aplica a função de agregação, compondo o conjunto Q. Um exemplo de consulta assim é: “para cada JC, encontre o número total de carros em seu interior localizados em segmentos de ruas”. Nesta situação, percorre-se recursivamente a árvore, seguindo somente pelas entradas que intersectam parcialmente alguma janela, calculando os valores das funções de agregação durante o caminho.

Se o nível de granularidade da consulta for mais baixo, como o de “para cada segmento de rua no interior da JC, calcule o número de carros”, então os nodos no interior da JC também são percorridos no sentido das folhas, a fim de encontrar os MBR dos objetos espaciais candidatos. Isto é válido tanto para uma JC quanto para um conjunto delas. Deste modo, a aR-tree age como um MAM sobre a tabela de fatos.

Por fim, relata-se que a aR-tree pode ser empregada ainda sob um DWG distribuído, conforme expõem Gorawski e Chechelski (2005). Eles apresentam uma técnica de balanceamento para este ambiente, visto que nele o balanceamento em computadores com capacidades distintas é o maior problema. Sua abordagem define um algoritmo de balanceamento para um DWG distribuído em uma arquitetura *shared nothing*, fixando o

tamanho do conjunto de dados armazenado em cada nodo. Este trabalho não será discutido, embora cite e aplique a aR-tree.

3.5 a3DR-tree e aRB-tree

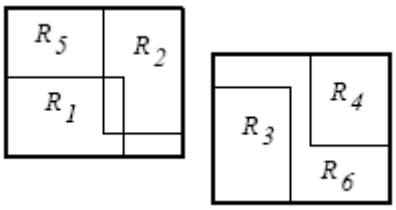
Papadias et al. (2002), motivados pelo emergente armazenamento de grandes volumes de dados multidimensionais envolvendo tempo em aplicações diversas, como supervisão de tráfego e telefonia celular, propuseram estruturas de indexação espaço-temporais para DW. Sua abordagem modela as dimensões espacial e temporal em uma mesma dimensão combinada no hipercubo, integrando indexação espaço-temporal com pré-agregação de medidas sob a implementação de múltiplas árvores.

O ambiente é caracterizado por falta de hierarquias predefinidas, e o enorme volume de dados requer estratégias de pré-agregação de algumas visões. As agregações podem permanecer constantes por longos intervalos, exigindo menos espaço para armazenamento. Por exemplo, embora o número exato de carros em trânsito em uma área da cidade possa ser alto, as aproximações "tráfego intenso", "tráfego médio" e "pouco tráfego" agregam a quantidade de veículos de forma que ela permaneça constante por certo tempo.

São duas as vertentes analisadas em seu trabalho. Uma vertente considera que as dimensões espaciais são dinâmicas e evoluem no tempo. Por exemplo, a área de uma célula de cobertura de rádio, que exposta a condições climáticas pode ter seu raio alterado. A outra vertente analisa dimensões espaciais estáticas, tal como a análise do tráfego nas ruas de uma cidade. Esta segunda abordagem é a que será tratada, porque este trabalho não contempla o estudo de objetos espaciais cujas geometrias mudam ao longo do tempo.

A primeira proposta é uma generalização da aR-tree para o espaço tridimensional, denominada a3DR-tree (*aggregate 3DR-tree*). Cada entrada *r* tem a forma <*r*.MBR, *r*.pointer, *r*.lifespan, *r*.aggr[]>, e portanto mantém o MBR do objeto espacial indexado, um ponteiro para um nodo de nível inferior (ou para o endereço efetivo da tupla), o intervalo durante o qual o valor da medida é válido e a medida agregada. Sempre que a medida se altera, uma nova entrada é criada. A Figura 29c exibe as entradas requeridas para uma região R_1 de acordo com os valores agregados (Figura 29b), supondo a R-tree da Figura 29a.

Desta maneira, a 3DR-tree possibilita a integração das dimensões espacial e temporal na mesma dimensão. Contudo, desperdiça espaço porque armazena o MBR a cada vez que há mudança na medida agregada. Por exemplo, o MBR de R_1 é armazenado 4 vezes . Além disso, sua estrutura é grande em tamanho, porém seus nodos têm pequeno *fanout*, comprometendo a eficiência do processamento de consultas.



(a) Exemplo de R-tree

Região	Tempo	Quantidade
R_1	1	150
R_1	2	150
R_1	3	145
R_1	4	135
R_1	5	130

(b) Fragmento da tabela de fatos

<R_1.MBR, R_1.ptr, [1, 1], 150>
<R_1.MBR, R_1.ptr, [1, 2], 300>
<R_1.MBR, R_1.ptr, [1, 3], 445>
<R_1.MBR, R_1.ptr, [1, 4], 580>

(c) Entradas para R_1

Figura 29 – Exemplo de a3DR-tree (PAPADIAS et al., 2002).

A segunda proposta, a aRB-tree (*aggregate* RB-tree) visa armazenar as regiões que constituem a hierarquia espacial apenas uma vez, indexando-as por uma R-tree. As entradas da R-tree têm um ponteiro para uma B-tree, a qual armazena os valores históricos agregados referentes à entrada.

Cada entrada *r* da R-tree tem a forma <*r*.MBR, *r*.pointer, *r*.btree, *r*.aggr[]>, em que MBR e pointer têm seus significados usuais, *r*.btree é o ponteiro para a B-tree correspondente e *r*.aggr[] mantém dados resumidos sobre *r* acumulados por todos os intervalos. Cada entrada *b* da B-tree tem a configuração <*b*.time, *b*.pointer, *b*.aggr[]>, sendo que *b*.aggr[] constitui os dados agregados no tempo *b*.time. Se o valor de *b*.aggr[] não muda em intervalos consecutivos, então não é replicado.

A Figura 30, adaptada de Papadias et al. (2002), ilustra uma aRB-tree para diversas regiões, dentre elas a região R_1 da Figura 29. Por exemplo, 710 armazenado na entrada R_1 da R-tree denota a quantidade de objetos dentro de R_1. Já a primeira entrada folha na B-tree para R_1 (1, 150) informa que o número de objetos em R_1 durante o intervalo [1, 1] é 150. De modo similar, a primeira entrada na raiz da mesma B-tree (1, 445) traduz o fato de que existem 445 objetos em R_1 no intervalo [1, 3]. A B-tree corresponde à agregação para o espaço todo, respondendo eficientemente a consultas que envolvem apenas predicados temporais.

O processo de agregação dos resultados de consultas é facilitado, pois se dispensa a visita a nodos totalmente incluídos na janela de consulta (JC). Um exemplo é encontrar todos os objetos na janela de consulta da Figura 31 durante o intervalo [1,3]. A busca começa na raiz da R-tree. A janela de consulta contém R_5 em sua totalidade, então a B-tree de R_5 é acessada. Ela tem o nodo raiz (1, 685) e (4, 445), que denotam os dados resumidos para os intervalos [1, 3] e [4, 5], respectivamente. Então é desnecessário visitar nodos inferiores da B-tree. Portanto, R_5 contribui com a busca apontando o valor 685.

A segunda entrada da árvore, R_6, é sobreposta parcialmente pela JC. Então o nodo é visitado. Dentro dele, apenas R_3 intersecta a JC, e sua B-tree é acessada. A primeira entrada da raiz dela determina que a colaboração de R_3 para o intervalo [1, 2] é 259. Para

completar o resultado, é necessário percorrer o nível inferior, e recuperar o valor agregado de R3 para o tempo 3: 125. O resultado da consulta é a soma 685+259+125.

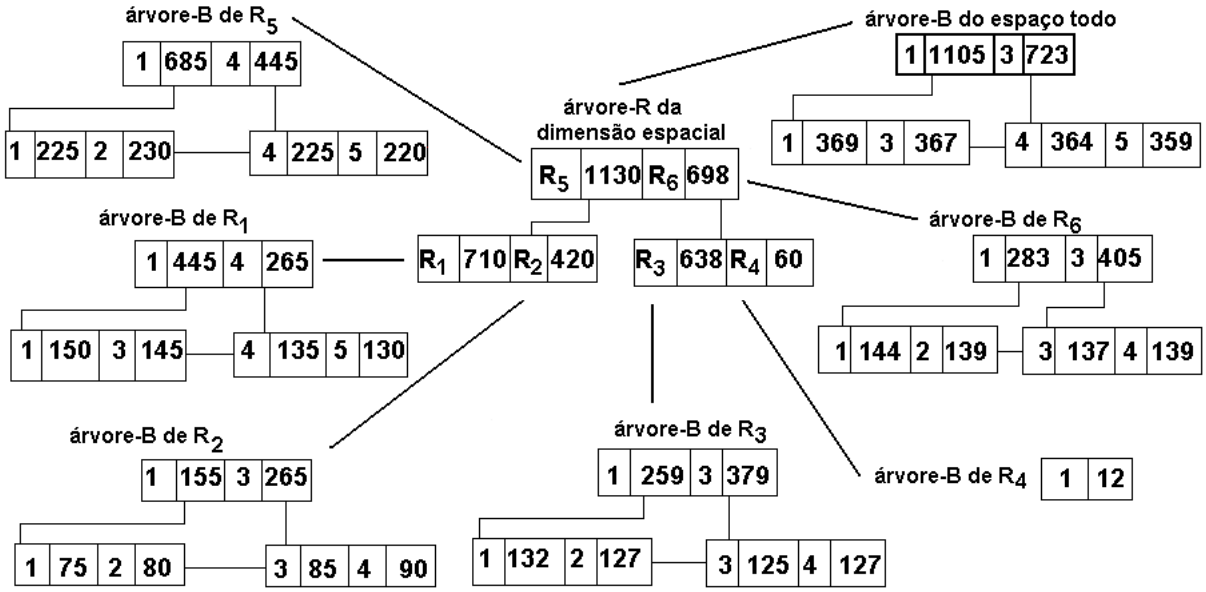


Figura 30 – Um exemplo de aRB-tree para as regiões da Figura 29a.

Da mesma forma que a aR-tree, a aRB-tree responde a consultas com *group-by*. Isto é, para um conjunto de janelas de consulta, tal como: "para todo bairro da cidade, encontre o tráfego médio durante as horas de pico". Porém, diferentemente da aR-tree, a aRB-tree não possui um algoritmo específico para este caso. Na realidade, o mesmo algoritmo para uma janela de consulta é empregado para cada JC do conjunto, culminando em mais que uma visita a nodos sobrepostos por mais de uma JC. Papadias et al. (2002) afirmam que a extensão de algoritmos aptos a realizar a junção espacial entre os dados indexados pela R-tree com os da janela de consulta são mais eficientes neste sentido. É válido ressaltar que a união das janelas não necessariamente cobre o espaço inteiro. Por exemplo, se a janela de consulta JC da Figura 31 representar a união de um conjunto de janelas, é simples perceber que nem todos os objetos do espaço são sobrepostos por ela.

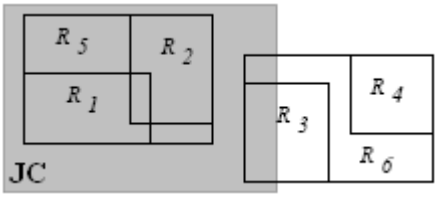


Figura 31 – Exemplo de consulta para a aRB-tree.

Observa-se que, além de uma estrutura de indexação, a aRB-tree consiste no próprio cubo de dados do DW espaço-temporal. Se os dados agregados não são muito dinâmicos, estima-se que o tamanho da estrutura é inferior ao do cubo, pois não replica

informações que permanecem constantes em intervalos de tempo adjacentes. No pior caso, os dados resumidos de todas as regiões se alteram a cada rótulo de tempo, então o tamanho da aRB-tree é o dobro do tamanho do cubo. Isto porque os nodos folha consomem pelo menos metade do espaço.

Os testes de desempenho da a3DR-tree e da aRB-tree foram realizados por Papadias et al. (2002) implementando a B+-tree (FOLK; ZOELLICK, 1992) e a R*-tree (BECKMANN et al., 1990). Dois parâmetros têm influência direta sobre os resultados: o tamanho da JC (corresponde à razão do espaço total do universo) e o comprimento do intervalo de tempo. O primeiro determina que, à medida que a janela cresce, aumenta o número de B-tree's que precisam ser visitadas. O segundo indica que o número de B-tree's acessadas pela aRB-tree é constante, porque longos intervalos têm chance de ser respondidos pelo nodo raiz ou nodos próximos dele.

Por outro lado, conforme crescem os intervalos, a chance da informação agregada também se alterar igualmente aumenta. Logo, há o crescimento linear do custo da a3DR-tree, porque diversas versões do mesmo nodo devem ser recuperadas (cada uma tem valores diferentes da medida agregada). Portanto, para intervalos curtos, a a3DR-tree é menos custosa que a aRB-tree, já que não tem necessidade de acessar uma B-tree para recuperar dados resumidos. Quando os intervalos crescem, a aRB-tree supera a a3DR-tree.

3.6 Considerações Finais

Neste capítulo foram discutidas **estruturas de indexação para DW e DWG**. O Índice Bitmap é alheio aos problemas relacionados ao excesso de dimensões. Por outro lado, ele enfrenta dificuldades quando indexa atributos com alta cardinalidade. A razão para isto é que a complexidade de seu espaço de armazenamento é o produto da cardinalidade do atributo indexado, $|\mathbf{X}|$, pelo número de tuplas da relação, N. No pior caso, $|\mathbf{X}| = N$, e a complexidade atinge indesejáveis N^2. A complexidade do processamento das consultas é função do número de vetores de bits que são acessados, e do número de operações lógicas realizados sobre os mesmos. Há de se destacar a rapidez com que tais operações lógicas podem ser realizadas, visto o auxílio fornecido pelo hardware. O Índice Bitmap torna-se ainda mais importante quando aplicado sobre a forma de índice de junção, sendo perfeitamente adequado para indexar junções em um esquema estrela.

São propostas três técnicas a fim de atenuar os prejuízos causados pela alta cardinalidade de um atributo. *Binning* é a primeira técnica comentada, e consiste em indexar faixas de valores inteiras usando vetores de bits. Por exemplo, criar um vetor para cada uma

das faixas: [0; 20), [20; 40) e [40; 60), segundo a relação de ordem total para os números inteiros. Este recurso de aproximação reduz a quantia de vetores armazenados e pode diminuir o número de vetores acessados em uma consulta. Todavia, introduz falsos candidatos dentre os resultados da consulta, obrigando realizar uma fase de refinamento junto aos dados originais.

Em seguida, foi descrita a técnica de compressão. Se por um lado comprimir bits do Índice Bitmap poupa espaço de armazenamento, por outro lado aumenta o tempo de execução das operações lógicas bit-a-bit. Por fim, a codificação é uma técnica de mapeamento do domínio de valores de um atributo. Ou seja, visando diminuir o número de vetores mantidos em disco e acessados no processamento de consultas, um atributo tem seus valores codificados. Os valores codificados é que são armazenados no índice. A codificação binária possui bom desempenho em termo de número de acessos e espaço de armazenamento. Ela é aplicada sobre os índices Bit-sliced (O'NEIL, P.; QUASS, 1997) e Bitmap codificado (WU, M.; BUCHMANN, 1998). Ambos têm comprovada eficiência em domínios com relação de ordem total.

Conclui-se que o Índice Bitmap é bastante adequado ao DW, e apresenta campos de pesquisa promissores no tocante às três técnicas mencionadas. Se ainda mais trabalhos forem produzidos nestes campos, o bom desempenho alcançado pelo Índice Bitmap poderá ser incrementado. Observa-se ainda que os trabalhos científicos sobre este índice tiveram início na década de 1990, e têm prosseguimento até os dias atuais, seja para atuar em DW ou não. Fica também evidente o equilíbrio que se deve atingir pela combinação das três técnicas descritas.

As dificuldades associadas à implementação e manutenção do Índice Bitmap sob os detalhes de baixo nível dos SGBD, e o fato de não haver nenhum projeto definitivo ou padrão para sua estrutura são obstáculos apontados por O'Neil, E. et al. (2007) que têm inibido sua adoção nos sistemas comerciais. Os obstáculos impostos à adoção do Índice Bitmap também incluem seu alto custo de atualização após novas inserções, o que obriga todos os vetores de bits a serem modificados.

O sistema FastBit foi identificado como sendo atualmente o único software livre na atualidade voltado à construção e a manipulação do índice Bitmap. Ele permite a criação de IBJ e ainda viabiliza a aplicação de técnicas de *binning*, compressão e codificação. Por isso, constitui uma excelente opção. Por outro lado, não foi verificado em FastBit o suporte à indexação de DWG.

Na literatura consultada, não foi encontrado nenhum relato do emprego do Índice Bitmap em DWG. Porém, é possível antecipar a dificuldade de se propor um índice para DWG. Ao mesmo tempo em que se almeja uma boa manipulação dos objetos espaciais, há uma preocupação com a quantidade de dimensões envolvidas nas consultas. E mais: deve-se procurar conciliar as operações *roll-up*, *drill-down*, *slice-and-dice* etc. com as consultas espaciais. Em adição, devem ser propostas soluções para as questões relacionadas às hierarquias de relacionamento de atributos e à própria junção estrela. Outra propriedade que desperta a atenção é a de que objetos espaciais, embora possuam relacionamentos topológicos (interseção, inclusão etc.), não seguem uma relação de ordem total.

O único índice específico para DWG encontrado na literatura é a aR-tree (PAPADIAS et al., 2001). Ele é adequado para domínios em que os objetos espaciais são organizados por meio de hierarquias *ad-hoc*. Todavia, este não é o caso para muitas das aplicações de DWG, que usam hierarquias predefinidas, tais como região $\prec$ nação $\prec$ cidade $\prec$ endereço. Visões materializadas para tais hierarquias provêm um tempo de resposta mais baixo do que acessar a aR-tree. Portanto, não existe um índice eficiente para DWG que possa prover suporte às **hierarquias espaciais predefinidas** e que lide adequadamente com a **multidimensionalidade**. O SB-index, apresentado nesta dissertação, é proposto exatamente para esta finalidade. Além disso, ao contrário da a3DR-tree e da aRB-tree, o presente trabalho não enfoca o contexto espaço-temporal, mas apenas o espacial.

4. SB-INDEX

O *Spatial Bitmap Index*, ou apenas SB-index, é uma estrutura de indexação para DWG cuja proposta:

a) introduz o Índice Bitmap em DWG;
b) reutiliza as técnicas criadas para o Índice Bitmap, tais como *binning*, compressão e codificação;
c) estende ao DWG as vantagens tradicionais já proporcionadas pelo Índice Bitmap ao DW, tal como o suporte eficiente à multidimensionalidade; e
d) otimiza o acesso a DWG com tabelas de dimensão espacial possuindo atributos espaciais organizados em uma hierarquia predefinida.

O SB-index consiste em uma adaptação do Índice de Projeção, incidindo sobre o atributo que compõe a chave primária da tabela de dimensão espacial. Cada entrada do SB-index mantém um valor de chave e um MBR, os quais são associados a um determinado objeto espacial. Embora requeira um percurso seqüencial pela estrutura de indexação, o processamento de consultas do SB-index possui duas virtudes que são fundamentais para um DWG:

a) filtra os objetos espaciais;
b) evita a computação da junção-estrela.

O filtro auxilia o cálculo do predicado espacial. Depois de calculado, o predicado espacial é transformado em um predicado convencional, o qual pode ser computado em conjunto com os outros predicados convencionais da consulta. Esta estratégia permite que a resposta completa da consulta seja obtida por meio de um Índice Bitmap de Junção Estrela.

Ao contrário da aR-tree, o SB-index foi projetado para DWG em que os atributos espaciais são organizados em hierarquias predefinidas. Deste modo, o SB-index viabiliza o processamento de operações OLAP (como roll-up e drill-down) aliadas a predicados espaciais (como **intersecta**, **está contido** e **contém**). É importante ressaltar que este trabalho não investiga o uso do SB-index sobre medidas espaciais.

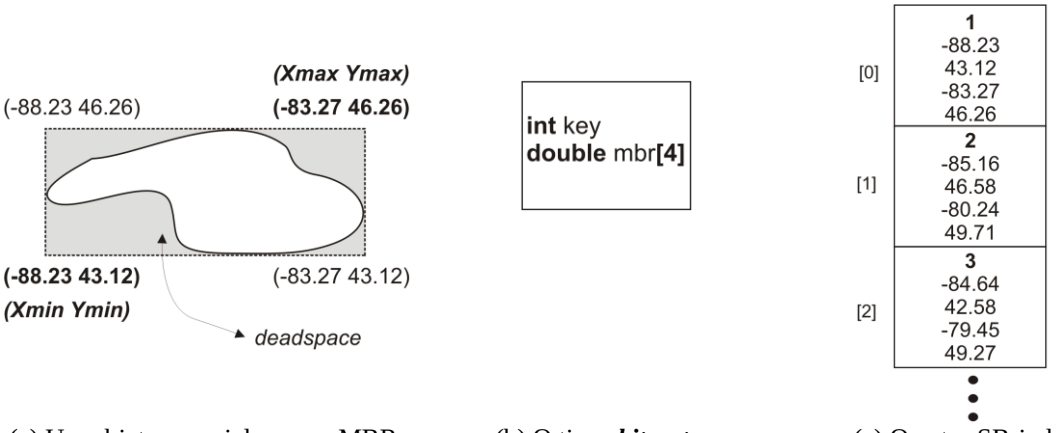
Nas seções seguintes, são descritos a estrutura de dados, o procedimento de construção do índice, o processamento de consultas do SB-index e os testes de desempenho realizados para validar a proposta.

4.1 Estrutura de Dados do SB-index

Definiu-se o tipo de dados ***sbitvector*** (*Spatial Bit-Vector*) contendo um valor de chave e um MBR. A chave referencia a chave primária da tabela de dimensão espacial e

identifica o objeto espacial representado pelo MBR correspondente. A Figura 32a exemplifica como é obtido o MBR a partir de um objeto espacial com a forma geométrica de um polígono. Já a Figura 32b exibe a estrutura de dados do tipo ***sbitvector***. O MBR é definido por dois pares de coordenadas, traduzidos em quatro números reais de dupla precisão: ($X_{mín}$, $Y_{mín}$) e ($X_{máx}$, $Y_{máx}$). Tendo em vista que o emprego de chaves artificiais (*surrogate keys*) é comum em DW, a chave primária pode ser expressa por um número inteiro. Portanto, o tamanho de um objeto do tipo ***sbitvector*** é obtido pela expressão $s = sizeof(int) + 4 \times sizeof(double)$**.**

Conceitualmente, o SB-index é um vetor unidimensional do tipo ***sbitvector***, como mostra a Figura 32c. A i-ésima entrada do SB-index aponta para o i-ésimo vetor de bits de ***F***, conforme ilustra a Figura 22. Por exemplo, considere-se que F[0] representa lo_suppkey = 1 na tabela de fatos. Então, SB-index[0] mantém o valor de chave 1 e o MBR do objeto espacial identificado por s_suppkey = 1. Além disso, para acessar o vetor de bits associado a SB-index[0] requer-se apenas aplicar o deslocamento 0 para ler F, ou seja: F[0]. Logo, SB-index[0] aponta implicitamente para o vetor de bits que é apontado por F[0]. Por fim, manter o SB-index como um *array* residente na memória primária pode trazer sérias limitações ao seu uso. Por isso, o SB-index é implementado como um arquivo seqüencial, em disco, cujas páginas possuem um tamanho fixo.



(a) Um objeto espacial e o seu MBR. (b) O tipo ***sbitvector***. (c) O vetor SB-index.

Figura 32 – Elementos básicos para estrutura de dados do SB-index.

4.2 Construção da Estrutura de Indexação

A construção do SB-index requer uma conexão ao banco de dados que mantém o DWG. Primeiramente ocorre o processo de **extração**, que executa uma consulta SQL realizando as seguintes tarefas:

a) seleciona da tabela de dimensão espacial aquele atributo que compõe a chave primária bem como o atributo espacial desejado;

b) classifica os resultados da consulta pela chave primária, em ordem crescente;

c) obtém o MBR dos objetos espaciais envolvidos.

Para cada tupla resultante da resposta da consulta, cria-se uma entrada em um *buffer* do tipo ***sbitvector***, residente na memória primária. Quando o buffer fica cheio, ele é escrito em uma página de disco, na etapa chamada de **armazenamento**. O processamento destas duas etapas se repete até que todas as tuplas resultantes da consulta SQL tenham sido extraídas e posteriormente armazenadas. A classificação promovida por esta consulta objetiva criar um arquivo de índice ordenado pelos valores da chave primária. Conseqüentemente, SB-index[i] refere-se a F[i], como descrito na seção anterior. A existência de uma hierarquia entre os atributos espaciais define que objetos de diferentes níveis de granularidade são associados a atributos distintos no banco de dados. Esta separação deve ser mantida no SB-index, resultando na construção de um índice para cada atributo da hierarquia.

A Figura 33 ilustra o processo de construção do SB-index. Tanto o *buffer* quanto uma página de disco possuem *L* entradas, sendo que *L* é o número máximo de entradas do tipo ***sbitvector*** que podem ser armazenados em uma página de disco. O tamanho de uma entrada do tipo ***sbitvector*** ocupa $s = sizeof(int) + 4 \times sizeof(double) = 4 + 4 \times 8 = 36$ bytes. Se uma página de disco tem 4096 bytes, então ela comporta $L = (4096\ DIV\ 36) = 113$ entradas, assim o tamanho do buffer deve ser no mínimo de $113 \times 36 = 4068$ bytes. A fim de evitar a fragmentação de entradas entre páginas de disco distintas, uma porção de bytes inutilizados deve separá-los. Neste exemplo, a porção de bytes inutilizados corresponde a $U = 4096 - 4068 = 28$ bytes. A estratégia de evitar a fragmentação visa reduzir o número de acessos a disco no processamento de consultas usando o SB-index. Em caso de fragmentação, pode ser necessário realizar dois acessos a disco para obter uma entrada do índice.

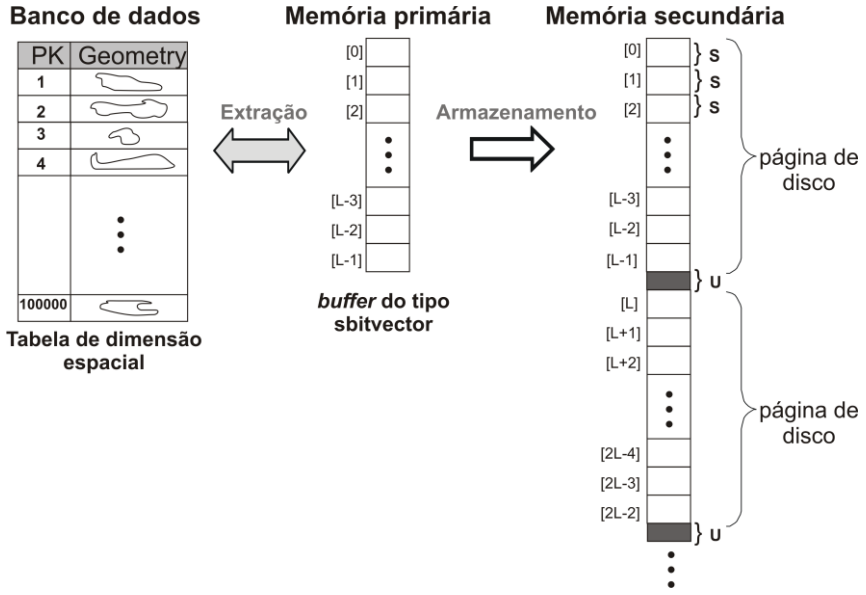


Figura 33 – O processo de construção do SB-index.

A Figura 34 exibe o algoritmo de construção do SB-index.

Sejam:

L: a quantidade de objetos do tipo ***sbitvector*** que uma página de disco é capaz de armazenar;
U: a porção de bytes inutilizados entre duas páginas de disco;
idx: o arquivo do SB-index;
T: uma tabela de dimensão espacial;
pk: o atributo que compõe a chave primária da tabela **T**;
ae: o atributo espacial da tabela **T**;

```
Construir ( L, U, idx, T, pk, ae )
Saída: Arquivo do SB-index devidamente construído em disco
01  recordptr ← execute_sgbd (select pk, xmín(ae), ymín(ae), xmáx(ae),
    ymáx(ae) from T order by pk)
02  i ← 0
03  abra (idx)
04  enquanto (recordptr ≠ eof ) faça
05        enquanto (i ≤ L) e (recordptr ≠ eof ) faça
06         buffer[i].pk ← recordptr.pk
07         buffer[i].xmín ← recordptr.xmín
08         buffer[i].ymín ← recordptr.ymín
09         buffer[i].xmáx ← recordptr.xmáx
10         buffer[i].ymáx ← recordptr.ymáx
11         i ← i + 1
12         recordptr.próximo( )
13        fim-enquanto
14        i ← 0
15        escreva (buffer, idx)
16        avance (U, idx)
17  fim-enquanto
18  feche (idx)
```

Figura 34 – Algoritmo da construção do SB-index.

No algoritmo da Figura 34, a linha 01 consiste em executar uma consulta no SGBD (**`execute_sgbd`**), que busca pela chave primária da tabela de dimensão espacial, além dos pares de coordenadas ($X_{mín}$ $Y_{mín}$) e ($X_{máx}$ $Y_{máx}$) de um atributo espacial desta tabela. A obtenção destas coordenadas usa as funções de manipulação de objetos geográficos do SGBD. Os resultados da consulta compõem um conjunto de registros que podem ser acessados pelo ponteiro `recordptr`. O ponteiro, inicialmente, referencia o primeiro registro. Depois que o arquivo do SB-index é aberto, inicia-se uma repetição que cessa apenas depois que o ponteiro percorra todos os registros (linhas 04 até 17).

Em cada iteração, deve-se preencher um *buffer* do tipo ***sbitvector,*** de comprimento `L`, (linhas 05 até 13) com dados resultantes da consulta submetida. Depois, escreve-se o conteúdo do *buffer* em uma página de disco do arquivo do índice (linha 15). Em seguida, avança-se o ponteiro do arquivo do índice em `U` bytes, a fim de evitar a fragmentação de páginas de disco (linha 16). O preenchimento do *buffer* (linhas 05 a 13) consiste em copiar

para uma entrada os dados de um registro resultante da consulta. O preenchimento dura até que o *buffer* se torne cheio (`i>L`) ou até que os registros já tiverem sido todos lidos (`recordptr = eof`). Depois de copiar os dados de um registro para uma entrada do *buffer*, o ponteiro de leitura dos registros é movido para o próximo registro (linha 12).

4.3 Processamento de Consultas Usando SB-index

As consultas SOLAP que incidem sobre o DWG possuem predicados convencionais e espaciais. O SB-index avalia o predicado espacial e indica as chaves primárias dos objetos espaciais que satisfazem este predicado. De posse destas chaves, cria-se um novo predicado convencional que é combinado àqueles já existentes na consulta. Então, um Índice Bitmap de Junção pode ser empregado para avaliar todos eles e prover a resposta completa para a consulta SOLAP. Assim, como ilustra a Figura 35, o processamento de consultas usando o SB-index é dividido em **duas etapas**. A primeira exige um percurso seqüencial sobre o SB-index e a adição de candidatos (possíveis respostas) a uma coleção (passos 1 a 3 da Figura 35). A segunda etapa requer checar quais candidatos são realmente respostas da consulta submetida ao DWG, criando um predicado convencional para as respostas encontradas (passos 3 a 5 da Figura 35).

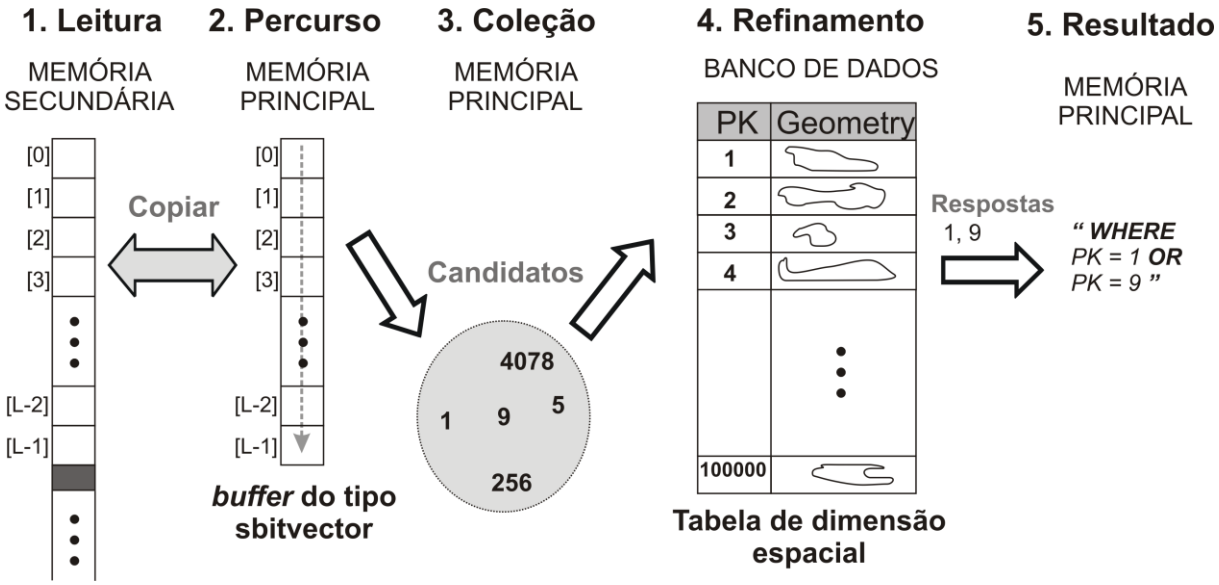


Figura 35 – O processamento de consultas usando o SB-index.

Durante a busca feita sobre o arquivo do SB-index (passo 1), cada página de disco deve ser lida e copiada para um *buffer* na memória primária. Depois da cópia, percorre-se o *buffer* seqüencialmente, verificando para cada entrada se o seu MBR satisfaz o predicado espacial (passo 2). Em caso positivo, o valor da chave correspondente deve ser adicionado a

uma coleção de dados (passo 3). Após a busca seqüencial sobre o SB-index, todos os candidatos que satisfizerem o predicado espacial terão sido colecionados. Basicamente, esta etapa **filtra** as possíveis respostas.

Uma segunda etapa verifica quais candidatos são, de fato, respostas para o predicado espacial. Esta etapa é o **refinamento**, que acessa a tabela de dimensão espacial usando as chaves primárias dos candidatos (passo 4). Ele obtém os objetos espaciais originais e os avalia em relação ao predicado espacial da consulta. Esta avaliação é realizada usando as funções disponibilizadas pelo SGBD. Se um determinado objeto satisfaz o predicado espacial, então a sua chave primária é usada para compor um novo predicado convencional (passo 5). É exatamente neste ponto que o SB-index transforma o predicado espacial em um predicado convencional, da maneira que segue.

As chaves primárias de todos os objetos espaciais que satisfazem o predicado espacial são concatenadas a uma cadeia de caracteres (*string*). A Figura 35 exemplifica esta concatenação: primeiro concatena-se “PK=1”, depois, “OR PK=9”, resultando em “PK=1 OR PK=9”. A *string* resultante representa fielmente o predicado convencional que é gerado a partir do predicado espacial. Considerando que a consulta SOLAP foi escrita sem o predicado espacial e que também consiste em outra *string*, então basta concatenar a *string* do predicado convencional à *string* da consulta SOLAP, para que a transformação do predicado espacial em predicado convencional esteja completa. Por fim, esta nova consulta é repassada ao software FastBit, o qual manipula um Índice Bitmap de Junção para respondê-la.

O algoritmo do processamento de consultas usando o SB-index é apresentado na Figura 36.

Sejam:

idx: o arquivo do SB-index;
L: a capacidade de objetos do tipo ***sbitvector*** que uma página de disco armazena;
R: um relacionamento espacial (**intersecta**, **contém** ou **está contido**);
jc: a janela de consulta espacial;
candidatos: uma porção da memória primária para acomodar as chaves primárias dos objetos espaciais considerados candidatos à resposta do predicado espacial;
predicado_conv: uma *string* que conterá o predicado convencional resultante do processamento de consultas do SB-index;
T: uma tabela de dimensão espacial;
pk: o atributo que compõe a chave primária da tabela **T;**
ae: o atributo espacial da tabela **T** (i.e. geometria do objeto);
consulta: a *string* da consulta SOLAP sem o predicado espacial;
resposta: o conjunto solução da consulta;

```
Busca_Seqüencial(idx, L, R, jc, candidatos, predicado_conv,
                         T, pk, ae, consulta, resposta)
Saída: resposta da consulta SOLAP
Declarações: página, buffer
01  abra (idx)
02  n ← 0
03  enquanto não (eof (idx))
04        leia (idx, página)
05        copie (página, buffer)
06        para i ← 0 até L faça
07           se R(buffer[i], jc)
08              então  candidatos[n] ← buffer[i].pk
09                     n ← n + 1
10        fim-para
11  fim-enquanto
12  feche (idx)
13  para i ← 0 até n faça
14    se (execute_sgbd(select R(jc, ae) from T where pk=candidatos[i]))
15      então concatene (candidatos[i], predicado_conv)
16  fim-para
17  concatene (predicado_conv, consulta)
18  resposta ← execute_fastbit (consulta)
```

Figura 36 – Algoritmo do processamento de consultas usando o SB-index.

O algoritmo da Figura 36 inicia abrindo o arquivo do SB-index e zerando a variável contadora de candidatos (linhas 01 e 02). As linhas 03 até 11 mostram um processo iterativo que coleciona todas as possíveis respostas do predicado convencional, percorrendo todas as entradas do SB-index. Neste processo, lê-se uma página de disco do índice para depois copiá-la num *buffer* (linhas 04 e 05). Um novo processo de repetição é responsável por percorrer todas as entradas do *buffer*, testando se o MBR de cada entrada satisfaz o relacionamento espacial (linha 07). Caso o relacionamento se verifique, então copia-se a chave primária da entrada para a coleção de candidatos (linha 08), e incrementa-se o contador de candidatos (linha 09).

Após a leitura integral do índice, fecha-se o arquivo correspondente (linha 12). Neste momento, é necessário refinar cada uma das entradas da coleção de candidatos. O refinamento também é um processo iterativo, como mostram as linhas 13 até 16. Cada entrada da coleção tem a sua chave primária fornecida para uma consulta no banco de dados. Esta consulta testa se o objeto espacial representado pela chave satisfaz o relacionamento espacial (linha 14). Em caso positivo, esta chave deve ser concatenada à *string* do predicado convencional. Por simplicidade, foi omitida no algoritmo a concatenação do operador lógico `OR`, o qual é mostrado na Figura 35.

Após a composição da *string* do predicado convencional, ela é concatenada à *string* da consulta (linha 17). Deve-se recordar que esta consulta consiste na consulta SOLAP

sem o predicado espacial. Por fim, obtém-se o conjunto resposta da consulta por meio da execução da *string* correspondente, usando o software FastBit.

Evidentemente, o SB-index depende do Índice Bitmap de Junção, e por isso também é suscetível aos efeitos negativos causados pela alta cardinalidade de um atributo. Por outro lado, é possível atenuar esses efeitos empregando as técnicas de *binning*, compressão e codificação. Nesta dissertação, o SB-index usou a técnica de compressão WAH para atenuar os prejuízos causados pela alta cardinalidade.

4.4 Testes de Desempenho

Antes de apresentar os resultados obtidos nos testes de desempenho da construção e do processamento de consultas do SB-index, convém detalhar a amostra dos dados e as consultas utilizadas nos testes, bem como as configurações e as plataformas de hardware e software empregadas. As subseções seguintes descrevem estes itens.

4.4.1 Dados e Consultas

O *Star Schema Benchmark* (SSB) (O'NEIL, P.; O'NEIL, E.; CHEN, X., 2007) foi adaptado para prover suporte à análise espacial, tal como um DWG. Este benchmark é derivado do TPC-H (POESS; FLOYD, 2000), e retrata uma aplicação de varejo, em que produtos são comercializados por fornecedores a clientes durante um certo período. As adaptações realizadas preservaram os dados descritivos e criaram uma hierarquia de atributos espaciais baseada nas tabelas de dimensão que já existiam. Tanto as tabelas de dimensão *Supplier* quanto *Customer* têm a seguinte hierarquia: Região $\prec$ Nação $\prec$ Cidade $\prec$ Endereço. Enquanto os domínios dos atributos relativos ao endereço (*s_address* e *c_address*) são disjuntos, as tabelas de dimensão *Customer* e *Supplier* compartilham atributos relativos à cidade, nação e região.

Segundo Fidalgo et al. (2004), em um esquema de DWG, os dados espaciais não devem ser redundantes, e precisam ser compartilhados sempre que possível. Isto também foi considerado na adaptação realizada sobre o SSB, exibida na Figura 37. Os atributos com sufixo "_geo" armazenam os dados geométricos de feições geográficas. Deve-se ressaltar que este esquema não é normalizado, uma vez que a hierarquia Região $\prec$ Nação $\prec$ Cidade $\prec$ Endereço não retrata um esquema floco-de-neve (KIMBALL; ROSS, 2002). Ao invés disso, o armazenamento das geometrias das tabelas de dimensão é separado em tabelas distintas. Por exemplo, a tabela *City* mantém as geometrias tanto para a tabela *Customer* quanto para a tabela *Supplier*. A escolha por um esquema híbrido ao invés de um floco-de-neve baseia-se em reduzir custos associados a junções entre tabelas.

O esquema da Figura 37 recebeu o nome de GHSSB (*Geographic Hybrid SSB*). Ele foi construído a partir de uma base de dados com fator de escala 10 do SSB. Esta geração de dados criou 60 milhões de tuplas na tabela de fatos, 5 regiões distintas, 25 nações por região, 10 cidades por nação, e um número arbitrário de endereços por cidade, variando de 349 a 455. As cidades, as nações e as regiões são representadas por polígonos, enquanto os endereços são expressos por pontos. Todas as geometrias foram adaptadas do Tiger/Line (U.S. CENSUS, 2007). A distribuição de dados espaciais pode ser observada na Figura 38. O DWG resultante ocupa 15 GB de armazenamento.

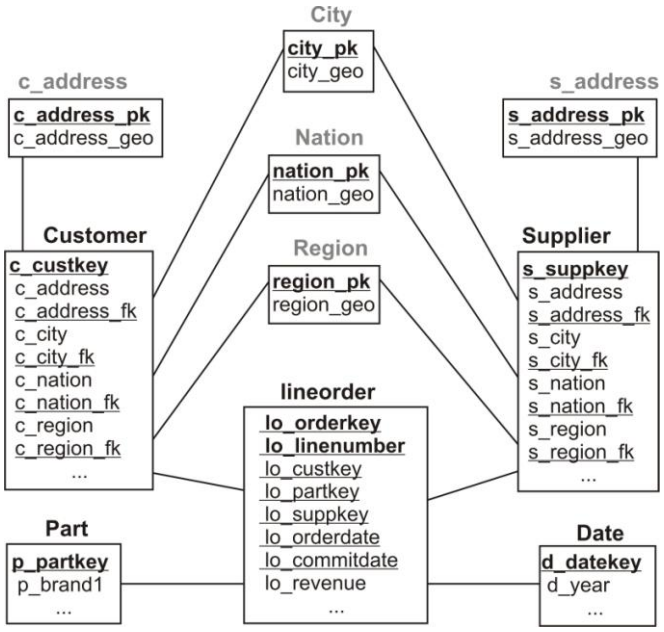


Figura 37 – SSB transformado em um DWG com esquema híbrido: GHSSB.

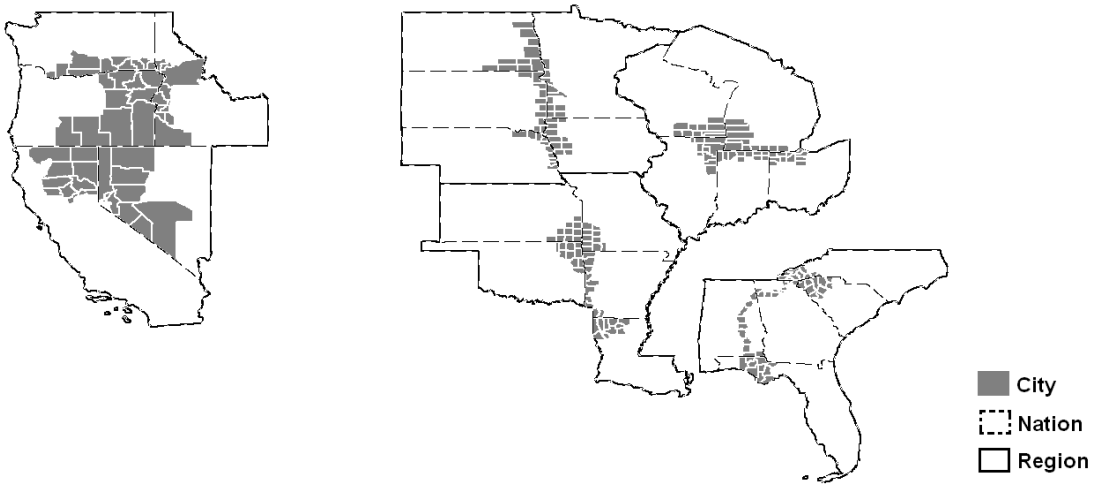


Figura 38 – A distribuição dos dados espaciais do GHSSB.

Foram definidos quatro tipos de consulta SOLAP, baseadas em consultas do SSB. As consultas dos Tipos 1, 2 e 3 se referem à consulta Q2.3 do SSB, enquanto o Tipo 4

consiste em uma adaptação da consulta Q3.3 do SSB. Os predicados convencionais sublinhados nas Figuras 39, 40, 41 e 42 foram substituídos por predicados espaciais baseados em janelas de consulta *ad hoc* (QW). Estas janelas de consulta são quadráticas e têm distribuição correlacionada com os dados espaciais. Os centróides das mesmas são endereços de fornecedores escolhidos arbitrariamente, oriundos da tabela *s_address*.

As janelas de consulta também permitem a agregação de dados em diferentes níveis de granularidade espacial, ou seja, viabilizam as operações de *roll-up* e *drill-down* espaciais. As Figuras 39, 40, 41 e 42 mostram como estas operações podem ser realizadas. Ambas usam quatro janelas de consulta de diferentes tamanhos e com o mesmo centróide. O tamanho da janela está associado à granularidade espacial a que se refere.

Conforme ilustra a Figura 39, a consulta do Tipo 1 consiste em uma operação de *drill-down/roll-up* cujas janelas de consulta verificam o relacionamento espacial **está contido** no nível de granularidade Endereço, e **intersecta** nos níveis de granularidade Cidade, Nação e Região. As janelas de consulta do Tipo 2, por sua vez, avaliam o relacionamento espacial **está contido** em todos os níveis, como mostra a Figura 40. No Tipo 3, como mostra a Figura 41, o relacionamento espacial **está contido** é testado no nível Endereço. Porém, nos demais níveis, o relacionamento espacial avaliado é o **contém**. Por fim, o Tipo 4 herda as características do Tipo 1, mas introduz um alto custo extra de junção, pois processa duas janelas de consulta. O comando SQL da consulta do Tipo 4 é exibido na Figura 42.

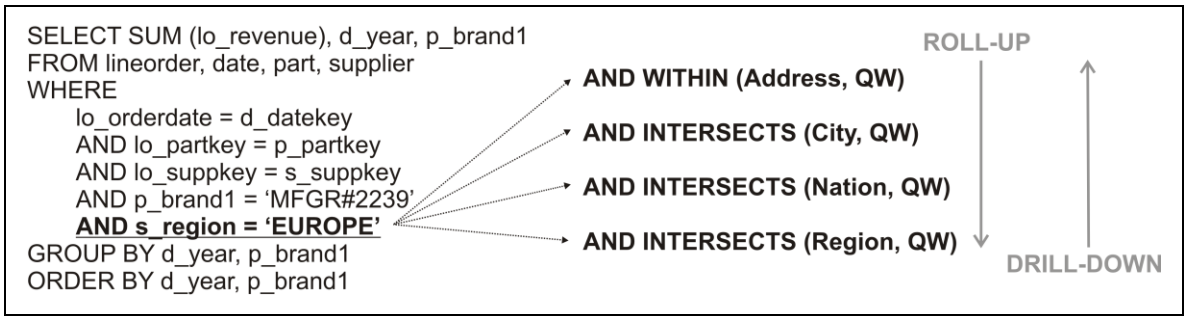


Figura 39 – Adaptação da consulta Q2.3 do SSB para consultas do Tipo 1.

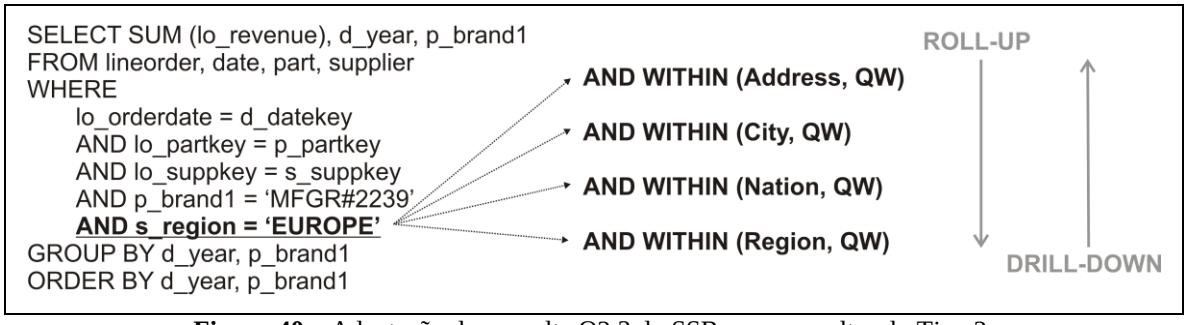


Figura 40 – Adaptação da consulta Q2.3 do SSB para consultas do Tipo 2.

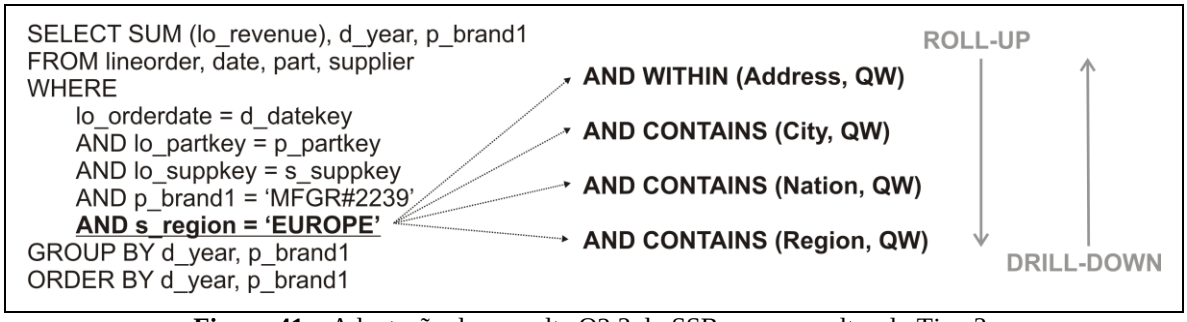


Figura 41 – Adaptação da consulta Q2.3 do SSB para consultas do Tipo 3.

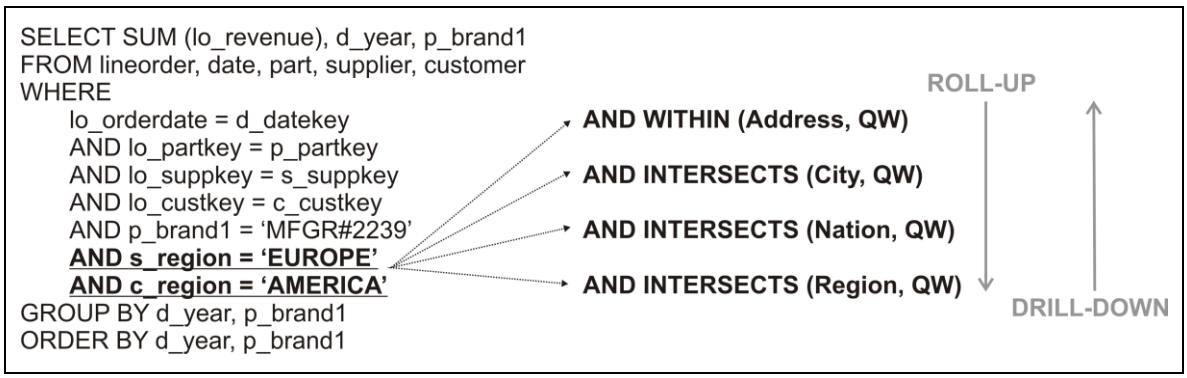


Figura 42 – Adaptação da consulta Q3.3 do SSB para consultas do Tipo 4.

A área de cada janela de consulta foi calculada a partir da área do MBR que envolve o *extent* onde estão dispostos os objetos espaciais. Na Tabela 3, são exibidas as frações do *extent* cobertas pelas janelas de consulta, segundo cada nível de granularidade e segundo o tipo da consulta. O Tipo 2 apresenta janelas maiores, a fim de assegurar a recuperação de objetos espaciais satisfazendo o relacionamento **está contido**. Por outro lado, as menores janelas de consulta estão associadas ao Tipo 3, para assegurar a recuperação de objetos espaciais na verificação do relacionamento **contém**. Neste sentido, a Tabela 4 mostra o número médio de objetos espaciais distintos retornados por cada tipo de consulta, em cada nível de granularidade.

As Figuras 43 e 44 exemplificam janelas de consulta utilizadas no Tipo 1. Na Figura 43, as janelas de número 1, 2 e 3 se referem aos níveis de granularidade Endereço, Cidade e Nação, respectivamente. A Figura 44 mostra uma quarta janela envolvendo as três janelas da Figura 43, referente ao nível de granularidade Região. Estas imagens ilustram fielmente como as operações SOLAP *roll-up/drill-down* são viabilizadas. Por fim, a Figura 45 exibe a disposição aproximada das janelas de consulta do Tipo 4. Embora não estejam exemplificados, os Tipos 2 e 3 possuem janelas de consulta definidas de forma análoga.

Tabela 3 – Frações do *extent* sobrepostas por cada janela de consulta.

	Tipo da consulta		
Granularidade	**Tipo 1**	**Tipo 2**	**Tipo 3**
Endereço	0,001%	0,01%	0,00001%
Cidade	0,05%	0,1%	0,0005%
Nação	0,1%	10%	0,001%
Região	1%	25%	0,01%

Tabela 4 – Número médio de objetos espaciais distintos retornados por Tipo de consulta.

	Consulta			
Granularidade	**Tipo 1**	**Tipo 2**	**Tipo 3**	**Tipo 4**
Endereço	22,2	190,4	1,0	-
Cidade	5,6	1,4	0,8	13,0
Nação	1,6	1,4	0,8	-
Região	1,2	0,4	1,0	-

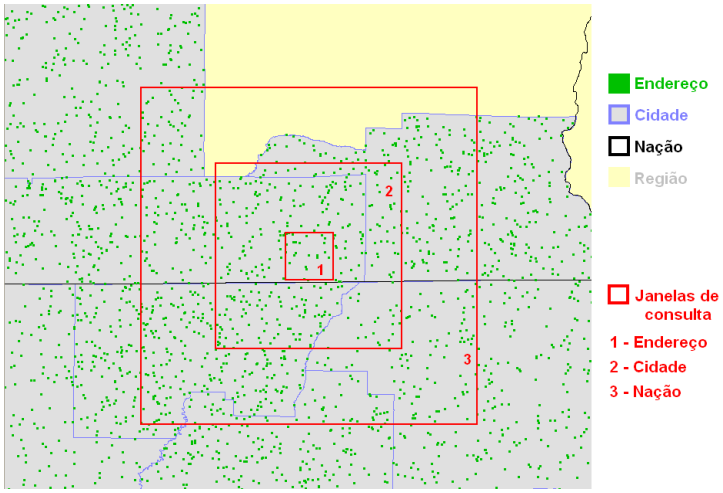


Figura 43 – Janelas de consulta utilizadas no Tipo 1: níveis Endereço, Cidade e Nação.

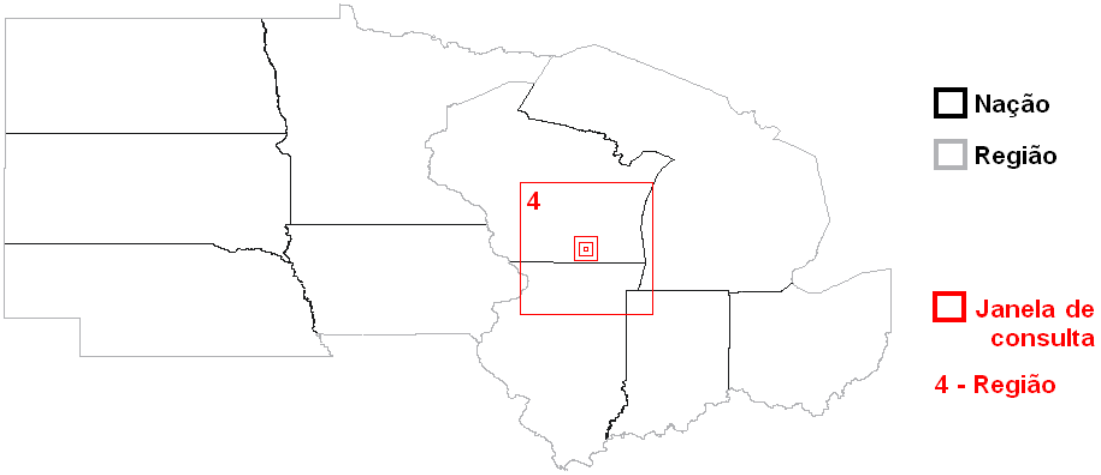


Figura 44 – Janela de consulta utilizada no Tipo 1: nível Região.

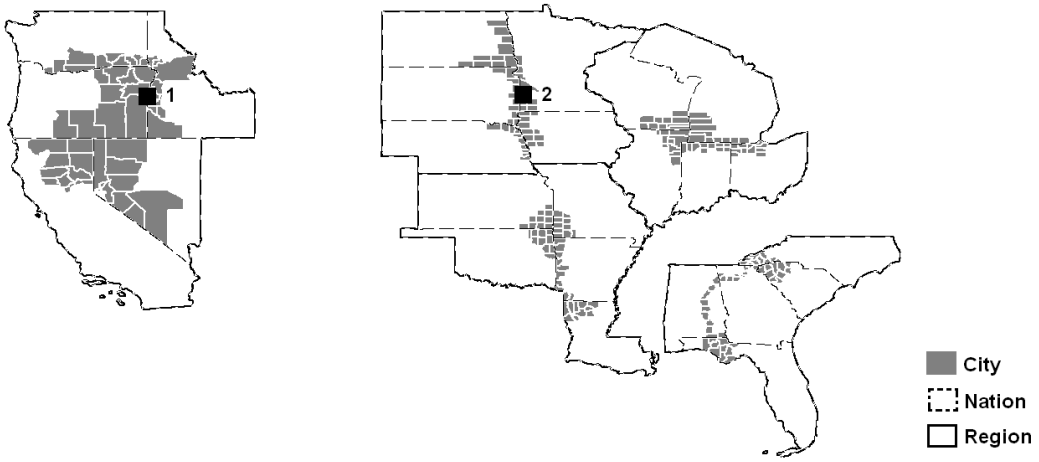


Figura 45 – Disposição aproximada das janelas de consulta presentes no Tipo 4.

4.4.2 A Plataforma Computacional e as Configurações dos Testes

Os experimentos foram conduzidos em um computador com processador Pentium D de 2.8 GHz, 2 GB de memória primária, disco rígido SATA de 320 GB e 7200 RPM, sistema operacional Linux CentOS 5.2, PostgreSQL 8.2.5 e PostGIS 1.3.3. Este SGBD foi selecionado porque é um software livre reconhecido e eficiente. Ainda, o PostGIS é que habilita no PostgreSQL os tipos de dados e as funções espaciais de acordo com os padrões OGC (*Open Geospatial Consortium*) (OPENGIS, 2007). Foram propostas ainda três configurações distintas de testes baseadas no esquema GHSSB:

a) **C1**: computação da junção-estrela auxiliada pelo índice espacial R-tree (GUTTMAN, 1984) sobre os atributos espaciais;
b) **C2**: computação da junção-estrela auxiliada pelo índice espacial GiST (HELLERSTEIN; NAUGHTON; PFEFFER, 1995; GIST, 2008) sobre os atributos espaciais;
c) **C3**: SB-index para evitar a junção-estrela.

As configurações C1 e C2 representam os recursos disponíveis atualmente nos SGBD, para o processamento de consultas SOLAP. A configuração C3 se refere à estrutura de indexação desenvolvida neste trabalho: o SB-index, que foi implementado usando a linguagem de programação C++ e compilado com gcc 4.1. Particularmente nestes testes, a página de disco foi ajustada para 4 KB, e os índices bitmap foram construídos sem métodos de codificação ou binning, porém com a técnica de compressão WAH.

Para investigar os custos de construção do SB-index, foram medidos o número de acessos a disco, o tempo decorrido (em segundos) e o espaço de armazenamento requerido para acomodar o índice. Os experimentos referentes ao processamento de consultas foram realizados, submetendo-se 5 operações completas de *roll-up* sobre dados organizados segundo o esquema GHSSB. Para cada nível de granularidade, coletou-se o tempo decorrido da

consulta em segundos, e posteriormente, calculou-se a média das medições obtidas. Especificamente para a configuração C3, também foi aferido o número de acessos a disco.

4.4.3 Resultados de Desempenho

As subseções seguintes relatam os resultados obtidos na construção do SB-index e no seu processamento de consultas, para o contexto do esquema GHSSB. Notou-se que o SB-index requer pouco espaço de armazenamento em relação aos Índices Bitmap de Junção. Além disso, ele é capaz de reduzir o tempo de resposta das consultas SOLAP em mais de 90%. Apresenta-se ainda uma análise do comportamento do SB-index frente a diferentes volumes de dados, mostrando que a sua eficiência não é afetada por este aspecto.

4.4.3.1 Construção

A Tabela 5 exibe os resultados obtidos na construção do SB-index. O Índice Bitmap de Junção construído usando FastBit ocupa 3,4 GB, e sua construção demorou 12.437,70 segundos (aproximadamente 3h28min). Foram indexados todos os atributos convencionais necessários para o processamento das consultas definidas na Seção 4.1. Conforme mostra a coluna *Acréscimo*, o SB-index requer um acréscimo máximo de 0,10% aos requisitos de armazenamento, especificamente no nível de granularidade Endereço. Nos outros três níveis de granularidade, este aumento é ínfimo se comparado ao tamanho do Índice Bitmap de Junção.

Tabela 5 – Medidas coletadas na construção do SB-index para o esquema GHSSB.

	Tempo decorrido (s)	Acessos a disco	Tamanho do SB-index	Acréscimo (%)
Endereço	18	886	3.5 MB	0,10 %
Cidade	4	4	16 KB	0,00047 %
Nação	4	2	8 KB	0,000235 %
Região	2	2	8 KB	0,000235 %

No nível de granularidade Endereço, tem-se 100.000 entradas para serem acomodadas em páginas de disco de 4096 bytes, e L igual a 113 entradas por página de disco (4096 DIV 36 = 113). Sabendo-se que 100.000 ÷ 113 $\simeq$ 885 e somando uma página de disco para o cabeçalho do arquivo seqüencial, tem-se 886 acessos a disco. Observando os demais níveis de granularidade, conclui-se que quanto maior for o número de objetos espaciais, mais espaço ocupará o índice e maiores serão o número de acessos a disco e o tempo de sua construção. Em particular, o número de acessos a disco também depende do tamanho da página de disco utilizada, pois páginas maiores demandam menos acessos.

4.4.3.2 Processamento de Consultas

Tipo de Consulta 1 – Predicado espacial intersecta

As Tabelas 6 e 7 mostram os resultados referentes ao processamento de consultas do Tipo 1. A Tabela 6 se refere às configurações C1 e C2. Ela mostra que a junção estrela, ainda que auxiliada por eficientes índices espaciais, tais como R-tree e GiST, proporcionou tempos proibitivos para a realização de consultas SOLAP do Tipo 1.

Tabela 6 – Medidas coletadas no processamento de consultas do Tipo 1 (vide Figura 39) executando a Junção Estrela auxiliada por índices espaciais no esquema GHSSB. A coluna C1 se refere ao uso da R-tree, enquanto a coluna C2 se refere ao uso de GiST.

	C1 (s)	C1 (min)	C2 (s)	C2 (min)
Endereço	2.866,51	~ 48	2.853,85	~ 48
Cidade	2.763,17	~ 46	2.758,70	~ 46
Nação	2.766,14	~ 46	2.765,61	~ 46
Região	2.787,83	~ 46	2.790,29	~ 46

A Tabela 7 se refere à configuração C3 (SB-index). Sua quarta coluna, denominada *Redução de tempo*, indica em quanto C3 supera o melhor resultado obtido entre C1 e C2. O SB-index identifica se um ponto está contido em uma janela de consulta, dispensando a etapa de refinamento. Por isso, o nível de granularidade Endereço apresenta maior ganho de desempenho. Embora o refinamento seja necessário para testar se um objeto uni ou bi-dimensional está contido em uma janela de consulta, os ganhos de desempenho do SB-index também são expressivos nos níveis Cidade, Nação e Região. De forma geral, o SB-index atuou de forma bastante satisfatória, reduzindo o tempo de processamento das consultas sempre em mais de 90%. Além disso, analisando a coluna *Tempo decorrido usando índices*, vê-se que o SB-index requereu apenas de 0,65% a 1,39% do tempo total de processamento das consultas.

Tabela 7 – Medidas coletadas no processamento de consultas do Tipo 1 (vide Figura 39) usando o SB-index com o esquema GHSSB. A redução de tempo é calculada em relação ao melhor resultado dentre C1 e C2 mostrado na Tabela 6.

	Tempo decorrido do SB-index (s)	Tempo decorrido do FastBit (s)	Tempo decorrido usando índices (s)	Redução de tempo	Acessos a disco
Endereço	0,09	131,82	131,91	95,38%	886
Cidade	0,07	149,93	150,00	94,56%	4
Nação	0,05	201,65	201,70	92,70%	2
Região	0,05	268,32	268,37	90,38%	2

Tipo de Consulta 2 – Predicado espacial está contido

As Tabelas 8 e 9 exibem os resultados provenientes do processamento de consultas do Tipo 2. A configuração C2 (GiST) foi escolhida porque os experimentos não mostraram diferença significativa entre o uso de R-tree ou GiST. Além disso, C2 se mostrou

mais eficiente que C1 na maioria dos casos. Ainda assim, fica nítido na Tabela 8, que a configuração C2 proporcionou altos tempos de resposta em consultas SOLAP do Tipo 2.

Tabela 8 – Medidas coletadas no processamento de consultas do Tipo 2 (vide Figura 40) executando a Junção Estrela auxiliada por GiST no esquema GHSSB.

	C2 (s)	C2 (min)
Endereço	3.004,44	~ 50
Cidade	2.758,30	~ 46
Nação	1.687,82	~ 28
Região	1.140,09	~ 19

A Tabela 9 mostra os resultados referentes à configuração C3 (SB-index). Sabe-se que o predicado espacial **está contido** é mais restritivo para polígonos do que o predicado espacial **intersecta**. Logo, menos objetos espaciais são conduzidos ao refinamento, e a quantidade de respostas da consulta SOLAP tende a ser menor. Além disso, as tabelas de dimensão dos níveis de granularidade Cidade, Nação e Região mantêm poucos objetos espaciais no esquema GHSSB. Deste modo, a redução de tempo variou de 90,44% a 96,07%. Desta forma, os ganhos para o predicado **contém** foram tão bons quanto os obtidos para o predicado **intersecta**. Ainda, o SB-index representou as ínfimas frações de 0,09% até 0,99% do tempo decorrido do processamento de consultas usando índices.

Tabela 9 – Medidas coletadas no processamento de consultas do Tipo 2 (vide Figura 40) usando o SB-index no esquema GHSSB. A redução de tempo é calculada em relação ao resultado de C2 mostrado na Tabela 8.

	Tempo decorrido do SB-index (s)	Tempo decorrido do FastBit (s)	Tempo decorrido usando índices (s)	Redução de tempo	Acessos a disco
Endereço	0,11	117,99	118,10	96,07%	886
Cidade	1,41	141,67	143,08	94,81%	4
Nação	0,91	160,50	161,41	90,44%	2
Região	0,62	106,56	107,18	90,60%	2

Tipo de Consulta 3 – Predicado espacial contém

As Tabelas 10 e 11 exibem os resultados obtidos no processamento de consultas do Tipo 3. A Tabela 10 retrata a configuração C2 (Junção Estrela auxiliada por GiST), enquanto a Tabela 11 retrata a configuração C3 (SB-index). Novamente, o tempo de resposta do processamento de consultas na configuração C2 mostrou-se proibitivo para todos os níveis de granularidade (superior a 35 minutos). Por outro lado, o SB-index proporcionou ganhos de desempenho que superaram 90% em todos os níveis de granularidade (inferior a 5 minutos). Além disso, a participação do SB-index no tempo decorrido da consulta variou de 0,65% a 1,39%. Isto é, o tempo gasto por SB-index é muito pequeno em relação ao tempo total do processamento da consulta na configuração C3.

Tabela 10 – Medidas coletadas no processamento de consultas do Tipo 3 (vide Figura 41) executando a Junção Estrela auxiliada por GiST no esquema GHSSB.

	C2 (s)	C2 (min)
Endereço	2.779,02	~ 46
Cidade	2.239,54	~ 37
Nação	2.237,05	~ 37
Região	2.848,80	~ 47

Tabela 11 – Medidas coletadas no processamento de consultas do Tipo 3 (vide Figura 41) usando o SB-index no esquema GHSSB. A redução de tempo é calculada em relação ao resultado de C2 mostrado na Tabela 10.

	Tempo decorrido do SB-index (s)	Tempo decorrido do FastBit (s)	Tempo decorrido usando índices (s)	Redução de tempo	Acessos a disco
Endereço	1,11	78,65	79,76	97,13%	886
Cidade	1,13	112,88	114,01	94,91%	4
Nação	1,51	124,31	125,82	94,38%	2
Região	1,74	265,90	267,64	90,61%	2

Tipo de Consulta 4 – Predicado espacial intersecta e duas janelas de consulta espaciais

Por fim, as Tabelas 12 e 13 mostram os resultados coletados no processamento de consultas do Tipo 4. Elas retratam as configurações C2 e C3, respectivamente. Observa-se que o SB-index proporcionou significativa redução no tempo de resposta da consulta: 76,69%. Em particular, devido ao alto custo deste tipo de consulta em relação às consultas dos Tipos 1, 2 e 3, foi utilizada uma plataforma de hardware mais aprimorada, com maior capacidade de processamento e de recursos de armazenamento. Este computador possui um processador Pentium D com 3.00 GHz, 8 GB de memória primária e um disco rígido SATA com 750 GB, 7200 RPM e 32 MB de cache.

Tabela 12 – Medidas coletadas no processamento de consultas do Tipo 4 (vide Figura 42) usando a Junção Estrela auxiliada por GiST no esquema GHSSB.

	C2 (s)
Cidade	130,34

Tabela 13 – Medidas coletadas no processamento de consultas do Tipo 4 (vide Figura 42) usando o SB-index no esquema GHSSB. A redução de tempo é calculada em relação ao resultado de C2 mostrado na Tabela 12.

	Tempo decorrido do SB-index (s)	Tempo decorrido do FastBit (s)	Tempo decorrido usando índices (s)	Redução de tempo
Cidade	0,60	29,78	30,38	76,69%

4.4.3.3 Escalabilidade do Volume de Dados

Nesta seção, são descritos alguns resultados obtidos a partir de uma investigação feita sobre o impacto do crescimento do volume de dados sobre o desempenho do processamento de consultas SOLAP. Especificamente, tal investigação usou consultas do

Tipo 1 (predicado espacial **intersecta**, vide Figura 39). Para tanto, foram construídos DWG usando os fatores de escala 2, 6 e 10, representando volumes de dados crescentes. Enquanto o fator de escala 10 gera 60 milhões de tuplas na tabela de fatos, os fatores 6 e 2 geram 36 e 12 milhões de tuplas, respectivamente. Ou seja, a quantidade de tuplas na tabela de fatos é proporcional ao fator de escala escolhido. É importante frisar que a alteração sobre o volume de dados não afetou o número de objetos espaciais nas tabelas de dimensão. Isto é, foram mantidos 100.000 endereços, 250 cidades, 25 nações e 5 regiões.

Nas Tabelas **14** a 17, constam os resultados dos testes realizados para os níveis de granularidade Endereço, Cidade, Nação e Região, respectivamente. Nestas Tabelas, a coluna *Redução de tempo* indica o quanto a configuração C3 (SB-index) é mais rápida que a configuração C2 (Junção Estrela auxiliada por GiST). Conclui-se que o aumento do volume de dados não prejudica o ganho de desempenho do SB-index. Particularmente, a redução de tempo observada foi muito alta, excedendo 89% em todos os níveis de granularidade.

Tabela 14 – Medidas coletadas no processamento de consultas do Tipo 1 sobre o nível de granularidade Endereço, usando volumes de dados distintos do GHSSB.

	SSB Fator de escala 2 (s)	**SSB Fator de escala 6 (s)**	**SSB Fator de escala 10 (s)**
Configuração C2	594,31	1.803,62	2.853,85
Configuração C3	1,17	1,24	131,91
Redução de tempo	99,80%	99,93%	95,38%

Tabela 15 – Medidas coletadas no processamento de consultas do Tipo 1 sobre o nível de granularidade Cidade, usando volumes de dados distintos do GHSSB.

	SSB Fator de escala 2 (s)	**SSB Fator de escala 6 (s)**	**SSB Fator de escala 10 (s)**
Configuração C2	562,08	1.686,61	2.758,70
Configuração C3	45,27	104,48	150,00
Redução de tempo	91,95%	93,81%	94,56%

Tabela 16 – Medidas coletadas no processamento de consultas do Tipo 1 sobre o nível de granularidade Nação, usando volumes de dados distintos do GHSSB.

	SSB Fator de escala 2 (s)	**SSB Fator de escala 6 (s)**	**SSB Fator de escala 10 (s)**
Configuração C2	545,59	1.694,00	2.765,61
Configuração C3	48,26	103,89	201,70
Redução de tempo	91,15%	93,87%	92,70%

Tabela 17 – Medidas coletadas no processamento de consultas do Tipo 1 sobre o nível de granularidade Região, usando volumes de dados distintos do GHSSB.

	SSB Fator de escala 2 (s)	**SSB Fator de escala 6 (s)**	**SSB Fator de escala 10 (s)**
Configuração C2	552,94	1703,31	2790,29
Configuração C3	56,74	164,87	268,37
Redução de tempo	89,74%	90,33%	90,38%

4.4.3.4 Considerações finais

Os diversos testes de desempenho realizados e descritos nas seções anteriores mostraram resultados relevantes obtidos a partir do uso do SB-index. Primeiramente, observa-se que a estrutura de dados compacta das entradas do SB-index é determinante para um acréscimo muito pequeno aos requisitos de armazenamento. Esta estrutura, que mantém um valor de chave e os dois pares de coordenadas do MBR, proporciona um acréscimo máximo de 0,10%, especificamente no nível de granularidade Endereço (vide Tabela 5).

Conforme a Seção 4.4.3.1, a construção do SB-index não requer muito tempo e ou acessos a disco. O espaço de armazenamento requerido e o número de acessos a disco são diretamente proporcionais à quantidade de objetos espaciais a serem indexados. Os níveis mais altos de granularidade possuem menos objetos espaciais, por isso têm índices menores. Por outro lado, o número de acessos a disco é inversamente proporcional ao tamanho da página de disco adotada.

Além das vantagens observadas durante a construção do índice, foram obtidos resultados expressivos no processamento de consultas SOLAP com SB-index. Estes resultados se estendem a três predicados espaciais: **intersecta**, **contém** e **está contido**. Embora SB-index demande um percurso seqüencial sobre a sua estrutura para filtrar os candidatos à resposta do predicado espacial, esta filtragem elimina os MBR que não se relacionam com a janela de consulta. Conseqüentemente, apenas uma pequena fração dos objetos espaciais precisa ser acessada no banco de dados, reduzindo a parcela deste custoso processamento. Justamente por este motivo é que a participação do SB-index no tempo decorrido da consulta é bastante baixa: não ultrapassou 1,4%.

Após a filtragem e o refinamento das respostas do predicado espacial, reescreve-se a consulta com predicados convencionais. Então, são empregados Índices Bitmap de Junção para respondê-la. O uso destes índices é providencial, evitando as custosas operações de junção intrínsecas das consultas SOLAP. Por isso, isenta de realizar tais operações e auxiliada pela filtragem e pelo refinamento dos objetos espaciais, a configuração C3 (SB-index) é capaz de processar consultas com melhor desempenho que as configurações C1 e C2 (Junção Estrela auxiliada por índices espaciais).

Neste sentido, a redução de tempo observada ficou sempre acima de 90% na existência de uma janela de consulta espacial. As janelas de consulta são mais abrangentes em níveis mais altos de granularidade. Logo, mais tuplas devem fazer parte da resposta da consulta SOLAP. Por isso, a redução tende a ser maior no nível de granularidade endereço, e menor no nível Região. O gráfico da Figura 46 sintetiza, para consultas SOLAP com o

predicado espacial **intersecta**, quanto foi o ganho de desempenho do SB-index em relação às configurações C1 e C2, especificamente no esquema GHSSB.

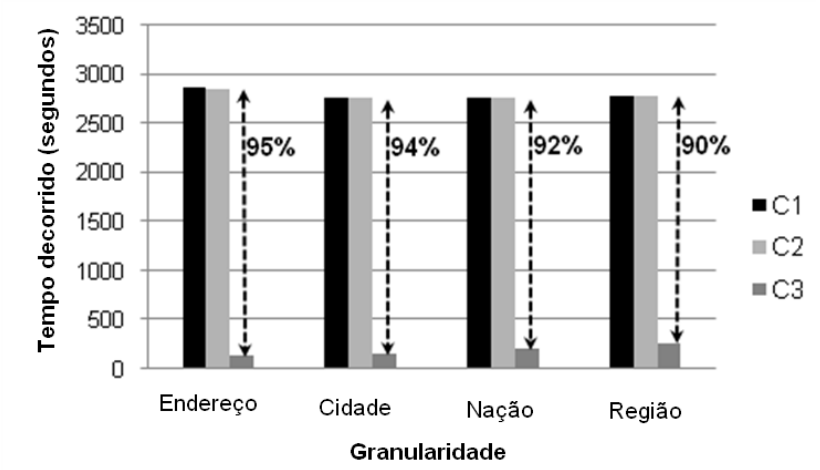


Figura 46 – O SB-index comparado à junção estrela auxiliada por índices espaciais no esquema GHSSB, em consultas SOLAP com o predicado espacial **intersecta**.

Na existência de duas janelas de consulta espaciais, a redução superou 76% especificamente no nível de granularidade Cidade. Por fim, é importante ressaltar que o volume de dados crescente não influenciou negativamente o desempenho do processamento de consultas SOLAP usando SB-index. Isto foi observado nos testes realizados em DWG com fatores de escala 2, 6 e 10, variando-se o volume de dados na tabela de fatos e mantendo-se constante o volume de dados espaciais. Em todos os volumes de dados, a redução de tempo proporcionada pelo SB-index foi superior a 90%.

5. O IMPACTO DA REDUNDÂNCIA DE DADOS ESPACIAIS

A destacada importância do DWG tem motivado o desenvolvimento de diversos trabalhos propondo modelos lógicos e conceituais, como Stefanovic et al. (2000), Malinowski e Zimányi (2004), Bimonte et al. (2005), Fidalgo (2005) e Sampaio et al. (2006). Todavia, pouca atenção tem sido destinada à investigação das seguintes questões: **como a redundância de dados espaciais afeta o tempo de resposta das consultas SOLAP e os requisitos de armazenamento em um DWG?** Neste capítulo, estas questões são abordadas.

O problema da redundância de dados espaciais em DWG é descrito na Seção 5.1. A Seção 5.2 descreve os testes de desempenho realizados com o intuito de auxiliar a investigação e a validação da nova proposta do SB-index. Primeiramente, são comparados os esquemas de DWG redundante (GRSSB) e não redundante (GHSSB), concluindo que a redundância de dados espaciais de fato prejudica o processamento de consultas SOLAP, podendo aumentar o tempo de resposta das consultas de 24% a 815%. Esta conclusão subsidiou a introdução do SB-index em esquemas de DWG redundantes. Como resultado, o SB-index obteve ganhos de desempenho variando de 25% a 95%. Em seguida, o SB-index foi aperfeiçoado para lidar melhor com a redundância de dados espaciais. Uma melhoria baseada na avaliação de MBR distintos foi proposta e, a partir dela, o ganho mínimo de desempenho passou a ser 80%. Finalmente, este capítulo é concluído na Seção 5.3.

5.1 A Redundância de Dados Espaciais

Em um DW, a hierarquia de atributos determina a redundância dos dados sobre uma tabela de dimensão. Esta hierarquia viabiliza a execução das operações OLAP para agregação e desagregação de dados, segundo níveis de granularidade distintos. Estas operações são o *roll-up* e o *drill-down*, respectivamente. Kimball e Ross (2002) afirmam que um esquema de DW redundante (esquema estrela) é mais adequado que um normalizado (esquema floco-de-neve), porque não introduz novos custos de junção entre tabelas. Também é importante frisar que, no DW, os atributos assumem tipos de dados que requerem poucos bytes de armazenamento.

Por outro lado, no DWG é impossível estimar os requisitos de armazenamento de um objeto espacial representado por um conjunto de coordenadas (Stefanovic et al. 2000). Além disso, a avaliação de um predicado espacial é mais custosa que a de um predicado convencional (Gaede e Günther, 1998). Conseqüentemente, optar entre os esquemas não-redundante ou redundante de DWG pode não coincidir com a opção de esquema adotada no

DW convencional. Logo, uma avaliação experimental pode auxiliar os projetistas de DWG a fazerem as suas escolhas.

Stefanovic et al. (2000) foram os primeiros a propor um *framework* para DWG com tabelas de dimensão espacial e medidas espaciais. Em uma tabela de dimensão estritamente espacial, todos os níveis de uma hierarquia de atributos mantêm feições geométricas representando objetos espaciais. Todavia, os autores não discutem os efeitos da redundância de dados espaciais sobre esta tabela de dimensão, mas focam as atenções sobre a materialização seletiva das medidas espaciais.

Fidalgo et al. (2004) afirmaram que a redundância de dados espaciais poderia deteriorar o desempenho geral do DWG. Neste sentido, foi proposto um *framework* para auxiliar o projeto de esquemas multidimensionais geográficos que obrigatoriamente evitam a redundância de dados espaciais. O trabalho foi validado por meio da adaptação do esquema de um DW, transformando-o num DWG. Porém, esta validação não comparou se as consultas SOLAP são mais eficientemente respondidas no DWG não-redundante ou no DWG redundante.

Por fim, Sampaio et al. (2006) apresentaram um modelo lógico multidimensional. Para tanto, o *framework* proposto por Stefanovic et al. (2000) foi reusado, não havendo discussão acerca da redundância dos dados espaciais. Desta forma, é possível notar que nenhum dos mencionados trabalhos tratou a investigação experimental dos efeitos da redundância de dados espaciais sobre DWG, bem como não examinaram se este fator afeta o desempenho de consultas SOLAP. Tal investigação experimental é a motivação do trabalho descrito neste capítulo.

5.2 Testes de Desempenho

Esta seção descreve a investigação experimental motivada nas seções anteriores. Foi necessário criar um DWG sintético com redundância de dados espaciais: o GRSSB (*Geographic Redundant SSB*). Este assunto é tratado na subseção 5.2.1. Além disso, para efetuar uma análise comparativa justa entre o esquema redundante e o não redundante, foram reutilizadas as consultas e as janelas de consultas empregadas no Capítulo 4. A Seção 5.2.2 descreve os testes realizados na construção do SB-index e no processamento de consultas SOLAP sobre o GRSSB, usando a junção estrela auxiliada por índices espaciais e o SB-index.

5.2.1 Ambiente de Testes

O Star Schema Benchmark (SSB) foi adaptado para prover suporte à análise espacial, tal como um DWG redundante. As adaptações realizadas preservaram os dados descritivos e criaram uma hierarquia de atributos espaciais baseada nas tabelas de dimensão que já existiam. Tanto as tabelas de dimensão *Supplier* quanto *Customer* têm a seguinte hierarquia: Região $\prec$ Nação $\prec$ Cidade $\prec$ Endereço. Segundo Stefanovic et al. (2000), *Supplier* e *Customer* são tabelas de dimensão estritamente espaciais. Logo, elas devem manter os dados geográficos da forma como mostra a Figura 47, considerando que os atributos com o sufixo "_geo" armazenam os dados geográficos. Por exemplo, no esquema GRSSB, o mapa do Brasil é armazenado em toda tupla cujo fornecedor se localiza no Brasil. Nitidamente, este esquema promove a redundância de dados espaciais necessária para a realização dos testes.

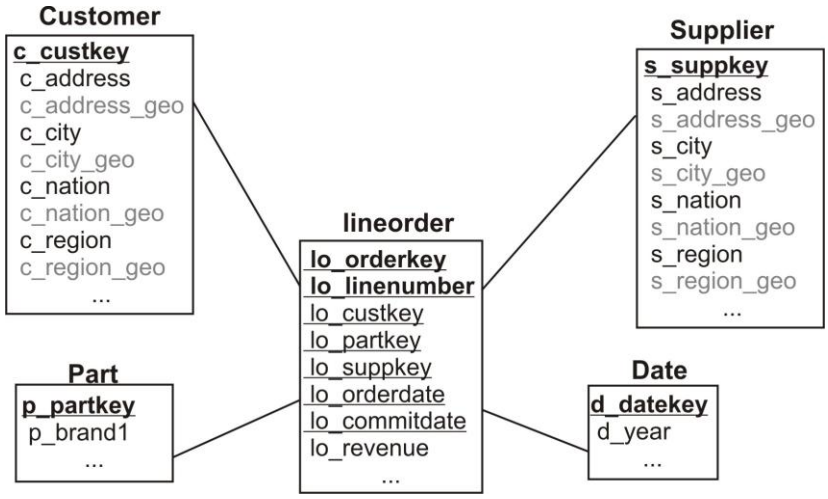


Figura 47 – SSB transformado em um DWG com esquema redundante: GRSSB.

Tal como o GHSSB, o GRSSB conta com 60 milhões de tuplas na tabela de fatos, 5 regiões distintas, 25 nações por região, 10 cidades por nação, e um número arbitrário de endereços por cidade, variando de 349 a 455. As cidades, as nações e as regiões são representadas por polígonos, enquanto os endereços são expressos por pontos. As geometrias foram adaptadas do Tiger/Line (U.S. CENSUS, 2007). O nível de granularidade Endereço não é redundante, porque há uma relação 1:1 entre fornecedores e seus endereços, bem como entre os clientes e os seus endereços.

Por outro lado, os níveis de granularidade Cidade, Nação e Região são redundantes. Observa-se que, quanto menor a cardinalidade do atributo espacial, maior é a quantidade de repetições de um determinado objeto. Logo, se existem menos objetos distintos no domínio de um atributo espacial, então a tendência é que cada um destes objetos seja

repetido um número alto de vezes na tabela. A Tabela 18 exibe o número mínimo e o número máximo de repetições de objetos observados nos diferentes níveis, na tabela de dimensão *Supplier*. A distribuição de dados espaciais é a mesma da Figura 38 (Capítulo 4). A única diferença é que um determinado objeto espacial ocorre mais que uma vez nos níveis Cidade, Nação e Região. O DWG resultante ocupa 150 GB de armazenamento, ou seja, dez vezes mais que o GHSSB. Conforme comentado, no nível de granularidade Região, em que existem apenas 5 objetos distintos, o número de repetições de cada objeto é alto. Por outro lado, no nível Cidade existem 250 objetos distintos, levando a um número menor de repetições de cada um deles.

Tabela 18 – Número mínimo e máximo de repetições de objetos na tabela *Supplier*.

	Repetição de objetos	
Granularidade	**Mínimo**	**Máximo**
Cidade	349	455
Nação	3924	4095
Região	19837	20051

Para estes experimentos, foram utilizados os mesmos tipos de consulta definidos na Seção 4.1.1. Obviamente, adaptações nas consultas SQL foram realizadas, adequando-as ao esquema GRSSB. Uma vez que os tipos de consulta foram reusados nos experimentos deste capítulo, também as janelas de consulta espaciais foram mantidas, segundo os diferentes predicados espaciais. Tais janelas também estão definidas na Seção 4.1.1. A plataforma de hardware e software, as configurações dos testes e as medições dos experimentos são também os mesmos utilizados para a avaliação do esquema GHSSB, como consta na Seção 4.4.2. A seguir, são listadas, novamente, as três configurações de testes:

d) **C1**: computação da junção-estrela auxiliada pelo índice espacial R-tree sobre os atributos espaciais;
e) **C2**: computação da junção-estrela auxiliada pelo índice espacial GiST sobre os atributos espaciais;
f) **C3**: SB-index para evitar a junção-estrela.

5.2.2 Resultados

Esta seção discorre sobre os resultados obtidos nos testes de desempenho realizados sobre o GRSSB. Primeiramente, mede-se o custo da construção do SB-index. Depois, analisa-se o comportamento do processamento de consultas SOLAP. Para os predicados espaciais **intersecta**, **está contido** e **contém**, são feitas duas comparações.

Primeiro, os resultados da junção estrela auxiliada por índices espaciais no GRSSB são confrontados com os resultados da junção estrela auxiliada por índices espaciais no GHSSB. Em seguida, são calculados os ganhos de desempenho proporcionados pelo uso do SB-index no GRSSB, em relação ao tempo gasto pela junção estrela auxiliada por índices espaciais. Mesmo obtendo ótimos resultados com o uso do SB-index, identificou-se um aspecto do mesmo sobre avaliação de MBR que poderia ainda ser melhorado. Portanto, foi proposto e testado um aperfeiçoamento sobre o SB-index.

5.2.2.1 Construção

A Tabela 19 exibe os resultados obtidos na construção do SB-index. O Índice Bitmap de Junção construído usando FastBit ocupa 2,3 GB, e sua construção demorou 11.237,70 segundos (aproximadamente 3h07min). Foram indexados todos os atributos convencionais necessários para o processamento das consultas definidas na Seção 4.4.1. Devido à redundância de dados espaciais, cada SB-index possui 100.000 entradas (cardinalidade da tabela de dimensão *Supplier*). Portanto, estes índices têm o mesmo tamanho e requerem o mesmo número de acessos a disco para serem construídos.

Observando as Tabelas 5 (Seção 4.4.3.1) e 19, nota-se que a redundância de dados espaciais é responsável por gerar índices maiores nos níveis de granularidade Cidade, Nação e Região. Estes índices também levam mais tempo para serem construídos, porque um determinado objeto espacial é recuperado diversas vezes em diferentes tuplas. Mesmo maior, um SB-index redundante provoca um aumento nos requisitos de armazenamento que é quase imperceptível: 0,14%, conforme mostra a quinta coluna, denominada *Acréscimo*.

Tabela 19 – Medidas coletadas na construção do SB-index para o esquema GHSSB.

	Tempo decorrido (s)	Acessos a disco	Tamanho do SB-index	Acréscimo (%)
Endereço	48	886	3,5 MB	0,14%
Cidade	1.856	886	3,5 MB	0,14%
Nação	11.566	886	3,5 MB	0,14%
Região	19.453	886	3,5 MB	0,14%

Em todos os níveis de granularidade, os SB-index acomodam 100.000 entradas em páginas de disco de 4096 bytes (sendo que L = 4096 DIV 36 = 113 entradas por página de disco). Sabendo-se que 100.000 ÷ 113 ~ 885 e somando uma página de disco para o cabeçalho do arquivo seqüencial, tem-se 886 acessos a disco. Logo, páginas de disco maiores demandariam menos acessos a disco. Os resultados desta seção são válidos para todos os tipos de consulta cujos processamentos são avaliados na próxima seção.

5.2.2.2 Processamento de consultas

Tipo de Consulta 1 – Predicado espacial intersecta

A Tabela 20 mostra os resultados referentes ao processamento de consultas do Tipo 1 referentes às configurações C1 e C2. Ela mostra que a junção estrela, ainda que auxiliada por índices espaciais eficientes, resultou em tempos proibitivos para a realização de consultas SOLAP do Tipo 1. Além disso, a coluna *Aumento* mostra em quanto o tempo de resposta do GRSSB superou o do GHSSB. O nível de granularidade Endereço não é redundante, portanto não foi realizada a comparação. O nível Cidade é, dentre os níveis redundantes, aquele que possui menos repetições de objetos espaciais (vide Tabela 18). Observa-se que o aumento também é quase nulo no nível Cidade. Por outro lado, nos níveis de granularidade redundantes Nação e Região, há mais repetição de objetos espaciais. Neles, o aumento variou de 24,74% a 123,13%, indicando que a redundância de dados espaciais afetou negativamente o processamento de consultas do Tipo 1 nas configurações C1 e C2.

Tabela 20 – Medidas coletadas no processamento de consultas do Tipo 1 (vide Figura 39) executando a Junção Estrela auxiliada por índices espaciais no esquema GRSSB. A coluna C1 se refere ao uso da R-tree, enquanto a coluna C2 se refere ao uso de GiST.

	C1 (s)	C1 (min)	Aumento C1	C2 (s)	C2 (min)	Aumento C2
Endereço	2.854,17	~ 48	-	2.831,23	~ 48	-
Cidade	2.773,39	~ 46	1,00%	2.773,10	~ 46	0,52%
Nação	4.047,35	~ 68	46,31%	3.449,76	~ 58	24,74%
Região	6.220,68	~ 104	123,13%	6.200,44	~ 103	122,21%

Os testes do Tipo 1 sobre o GRSSB nas configurações C1 e C2 demonstram tempos de resposta proibitivos. É indispensável prover tal esquema de mecanismos para tornar o processamento de consultas mais ágil. Neste sentido, a Tabela 21 exibe os resultados do processamento de consultas do Tipo 1, usando SB-index (configuração C3). Foram requeridos 886 acessos a disco, independentemente do nível de granularidade. Este número é fixo porque todos os índices possuem o mesmo número de entradas. Aliado a este fato, sabe-se que uma busca seqüencial incide sobre o índice referente a cada nível de granularidade.

Tabela 21 – Medidas coletadas no processamento de consultas do Tipo 1 (vide Figura 39) usando o SB-index no esquema GRSSB. A redução de tempo é calculada em relação ao melhor resultado dentre C1 e C2 mostrado na Tabela 20.

	Tempo decorrido do SB-index (s)	Tempo decorrido do FastBit (s)	Tempo decorrido usando índices (s)	Redução de tempo	Acessos a disco
Endereço	0,09	132,32	132,41	95,32%	886
Cidade	32,80	238,94	271,74	90,20%	886
Nação	661,41	516,71	1.178,12	65,84%	886
Região	3.223,19	1.398,80	4.621,99	25,45%	886

A coluna *Redução de tempo* indica em quanto C3 supera o melhor resultado entre C1 e C2. Vê-se que os ganhos de desempenho obtidos no processamento da consulta SOLAP do Tipo 1 são expressivos, variando de 25,45% a 95,32%. Assim como observado no esquema GHSSB, o nível de granularidade Endereço apresenta os melhores resultados referentes ao tempo de resposta das consultas. Isto ocorre porque o SB-index identifica um ponto no interior de um polígono sem a necessidade da custosa etapa de refinamento.

Por outro lado, a redundância de dados espaciais nitidamente prejudicou o desempenho do SB-index, uma vez que o mesmo MBR precisa ser avaliado múltiplas vezes na busca seqüencial. Além disso, diferentes valores de chave primária referentes a MBR idênticos são adicionados à coleção de candidatos, e posteriormente encaminhadas ao refinamento. Portanto, o mesmo objeto espacial é avaliado diversas vezes durante o refinamento, etapa esta considerada a mais custosa no processamento das consultas.

A redundância de dados espaciais também influencia a parcela de participação do SB-index no tempo total decorrido da consulta. No nível de granularidade Endereço (não-redundante), o SB-index é responsável pela ínfima fração de 0,07%. Já nos níveis Cidade, Nação e Região (redundantes), as frações são 12,07%, 56,14% e 69,74%, respectivamente. Nota-se que à medida que a redundância de dados espaciais aumenta, o tempo de resposta do SB-index supera o tempo de resposta de FastBit. Este fato determina valores menores na redução de tempo, como 25,45% no nível de granularidade Região. Porém, o resultado neste nível em particular não deixa de ser substancial.

Tipo de Consulta 2 – Predicado espacial está contido

A configuração C2 (Junção Estrela auxiliada por GiST) mostrou ser mais eficiente que a configuração C1 (Junção Estrela auxiliada pela R-tree) na maioria dos casos de teste. Por este motivo, no restante desta seção não serão mais tratados os resultados referentes a C1. A Tabela 22 mostra os resultados obtidos no processamento de consultas do Tipo 2, avaliando o predicado espacial **está contido** e a configuração C2. Esta tabela mostra, na coluna *Aumento C2*, em quanto o tempo decorrido de uma consulta no esquema GRSSB superou o tempo decorrido da mesma consulta no esquema GHSSB (vide Seção 4.4.3.2).

Os resultados da Tabela 22 indicam que, nos níveis de granularidade Endereço e Cidade, o processamento de consultas do Tipo 2 tem comportamento similar para os esquemas GHSSB e GRSSB. Nestes dois casos, a redundância de dados espaciais não prejudicou o desempenho. É válido ressaltar que tanto no GRSSB quanto no GHSSB não existem endereços (pontos) repetidos no nível Endereço, e que o teste do relacionamento

espacial **está contido** sobre pontos é menos custoso. Por outro lado, GRSSB determina a redundância de dados espaciais nos níveis Cidade, Nação e Região, os quais mantêm polígonos. Enquanto Cidade apresenta a menor incidência de redundância, os níveis Nação e Região são altamente redundantes. Por isso, foi observado expressivo impacto negativo sobre o tempo de resposta das consultas nos níveis Nação e Região. Isto é, aumentos de 138,56 a 815,96%, respectivamente.

Tabela 22 – Medidas coletadas no processamento de consultas do Tipo 2 (vide Figura 40) executando a Junção Estrela auxiliada por GiST no esquema GRSSB.

	C2 (s)	C2 (min)	Aumento C2
Endereço	2.811,08	~ 47 min	-
Cidade	2.789,53	~ 47 min	1%
Nação	4.026,63	~ 67 min	138,56%
Região	10442,73	~174 min	815,96%

Os tempos de resposta de consultas SOLAP do Tipo 2 no esquema GRSSB usando a configuração C2 são demasiadamente altos. É indispensável tornar tais consultas mais ágeis. Desta forma, a Tabela 23 exibe os resultados obtidos a partir da aplicação do SB-index (configuração C3) no referido cenário. Pode-se observar que os ganhos de desempenho variaram de 78,39% a 94,77%. No predicado espacial **está contido**, a redundância dos dados espaciais não prejudicou drasticamente o desempenho do SB-index, ao contrário do que ocorreu com o predicado espacial **intersecta**. Apesar disso, deve-se enfatizar que a mesma busca seqüencial foi realizada, avaliando o mesmo MBR várias vezes. O fato de o predicado **está contido** ser mais restrito que **intersecta** pode ser apontado como uma das razões pelas quais o desempenho não foi drasticamente afetado pela redundância em C3.

Tabela 23 – Medidas coletadas no processamento de consultas do Tipo 2 (vide Figura 40) usando o SB-index no esquema GRSSB. A redução de tempo é calculada em relação ao resultado de C2 mostrado na Tabela 22.

	Tempo decorrido do SB-index (s)	Tempo decorrido do FastBit (s)	Tempo decorrido usando índices (s)	Redução de tempo	Acessos a disco
Endereço	0,08	146,96	147,04	94,77%	886
Cidade	12,18	151,96	164,14	94,12%	886
Nação	435,09	435,20	870,29	78,39%	886
Região	1.032,06	471,05	1.503,11	85,61%	886

Nos níveis de granularidade Endereço (não-redundante) e Cidade (pouco redundante), é pequena a participação do SB-index no tempo total da consulta: as frações são de 0,05% e 7,42%, respectivamente. Entretanto, à medida que a redundância de dados espaciais cresce, esta participação aumenta e pode inclusive exceder o tempo de resposta do FastBit. No nível de granularidade Nação, ela correspondeu a 49,99% do tempo decorrido da consulta, enquanto que no nível Região correspondeu a 68,66%. Conforme o SB-index passa

a gastar mais tempo, o ganho de desempenho no processamento da consulta SOLAP é reduzido. Mesmo assim, a configuração C3 proveu ganhos de desempenho muito altos: entre 78,39% e 94,77%.

Tipo de Consulta 3 – Predicado espacial contém

A Tabela 24 exibe os resultados para a configuração C2 e o predicado **contém** (Tipo 3) com relação ao esquema GRSSB. Estes resultados de desempenho foram muito parecidos com os do predicado **está contido**, para ambos os esquemas GHSSB e GRSSB, particularmente nos níveis de granularidade Endereço e Cidade. Por outro lado, há notável disparidade no processamento de consultas nos níveis de granularidade Nação e Região, em que a redundância de dados espaciais é maior. No primeiro, o aumento foi de 29,34%, enquanto no segundo ele atingiu indesejáveis 116,03%. Deve-se ressaltar que, nestes níveis, menos objetos satisfazem o predicado **contém**.

Tabela 24 – Medidas coletadas no processamento de consultas do Tipo 3 (vide Figura 41) executando a Junção Estrela auxiliada por GiST no esquema GRSSB.

	C2 (s)	C2 (min)	Aumento C2
Endereço	2.739,64	~ 46	-
Cidade	2.259,66	~ 38	0,89%
Nação	2.893,50	~ 48	29,34%
Região	6.154,28	~ 103	116,03%

A Tabela 25, mostra os resultados obtidos a partir dos testes envolvendo consultas SOLAP do Tipo 3 e a configuração C3 (SB-index). Os ganhos de desempenho do SB-index variaram de significativo (aproximadamente 40%) até muito alto (97,09%). A redundância de dados espaciais prejudicou o desempenho do processamento de consultas do SB-index, particularmente devido ao alto custo do refinamento de geometrias complexas nos níveis de granularidade Nação e Região. Como resultado, o SB-index tem pouca participação no tempo total decorrido da consulta nos níveis Endereço e Cidade: 1,27% e 6,81%, respectivamente.

Tabela 25 – Medidas coletadas no processamento de consultas do Tipo 3 (vide Figura 41) usando o SB-index no esquema GRSSB. A redução de tempo é calculada em relação ao resultado de C2 mostrado na Tabela 24.

	Tempo decorrido do SB-index (s)	Tempo decorrido do FastBit (s)	Tempo decorrido usando índices (s)	Redução de tempo	Acessos a disco
Endereço	1,01	78,61	79,62	97,09%	886
Cidade	8,64	118,08	126,72	94,39%	886
Nação	545,00	260,55	805,55	72,16%	886
Região	2.620,61	1.162,30	3782,91	38,53%	886

Contudo, conforme a redundância de dados espaciais aumenta, o SB-index passa a gastar mais tempo que o FastBit. No nível de granularidade Nação, o processamento de consultas do SB-index corresponde a 67,66% do tempo total da consulta. Já no nível Região, o SB-index corresponde à fração de 69,27%. Este fato foi responsável pela queda de desempenho (expresso pela coluna *Redução de tempo*) da consulta SOLAP. Ainda assim, este ganho se mantém expressivo: quase 40%.

Tipo de Consulta 4 – Predicado espacial intersecta e duas janelas de consulta espaciais

Por fim, os testes de desempenho cujos resultados são exibidos nas Tabelas 26 e 27 estão associados às consultas SOLAP do Tipo 4 (predicado **intersecta** e duas janelas de consulta espaciais). Devido ao alto custo de processamento requerido por esta consulta, usou-se um computador mais sofisticado: processador Pentium D com 3.00 GHz, 8 GB de memória primária e um disco rígido SATA com 750 GB, 7200 RPM e 32 MB de cache.

A Tabela 26, mostra o resultado do teste sobre a configuração C2. Foi coletada apenas uma amostra no nível de granularidade Cidade, pois nos níveis de granularidade Nação e Região os tempos decorridos superaram 4 dias de execução. Isto exigiu a interrupção do processamento, uma vez que tais tempos de resposta são proibitivos. Mesmo no nível Cidade, com pouca redundância de dados espaciais, os resultados de desempenho indicam claramente que a redundância de dados espaciais traz prejuízos ao desempenho do processamento de consultas SOLAP do Tipo 4. Enquanto no esquema GHSSB esta consulta demorou apenas 130,34 segundos (vide Tabela 12, Seção 4.4.3.2), no esquema GRSSB o tempo necessário foi 172.900,15. Ou seja, um aumento impressionante de 132.900%.

Tabela 26 – Medidas coletadas no processamento de consultas do Tipo 4 (vide Figura 42) executando a Junção Estrela auxiliada por GiST no esquema GRSSB.

	C2 (s)	C2 (h)	Aumento C2
Cidade	172.900,15	48	132.900%

A Tabela 27 mostra os resultados obtidos a partir do SB-index (configuração C3), para evitar os custos decorrentes das junções. Mesmo com redundância de dados espaciais, o SB-index obteve excelentes resultados, promovendo um ganho de desempenho de 99,82%. Portanto, o emprego do SB-index foi fundamental para obter um bom tempo de resposta neste tipo de consulta SOLAP.

Tabela 27 – Medidas coletadas no processamento de consultas do Tipo 4 (vide Figura 42) usando o SB-index no esquema GHSSB. A redução de tempo é calculada em relação ao resultado de C2 mostrado na Tabela 26.

	Tempo decorrido do SB-index (s)	Tempo decorrido do FastBit (s)	Tempo decorrido usando índices (s)	Redução de tempo
Cidade	15,40	296,97	312,37	99,82%

5.2.2.3 SB-index NR

Os resultados obtidos na Seção 5.2.2.2 mostram que a redundância de dados espaciais degenerou o desempenho do SB-index. Isto ocorreu porque a busca seqüencial sobre o índice avalia MBR repetidos, bem como a custosa etapa de refinamento testa objetos espaciais repetidos. Este fato motivou a proposição de uma melhoria sobre o SB-index, a fim de evitar a redundância. A idéia principal deste melhoramento reside em avaliar apenas MBR distintos, assim como prevê o esquema GHSSB. A nova estrutura chama-se SB-index NR (*non-redundant SB-index*).

É proposto um segundo nível para o SB-index, que consiste em atribuir uma lista a cada MBR distinto. Cada uma destas listas é armazenada em disco, e mantém todos os valores de chaves associados a um MBR específico. No processamento de uma consulta SOLAP, deve-se testar o relacionamento espacial entre cada MBR distinto e a janela de consulta. Se este relacionamento existir, então apenas um valor de chave deve ser conduzido ao refinamento. Então, se o objeto espacial identificado por esta chave for uma resposta da consulta, devem-se adicionar todas as chaves da lista correspondente ao predicado convencional. Por fim, o predicado convencional é repassado ao software FastBit, o qual computa a solução da consulta completa.

O novo processamento difere daquele apresentado para o SB-index na Seção 4.3. A Figura 48 ilustra todos os seus passos. Os passos de leitura e percurso (1 e 2, respectivamente) não foram alterados, apesar da existência de uma lista associada a cada entrada do SB-index. As entradas 2 e 4 do *buffer* estão destacadas porque seus MBR satisfizeram o relacionamento espacial. Sendo assim, o primeiro valor de chave primária contido na lista associada à entrada 2 do buffer é colecionado como candidato. Este valor é 3. O mesmo ocorre com a entrada 4, isto é, o valor 16 é considerado candidato. Os candidatos são posteriormente conduzidos ao refinamento. Observa-se que apenas o objeto espacial identificado pela chave primária 2 é considerado resposta do predicado espacial. Portanto, a lista encabeçada por ele deve ser percorrida, incluindo todos os valores de chave contidos nela no predicado convencional. Neste percurso, expande-se o conjunto de respostas do predicado espacial, porque são identificadas todas as chaves que se referem ao mesmo objeto. Por fim, a consulta que agora possui apenas predicados convencionais é executada por FastBit, acessando os índices Bitmap de Junção.

O SB-index NR aumentou o número de acessos a disco necessários para a construção do índice. Conforme mostram a Tabela 28 e a Tabela 19 (Seção 5.2.2.1), este número passou de 886 para 100.000, em todos os níveis de granularidade. Este aumento

acontece porque cada um dos 100.000 valores de chave de uma tabela de dimensão precisa ser adicionado a uma lista específica. Ao adicionar um objeto na lista, conta-se um acesso a disco.

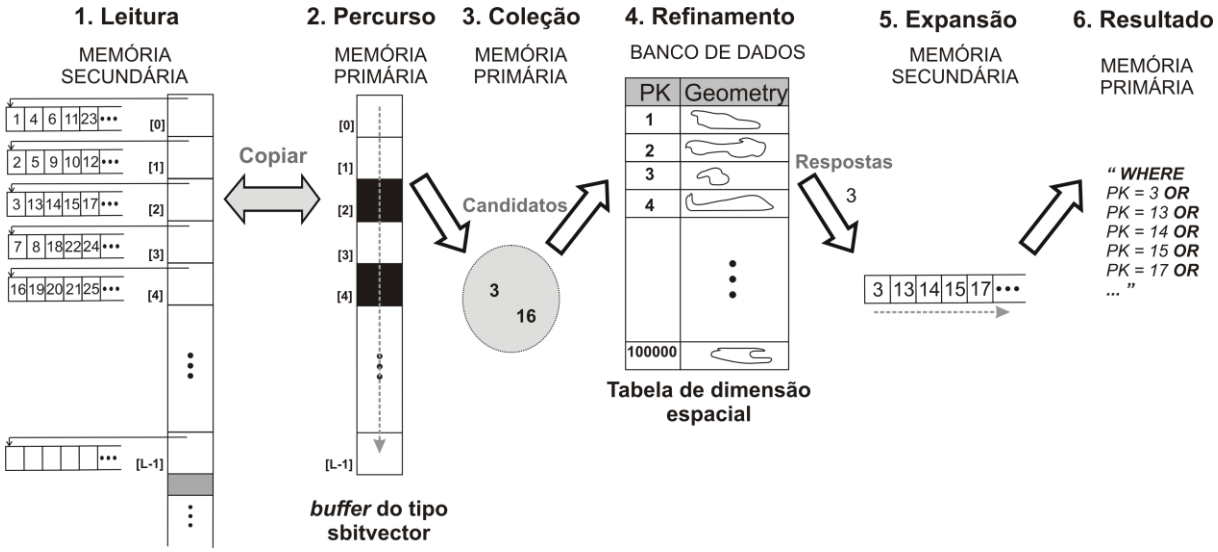


Figura 48 – O processamento de consultas do SB-index NR.

Ainda, o SB-index NR requereu uma pequena porção de espaço de armazenamento, se comparada ao volume do Índice Bitmap de Junção: de 0,16% a 0,20%. Conclui-se, portanto, que o SB-index NR exige mais tempo e mais acessos a disco para que se conclua a construção do índice. Porém, os requisitos de armazenamento não são drasticamente afetados.

Tabela 28 – Medidas referentes ao SB-index NR aplicado ao GRSSB: construção do índice.

	Tempo decorrido (s)	Acessos a disco	Tamanho doSB-index
Cidade	2.005	100.000	4,81 MB
Nação	11.428	100.000	3,93 MB
Região	19.446	100.000	3,86 MB

Os benefícios trazidos pelo SB-index NR têm mais destaque no processamento de consultas SOLAP. A Tabela 29 exibe os resultados dos testes realizados sobre o GRSSB usando consultas do Tipo 1. Comparando a Tabela 29 e a Tabela 20 (Seção 5.2.2.2), vê-se que o ganho de desempenho proporcionado foi muito alto em relação à junção estrela auxiliada por índices espaciais (C1 e C2). Este ganho é expresso pela coluna *Redução de tempo - Junção Estrela*, e variou de 80,41% a 91,74%. Comparando a Tabela 29 e a Tabela 21 (Seção 5.2.2.2), observa-se que o SB-index NR proporcionou um tempo de resposta menor que o SB-index sem a referida alteração. Como mostra a coluna *Redução de tempo - SB-index*, os ganhos variaram de 15,69% até 73,71%. Por fim, é válido frisar que o SB-index NR

determina uma melhoria na portabilidade do SB-index, tornando viável a sua aplicação em esquemas de DWG redundantes ou não.

Tabela 29 – Medidas referentes ao SB-index NR aplicado ao GRSSB: processamento de consultas.

	Tempo decorrido (s)	Acessos a disco	Redução de tempo - Junção Estrela	Redução de tempo - SB-index
Cidade	229,11	511	91,74%	15,69%
Nação	507,41	57	85,29%	56,93%
Região	1214,93	36	80,41%	73,71%

5.3 Considerações Finais

Embora a literatura mencione diversos trabalhos apresentando modelos conceituais e lógicos para DWG, dentre os trabalhos pesquisados nenhum investigou as seguintes questões sob uma abordagem experimental: **como a redundância de dados espaciais afeta os requisitos de armazenamento e o processamento de consultas SOLAP no DWG?** Justamente estas questões foram tratadas neste capítulo.

Primeiramente, foi proposto um esquema de DWG intrinsecamente redundante, a partir do *Star Schema Benchmark*, denominado GRSSB. Este esquema requer dez vezes mais espaço de armazenamento do que o GHSSB (cuja descrição está no Capítulo 4). Então, foram submetidas ao GRSSB, as mesmas consultas anteriormente submetidas ao esquema GHSSB. Avaliaram-se os recursos atuais presentes nos SGBD para realizar tais consultas SOLAP. Isto é, as configurações C1 e C2, representando o cálculo da junção estrela auxiliada por índices espaciais (R-tree e GiST, respectivamente).

A comparação entre os resultados obtidos por C1 e C2 com o esquema GHSSB e os resultados obtidos pelas mesmas configurações com o esquema GRSSB culminou na seguinte conclusão: a redundância de dados espaciais afeta drasticamente o desempenho do processamento de consultas SOLAP envolvendo os predicados espaciais **intersecta**, **está contido** e **contém**, conduzindo a tempos de resposta demasiadamente altos. Estes tempos superam o do GHSSB de 24% até 815% nos níveis de granularidade altamente redundantes.

O panorama mostrou-se ainda pior ao empregar no GRSSB, duas janelas de consultas espaciais, visto que o tempo de resposta excedeu um dia de execução. É válido enfatizar que, enquanto o esquema GHSSB evita a redundância de dados espaciais e ainda promove o compartilhamento de tabelas de dimensão espaciais, o esquema GRSSB é intrinsecamente redundante. Logo, segundo os testes experimentais realizados, a redundância de dados espaciais deve ser evitada no projeto do DWG.

Os resultados ruins obtidos no processamento de consultas SOLAP sobre o GRSSB usando os recursos do SGBD motivou usar o SB-index para tornar tal processamento mais ágil. Este índice proporcionou ótimos ganhos de desempenho nas consultas SOLAP com os predicados espaciais **intersecta**, **está contido** e **contém**. Os ganhos variaram de 25% a 95%. Além disso, o SB-index teve participação decisiva no teste envolvendo duas janelas de consulta espaciais, proporcionando um ganho de desempenho de 99,82%.

Concluiu-se que, conforme a redundância aumentou, o desempenho do SB-index diminuiu. Os motivos para esta diminuição foram apontados: o mesmo MBR era avaliado repetidas vezes na busca seqüencial sobre o índice, bem como distintos valores de chave primária representando o mesmo objeto espacial eram conduzidos à custosa etapa de refinamento. Tendo identificado estes motivos, uma melhoria foi proposta para atenuar os efeitos da redundância sobre o SB-index.

A melhoria introduzida no SB-index tem por meta manipular apenas MBR distintos, ainda que cada um deles se refira a uma grande quantidade de valores de chave. Este tratamento fez com que MBR repetidos não mais fossem avaliados na busca seqüencial no índice, bem como evitou que objetos espaciais repetidos fossem testados no refinamento. Deste modo, o ganho de desempenho mínimo do SB-index aumentou de 25% para 80%. Por fim, o gráfico da Figura 49 sintetiza, para consultas SOLAP com o predicado espacial **intersecta**, quanto foi o ganho de desempenho do SB-index em relação às configurações C1 e C2, especificamente no esquema GRSSB. Neste gráfico, já estão expressos os ganhos referentes ao melhoramento do SB-index que fora descrito.

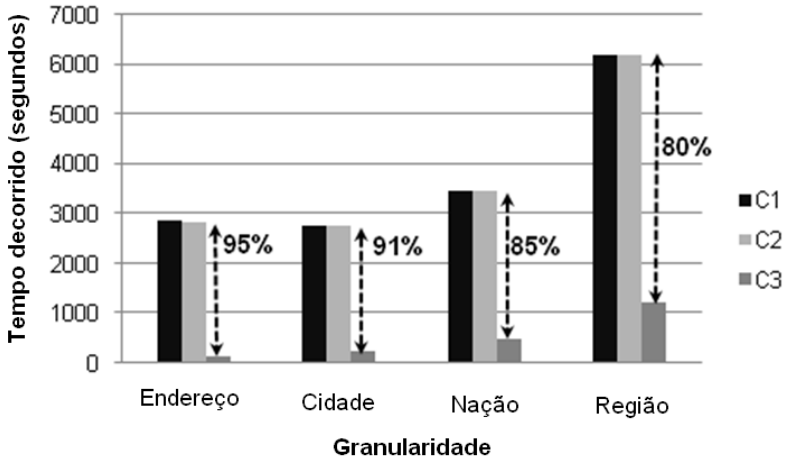


Figura 49 – O SB-index comparado à junção estrela auxiliada por índices espaciais no esquema GRSSB, em consultas SOLAP com o predicado espacial **intersecta**.

6. CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo conclui a dissertação e enumera trabalhos futuros.

6.1 Conclusões

Nesta dissertação foi proposta uma nova estrutura de indexação para *data warehouse* geográfico, denominada SB-index (*Spatial Bitmap Index*). Conforme os experimentos realizados, o SB-index diminui o tempo de resposta de consultas analíticas multidimensionais com os seguintes predicados espaciais: **intersecta**, **está contido** e **contém**.

O *data warehouse* geográfico (DWG) constitui um banco de dados resultante da integração das tecnologias de *data warehouse*, OLAP e sistemas de informações geográficas. Esta integração define um modelo de dados específico para suporte à decisão, flexível para a realização de análises multidimensionais sobre um grande volume de dados, e capaz de armazenar mapas e de permitir consultas espaciais. Em particular, as consultas analíticas multidimensionais com predicado espacial são extremamente custosas, devido à realização de múltiplas junções envolvendo grande quantidade de dados, e à avaliação de relacionamentos espaciais. As junções são originadas pelo esquema lógico do DWG, que se assemelha ao esquema estrela do DW convencional. Os relacionamentos espaciais são operações binárias, que comparam um objeto espacial presente em uma tabela com uma janela de consulta espacial arbitrária. Esta janela não consta previamente na base de dados.

Desta forma, o SB-index foi proposto com o intuito de diminuir o tempo de resposta de consultas assim caracterizadas, e ainda:

a) introduzir o Índice Bitmap no contexto do DWG; e
b) tratar hierarquias de atributos espaciais predefinidas.

Particularmente, a escolha pelo Índice Bitmap foi baseada na sua eficiência atuando como Índice Bitmap de Junção. O Índice Bitmap é caracterizado pela manipulação de vetores de bits por meio de operações lógicas que são auxiliadas pelo *hardware*. Já o Índice Bitmap de Junção possui estas características e uma característica adicional fundamental para o DWG: evita a custosa operação de junção entre tabelas. Escolher o Índice Bitmap exige ponderar as desvantagens causadas por atributos indexados que possuem domínios vastos, ou seja, os prejuízos causados por atributos com alta cardinalidade. Porém, a existência de técnicas de *binning*, codificação e compressão asseguram que estes prejuízos podem ser atenuados.

Há de se ressaltar que as hierarquias de atributos predefinidas são comuns em muitas aplicações de DWG, por exemplo: região $\prec$ nação $\prec$ cidade $\prec$ endereço. Estas

hierarquias estabelecem relacionamentos entre diferentes níveis de granularidade espacial. Eles determinam, por exemplo, que os objetos espaciais de níveis mais altos de granularidade contêm os de níveis inferiores. Esta abordagem seguida pelo SB-index difere da abordagem da aR-tree (PAPADIAS et al., 2001), que havia sido identificada na literatura como a única estrutura de indexação específica para DWG. Isto é, a aR-tree lida com hierarquias de objetos espaciais criadas *ad-hoc* com o auxílio do método de particionamento de espaço da R-tree.

A estrutura de dados compacta do SB-index prevê a criação de um arquivo seqüencial cujas entradas possuem um valor de chave primária e dois pares de coordenadas. A chave primária identifica o objeto espacial, enquanto os dois pares de coordenadas retratam o MBR do mesmo. A existência da hierarquia espacial determina a criação de um índice por atributo espacial da hierarquia. Embora requeira um percurso seqüencial pela estrutura de indexação, o processamento de consultas do SB-index possui duas virtudes críticas no DWG:

a) filtra os objetos espaciais;
b) evita a computação da junção-estrela.

O filtro consiste em testar o relacionamento existente entre a janela de consulta e o MBR de uma entrada, e auxilia o cálculo do predicado espacial. Ele seleciona candidatos à resposta deste predicado, sendo que os candidatos representam apenas uma pequena parcela do total de objetos espaciais existentes na tabela. Posteriormente, os candidatos são conduzidos a uma etapa de refinamento realizada no banco de dados. Esta etapa é mais custosa que o filtro, porém lida com muito menos objetos que a filtragem. Ao término do refinamento, obtém-se o conjunto resposta do predicado espacial. Este conjunto é manipulado de forma a compor um predicado convencional, o qual pode ser computado em conjunto com os outros predicados convencionais da consulta. Esta estratégia permite que a resposta completa da consulta seja obtida por meio de um Índice Bitmap de Junção. A implementação de Índice Bitmap empregada foi a do software FastBit. A transformação do predicado espacial em um predicado convencional é um aspecto muito relevante no processamento de consultas do SB-index.

A proposta do SB-index foi validada por meio de testes de desempenho experimentais. Eles comprovam a eficiência deste índice tanto em termos de armazenamento quanto de processamento de consultas. Os testes confrontaram os resultados do novo índice com os resultados obtidos pelo uso da tecnologia disponível nos sistemas gerenciadores de banco de dados, ou seja, o cômputo da junção estrela auxiliado por índices sobre os atributos espaciais (R-tree e GiST).

A análise dos resultados revelou que o tempo de resposta das consultas SOLAP foi drasticamente reduzido com o uso do SB-index. A Tabela 30 resume os expressivos valores de ganho de desempenho alcançados pelo SB-index por nível de granularidade e por predicado espacial. Há de se ressaltar que estes resultados foram alcançados em um esquema de DWG que evita a redundância de dados espaciais e compartilha tabelas de dimensão espaciais quando possível: o GHSSB (*Geographic Hybrid Star Schema Benchmark*).

Tabela 30 – Resumo dos ganhos de desempenho do SB-index no esquema GHSSB, por nível de granularidade e por predicado espacial.

	Predicado espacial		
Granularidade	**intersecta**	**está contido**	**contém**
Endereço	95,38%	96,07%	97,13%
Cidade	94,56%	94,81%	94,91%
Nação	92,70%	90,44%	94,38%
Região	90,38%	90,60%	90,61%

Os resultados retratados pela Tabela 30, sempre acima de 90%, demonstram que o desempenho do SB-index permaneceu muito bom, sendo o predicado espacial mais ou menos restritivo. Em seguida, foi testada uma consulta SOLAP com duas janelas de consulta espaciais. Ela consiste em "obter o total das receitas ganhas pelos fornecedores localizados em cidades que intersectam uma região arbitrária, comercializando produtos de uma determinada marca a consumidores localizados em cidades intersectadas por outra região arbitrária, agrupando os resultados por ano e por marca". Novamente, o SB-index proporcionou um ganho de desempenho excelente: 76%. Por fim, foi realizada uma análise sobre o volume de dados do DWG. Mantendo fixa a quantidade de objetos espaciais e variando a porção de dados convencionais, foram gerados outros dois DWG, com $^{1}/_{5}$ e $^{3}/_{5}$ do volume inicial de dados. Este fator não prejudicou o desempenho do SB-index, visto que os ganhos de desempenho variaram de 89% a 99%.

Terminada esta etapa de validação e de verificação do desempenho, realizou-se uma investigação sobre o impacto da redundância de dados espaciais sobre o DWG. Para tanto, criou-se um esquema de DWG intrinsecamente redundante denominado GRSSB (*Geographic Redundant Star Schema Benchmark*). Este esquema requereu 10 vezes mais espaço de armazenamento que o GHSSB. Logo, existe um sério impacto negativo da redundância de dados espaciais sobre os recursos de armazenamento do DWG. Era preciso explorar se no DWG, a exemplo do DW, seria viável priorizar baixo tempo de resposta no processamento de consultas em detrimento dos recursos de armazenamento.

Inicialmente, usou-se a junção estrela auxiliada por índices espaciais (R-tree e GiST). O confronto dos resultados do GRSSB com aqueles previamente obtidos no GHSSB revelou que a redundância de dados espaciais causou severos prejuízos ao desempenho do processamento de consultas. Na Tabela 31 são resumidas as perdas de desempenho proporcionadas pelo GRSSB em relação ao GHSSB, segundo diferentes níveis de granularidades e distintos predicados espaciais.

Tabela 31 – Resumo dos atrasos causados pela redundância de dados espaciais no DWG, usando a junção estrela auxiliada por eficientes índices espaciais.

	Predicado espacial		
Granularidade	**intersecta**	**está contido**	**contém**
Cidade	0,52%	1%	0,89%
Nação	24,74%	138,56%	29,34%
Região	122,21%	815,96%	116,03%

Foi executada a consulta SOLAP com duas janelas de consulta espaciais. Esta execução precisou ser interrompida após o processamento atingir o inadmissível tempo decorrido de 48h. Os tempos de resposta proibitivos observados no processamento de consultas do GRSSB, usando a junção estrela auxiliada por índices espaciais, motivaram a aplicação do SB-index neste esquema. A Tabela 32 resume os significativos valores de ganho de desempenho por nível de granularidade e por predicado espacial alcançados pelo SB-index no GRSSB. Os ganhos variaram de 25% a 97%.

Tabela 32 – Resumo dos ganhos de desempenho do SB-index no esquema GRSSB, por nível de granularidade e por predicado espacial.

	Predicado espacial		
Granularidade	**Intersecta**	**está contido**	**contém**
Endereço	95,32%	94,77%	97,09%
Cidade	90,20%	94,12%	94,39%
Nação	65,84%	78,39%	72,16%
Região	25,45%	85,61%	38,53%

Assim como já observado no caso da junção estrela, a redundância de dados espaciais degenerou ainda o desempenho do processamento de consultas do SB-index no GRSSB. Isto ocorreu principalmente nos níveis de granularidade com maior incidência de redundância (Nação e Região). Existem duas razões primordiais para isso. A primeira reside no fato de que, durante a busca seqüencial sobre o SB-index, testa-se o mesmo MBR diversas vezes em relação ao predicado espacial. Isto determina que chaves distintas referentes a objetos espaciais idênticos são conduzidas ao refinamento. Eis a segunda razão: o

refinamento, que é a etapa mais custosa do processamento, é realizado sobre objetos espaciais idênticas, gerando um significativo atraso no tempo de resposta.

Identificando estas duas razões, foi possível propor e desenvolver uma melhoria sobre o SB-index, especificamente para lidar com a redundância de dados espaciais. Este aperfeiçoamento determina um novo nível no SB-index, de forma que o índice passa a armazenar apenas MBR distintos. Todas as chaves que se associam a este MBR são armazenadas a parte, em uma lista. De posse desta melhoria, os experimentos foram repetidos para o predicado espacial **intersecta**, obtendo-se o expressivo ganho de desempenho mínimo de 80% no nível de granularidade Região.

De posse dos mencionados resultados, é razoável afirmar que esta dissertação atingiu o seu objetivo, propondo uma nova e eficiente estrutura de indexação para DWG: o SB-index. Além disso, o SB-index tem plenas condições de ser empregado em esquemas de DWG redundantes ou não-redundantes, apresentando assim uma boa portabilidade. Além de contribuir com a proposição do SB-index, este trabalho também oferece uma investigação sobre os efeitos da redundância de dados espaciais sobre o DWG. Os resultados desta investigação, que foi realizada por meio de testes experimentais, demonstraram que a redundância de dados espaciais deve ser evitada sempre que possível no projeto de DWG.

6.2 Trabalhos em andamento e futuros

Nas subseções seguintes são descritos trabalhos já em andamento e trabalhos futuros. Além deles, existem planos para rodar novos experimentos usando consultas SOLAP diferentes daquelas já abordadas, bem como utilizar outros sistemas gerenciadores de banco de dados. Ainda, um outro trabalho consiste em desenvolver algoritmos para promover operações de atualização sobre o SB-index, considerando a existência de domínios cujos objetos espaciais necessitam de modificações freqüentes em suas geometrias.

6.2.1 Etapa intermediária no processamento de consultas

A proposta atual do processamento de consultas usando o SB-index consiste em duas fases: filtragem e refinamento. Uma fase intermediária usando uma aproximação de objeto espacial mais exata poderia ser incluída, a fim melhorar o desempenho do índice. Tal aproximação reduziria ainda mais a quantidade de objetos espaciais que devem ser analisados na fase de refinamento, bem como é menos custosa que a avaliação dos objetos espaciais originais. Um trabalho futuro consiste em usar o *convex hull* ou o 5C (BRINKHOFF; KRIEGEL; SCHNEIDER, 1993) como aproximações nesta fase intermediária.

6.2.2 Investigações sobre o uso do Índice Bitmap

Um trabalho futuro consiste em investigar como as diferentes combinações de técnicas de *binning*, compressão e codificação podem melhorar o processamento de consultas SOLAP usando o SB-index.

Com relação ao *binning*, levando em consideração que um objeto espacial freqüentemente é aproximado por um retângulo n-dimensional (MBR), convém perguntar: é viável encaixotar diversos objetos espaciais segundo um relacionamento topológico? O pré-armazenamento de relacionamentos topológicos não é viável em bancos de dados espaciais, uma vez que os dados são altamente dinâmicos. Porém, como um DW armazena dados consolidados que permanecem estáticos por um longo intervalo de tempo, esta questão não é respondida de forma direta, ao contrário do que acontece para uma base OLTP.

Sabendo que existem métodos de codificação eficientes para domínios onde existe uma relação de ordem total (O'NEIL,P.; QUASS, 1997; WU, M.; BUCHMANN, 1998;), uma questão sobre a codificação envolve o DWG: como criar uma codificação que favoreça os relacionamentos topológicos entre objetos espaciais presentes em uma tabela de dimensão?

Por fim, originalmente, o projeto do FastBit objetivou a criação de uma estrutura seqüencial (Seção 3.3) para beneficiar especialmente as consultas por intervalos de valores (FASTBIT, 2007). Isto porque um percurso seqüencial em ***F*** (Seção 3.3) seria capaz de realizar todas as operações lógicas capazes de responder a consulta. Por outro lado, o SB-index não emprega esta classe de consultas, como é possível observar na transformação do predicado espacial em predicado convencional, na Seção 4.3. Ao invés disso, o SB-index solicita consultas por valor exato. Por exemplo, ao invés de compor um predicado *WHERE PK > 1 AND PK <= 5*, é composto um predicado *WHERE PK = 1 OR PK = 2 OR PK = 3 OR PK = 4 OR PK = 5*. Um trabalho futuro poderia investigar se a submissão de consultas por valor exato possui algum efeito negativo no tempo de resposta da consulta, usando atributos com alta e baixa cardinalidade.

6.2.3 Alteração da estrutura seqüencial

O SB-index introduziu o Índice Bitmap no DWG, como também levantou algumas questões:

a) uma vez que o índice é um arquivo seqüencial, ele requer uma busca seqüencial para testar o relacionamento espacial entre o MBR de cada entrada e a janela de consulta;

b) toda consulta em um determinado nível de granularidade requer um número fixo de acessos a disco, pois a busca seqüencial deve visitar todas as páginas de disco. Conseqüentemente, toda consulta tem uma complexidade linear: O(n);
c) não existe um método eficiente que permita reusar algumas das entradas do SB-index que haviam sido recuperadas em consultas anteriores, tal como um *buffer-pool*, a fim de evitar novos acessos a disco; e
d) o SB-index não é capaz de agrupar objetos espaciais. Logo, por exemplo, mesmo que a janela de consulta esteja localizada no oeste, as entradas que representam objetos espaciais das regiões oeste, leste, norte e sul precisam ser avaliadas segundo o predicado espacial. Idealmente, apenas as entradas da região oeste e de suas redondezas é que deveriam ser avaliadas.

Estas questões motivam a alteração da estrutura de dados do SB-index, a fim de refiná-lo para melhorar o seu desempenho no processamento de consultas. De fato, está sendo desenvolvida uma nova estrutura de indexação denominada SBR-tree (R-tree com Bitmap espacial). Ao invés de uma estrutura seqüencial, a SBR-tree é baseada na R*-tree, a qual proporciona menos acessos a disco durante o processamento de uma consulta, devido ao agrupamento de objetos espaciais e à leitura de páginas de disco específicas.

Alguns resultados preliminares deste trabalho demonstraram que há uma redução na etapa de busca sobre o índice, quando comparado com a busca seqüencial realizada pelo SB-index.

6.2.4 Benchmarks para *data warehouse* geográfico

A validação do SB-index foi realizada com testes de desempenho sobre os esquemas de DWG não-redundante e redundante: o GHSSB e o GRSSB, respectivamente. Em particular, não foi encontrado na literatura um *benchmark* específico para DWG. A falta de um *benchmark* com tais características motivou a adaptação do *Star Schema Benchmark* (SSB) para armazenar dados geográficos. A criação destes dois esquemas, por si, consistiu na construção de um novo *benchmark* específico para DWG. Está em andamento um trabalho de formalização e especificação das consultas do GHSSB e do GRSSB, a fim de que ambos estabeleçam um novo *benchmark* que possa ser empregado na avaliação e na validação de trabalhos acerca de DWG.

REFERÊNCIAS

ADZIC, J.; FIORE, V.; SISTO, L. Extraction, Transformation, and Loading Processes. In: W. R. e C. Koncilia (Ed.). **Data Warehouses and OLAP: Concepts, Architectures and Solutions**. Hershey, Pennsylvania, USA: IRM Press, 2007. Chapter 4, p.88-110.

ANTOSHENKOV, G. Byte-aligned bitmap compression. In: CONFERENCE ON DATA COMPRESSION, 1995, Snowbird. **Proceedings…** Washington: IEEE Computer Society, 1995. p.476.

ARIGON, A.-M.; TCHOUNIKINE, A., MIQUEL, M. Handling multiple points of view in a multimedia data warehouse. **ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)**, v.2, n.3, p.199-218, 2006.

BECKMANN, N. et al. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1990, Atlantic City. **Proceedings…** New York: ACM, 1990. p. 322-331.

BÉDARD, Y.; RIVEST, S.; PROULX, M.-J. Spatial Online Analytical Processing (SOLAP): Concepts, Architectures and Solutions from a Geomatics Engineering Perspective. In: W. R. e C. Koncilia (Ed.). **Data Warehouses and OLAP: Concepts, Architectures and Solutions**. Hershey, Pennsylvania, USA: IRM Press, 2007. Chapter 13, p.298-319.

BIMONTE, S. et al. GeWOlap: A Web Based Spatial OLAP Proposal. In: ON THE MOVE TO MEANINGFUL INTERNET SYSTEMS 2006: OTM 2006 WORKSHOPS, 2006, Montpellier. **Proceedings…** Berlin / Heidelberg: Springer, 2006. p. 1596-1605.

BIMONTE, S.; TCHOUNIKINE, A., MIQUEL, M. Towards a Spatial Multidimensional Model. In: ACM INTERNATIONAL WORKSHOP ON DATA WAREHOUSING AND OLAP, 8., 2005, Bremen. **Proceedings…** New York: ACM, 2005. p. 39 - 46 .

_____. GeoCube, a multidimensional model and navigation operators handling complex measures: application in spatial OLAP. In: INTERNATIONAL CONFERENCE ON ADVANCES IN INFORMATION SYSTEMS, 4., 2006, Izmir. **Proceedings…** Berlin / Heidelberg: Springer, 2006. p.100-109.

BRINKHOFF, T.; KRIEGEL, H-S; SEEGER, B. Efficient processing of spatial joins using R-trees. **ACM SIGMOD Record**, v.22, n.2, p.237-246, June, 1993.

BRINKHOFF, T.; KRIEGEL, H. P.; SCHNEIDER, R.. Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Database Systems. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 9., 1993, Vienna. **Proceedings…** Washington: IEEE Computer Society, 1993. p. 40-49.

BRINKHOFF, T.; KRIEGEL, H. P.; SCHNEIDER, R.; SEEGER, B. Multi-step Processing of Spatial Joins. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1994, Minneapolis. **Proceedings…** New York: ACM, 1994. p.197-208.

CÂMARA, G.; CASANOVA, M.A.; HEMERLY, A.S.; MAGALHÃES, G.C.; MEDEIROS, C.M.B. **Anatomia de Sistemas de Informação Geográfica**. Instituto de Computação. UNICAMP, Campinas: 193 p., 1996.

CASANOVA, M. et al. (Ed.). **Bancos de dados geográficos.** 2. ed. Curitiba: MundoGEO, 2005.

CHAN, C.-Y;. IOANNIDIS, Y. E. An efficient bitmap encoding scheme for selection queries. **ACM SIGMOD Record**, v.28, n.2, p.215-226 June, 1999.

CHAUDHURI, S.; DAYAL, U. An Overview of Data Warehousing and OLAP Technology. **ACM SIGMOD Record**, v.26, n.1, p.65-74, March, 1997.

CIFERRI, C. D. A. **Distribuição dos dados em ambientes de data warehousing**: o sistema WebD2W e algoritmos voltados à fragmentação horizontal dos Dados. 2002. 263f. Tese (Doutorado em Ciência da Computação). Centro de Informática, Universidade Federal de Pernambuco, Recife, PE, Brasil, 2002.

CIFERRI, C.D.A.; CIFERRI, R.R.; FORLANI, D.T.; TRAINA, A.J.M.; SOUZA, F.F. Horizontal Fragmentation as a Technique to Improve the Performance of Drill-Down and Roll-Up Queries. In: SYMPOSIUM ON APPLIED COMPUTING, 22., 2007, Seoul. **Proceedings...** New York: ACM, 2007. p. 494-499.

CIFERRI, R. R. **Análise da influência do fator distribuição espacial dos dados no desempenho de métodos de acesso multidimensionais**. 2002. 246 f. Tese (Doutorado em Ciência da Computação). Centro de Informática, Universidade Federal de Pernambuco, Recife, PE, Brasil, 2002.

COSTA, M.; MADEIRA, H. Handling big dimensions in distributed data warehouses using the DWS technique. INTERNATIONAL WORKSHOP ON DATA WAREHOUSING AND OLAP, 7., 2004, Washington. **Proceedings...** New York: ACM, 2004. p. 31-37, 2004.

DATTA, A.; MOON, B.; THOMAS, H. A case for parallelism in data warehousing and OLAP. In: INTERNATIONAL WORKSHOP ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, 9., 1998, Vienna. **Proceedings...** London, Springer-Verlag, 1998. p. 226-231.

ELMASRI, R. E.; NAVATHE, S. **Sistemas de Banco de Dados**. Addison-Wesley, 2005. 744p.

FASTBIT an efficient compressed bitmap index technology. Disponível em: <http://sdm.lbl.gov/fastbit/>. Acesso em: dez. 2007.

FERRARI, R. **Viagem ao SIG On Line:** Planejamento Estratégico, Viabilização, Implantação e Gerenciamento de Sistemas de Informação Geográfica. EdUFSCar. São Carlos, 1997. Disponível em: <http://www2.dc.ufscar.br/~ferrari/viagem/inicial.html>. Acesso em: mar 2008.

FIDALGO, R. N. **Uma Infra-estrutura para Integração de Modelos, Esquemas e Serviços Multidimensionais e Geográficos**. 2005. 165 f. Tese (Doutorado em Ciência da Computação). Centro de Informática, Universidade Federal de Pernambuco, Recife, PE, Brasil, 2005.

FIDALGO, R. N. et al. GeoDWFrame: a framework for guiding the design of geographical dimensional schemas. In: CONFERENCE ON DATA WAREHOUSING AND KNOWLEDGE DISCOVERY, 6., 2004, Zaragoza. **Lecture Notes in Computer Science.** v. 3181/2004. Berlin / Heidelberg: Springer, 2004, p.26-37.

FIDALGO, R. N.; TIMES, V. C.; DE SOUZA, F. F. GOLAPA: Uma Arquitetura Aberta e Extensível para Integração entre SIG e OLAP. In: SIMPÓSIO BRASILEIRO DE GEOINFORMÁTICA, 3., 2001, Campos do Jordão. **Anais...** Disponível em <http://www.geoinfo.info/geoinfo2001/papers/149robson.pdf>. Acesso em: fev. 2008.

_____. Providing Multidimensional and Geographical Integration Based on a GDW and Metamodels. In: SIMPÓSIO BRASILEIRO DE BANCOS DE DADOS, 19., 2004, Brasília. **Anais...** Porto Alegre: SBC, 2004. p.148-162.

FOLK, M. J.; ZOELLICK, B. **File Structures**. USA: Addison-Wesley. 1992. 590 p.

FURTADO, P. Experimental evidence on partitioning in parallel data warehouses. In: INTERNATIONAL WORKSHOP ON DATA WAREHOUSING AND OLAP, 7., 2004, Washington. **Proceedings…** New York**:** ACM, 2004. p. 23-30.

GAEDE, V.; GÜNTHER, O. Multidimensional Access Methods. **ACM Computing Surveys**, v.30, n.2, p.170-231, June, 1998.

GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOW, J. **Database System Implementation**. Upper Saddle River, New Jersey, USA: Prentice Hall, 2000. 653 p.

GIST Indexing Project. Disponível em: <http://gist.cs.berkeley.edu>. Acesso em: Jul 2008.

GOLFARELLI, M., MAIO, D., RIZZI, S. Applying Vertical Fragmentation Techniques in Logical Design of Multidimensional Databases. IIn: INTERNATIONAL CONFERENCE ON DATA WAREHOUSING AND KNOWLEDGE DISCOVERY, 2., 2000, London. **Proceedings…** London: Springer-Verlag, 2000. p. 11-23.

GOLFARELLI, M., MANIEZZO, V., RIZZI, S. Materialization of fragmented views in multidimensional databases. **Data & Knowledge Engineering**, v.49, n.3, p.325–351, 2004.

GOLFARELLI, M.; RIZZI, S.; IURIS, C. Beyond data warehousing: what's next in business intelligence? ACM INTERNATIONAL WORKSHOP ON DATA WAREHOUSING AND OLAP, 7., 2004, Washington. **Proceedings...** New York: ACM, 2004. p.1-6.

GORAWSKI, M.; CHECHELSKI, R. Online Balancing of aR-Tree Indexed Distributed Spatial Data Warehouse. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING AND APPLIED MATHEMATICS, 6., 2005, Poznan. **Lecture Notes in Computer Science.** v.3911. Berlin / Heidelberg: Springer, 2006. p.59-66.

GOYAL, N.; ZAVERI, S. K.; SHARMA, Y. Improved Bitmap Indexing Strategy for Data Warehouses. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY, 9., 2006, Bhubaneswar. **Proceedings…** Washington: IEEE Computer Society, 2006. p.213-216.

GUTTMAN, A. R-trees: a dynamic index structure for spatial searching. **ACM SIGMOD Record**, v.14, n.2, p.47 – 57, 1984.

HARINARAYAN, V.; RAJARAMAN, A.; ULLMAN, J. D. Implementing data cubes efficiently. **ACM SIGMOD Record**, v.25, n.2, p.205-216, 1996.

HELLERSTEIN, J. M.; NAUGHTON, J. F.; PFEFFER, A. Generalized search trees for database systems. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 21., 1995, Zurich. **Proceedings…** San Francisco: Morgan Kauffman, 1995. p. 562–573.

HYDE, J. Layers of a Mondrian System. Disponível em: <http://mondrian.pentaho.org/documentation/architecture.php>. Acesso em: jan 2008.

INMON, W. H. **Building the data warehouse**. 3rd. New York: Wiley Computer Publishing, 2002. 412 p.

JOHNSON, T. Performance Measurements of Compressed Bitmap Indices. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 25., 1999, Edinburgh. **Proceedings…** San Francisco: Morgan Kauffman, 1999. p. 278-289.

JOHNSON, T.; SHASHA, D. Some Approaches to Index Design for Cube Forests. **IEEE Data Engineering Bulletin**, v.20, n.1, p.27-35, 1997.

JÜRGENS, M.; LENZ, H.-J. Tree Based Indexes vs. Bitmap Indexes - a Performance Study. In: INTERNATIONAL WORKSHOP ON DESIGN AND MANAGEMENT OF DATA WAREHOUSES, 1., 1999, Heidelberg. **Proceedings…** Heildelberg: CEUR, 1999. p.1-10.

KAMEL, I.; FALOUTSOS, C. Parallel R-trees. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1992, San Diego. **Proceedings…** New York: ACM, 1992. p.195–204.

KAMEL, I.; FALOUTSOS, C. Hilbert R-tree: An improved R-tree using fractals. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 20., 1994, Santiago de Chile. **Proceedings…** San Francisco: Morgan Kaufmann, 1994. p.500-509.

KIMBALL, R.; ROSS, M. **The Data Warehouse Toolkit**. 2nd. New York: Wiley Computer Publishing, 2002.

KIMBALL, R.; CASERTA, J. **The Data Warehouse ETL Toolkit.** 1ST. Wiley, 2004.

KNUTH, D. Searching. In: KNUTH, D. **The art of computer programming**. 2nd. Berkeley: Addison-Wesley Professional, 1998. v. 3, c. 6, p. 560-563.

MALINOWSKI, E.; ZIMÁNYI, E. Representing spatiality in a conceptual multidimensional model. In: ANNUAL ACM INTERNATIONAL WORKSHOP ON GEOGRAPHIC INFORMATION SYSTEMS, 12., 2004, Washington. **Proceedings…** New York: ACM, 2004. p.12-22.

MALINOWSKI, E.; ZIMÁNYI, E. Spatial Hierarchies and Topological Relationships in the Spatial MultiDimER Model. In: BRITISH NATIONAL CONFERENCE ON DATABASES, 2005. **Lecture Notes in Computer Science.** v.3567. Berlin/Heidelberg: Springer, 2005. p. 17-28.

MALINOWSKI, E.; ZIMÁNYI, E. **Advanced Data Warehouse Design**: From Conventional to Spatial and Temporal Applications. Springer, 1st edition, 2008.

MAPINFO Professional User Guide. New York, 2005. Disponível em: <http://reference.mapinfo.com/software/mapinfo_pro/english/8.0/MI_UG.pdf >. Acesso em: out 2007.

MONDRIAN Project. Disponível em: <http://mondrian.pentaho.org>. Acesso em: jan. 2008.

OPENGIS standards and specifications. Disponível em: <http://www.opengeospatial.org/standards>. Acesso em: out. 2007.

OLAP COUNCIL: OLAP AND OLAP Server Definitions. Disponível em: <http://www.olapcouncil.org/research/glossaryly.htm>. Acesso em: jan 2008.

O'NEIL, E.; O'NEIL, P.; WU, K. Bitmap Index Design Choices and Their Performance Implications. In: INTERNATIONAL DATABASE ENGINEERING AND APPLICATIONS SYMPOSIUM, 11.2007, Banff. **Proceedings…** Washington: IEEE Computer Society**,** 2007. p. 72-84.

O'NEIL, P.; O'NEIL, E.; CHEN, X. The Star Schema Benchmark. Disponível em: <http://www.cs.umb.edu/~poneil/ starschemab.pdf>. Acesso em: jan. 2007.

O'NEIL, P.; GRAEFE, G. Multi-table joins through bitmapped join indices. **ACM SIGMOD Record**, v.24, n.3, p.8-11, 1995.

O'NEIL, P.; QUASS, D. Improved query performance with variant indexes. ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1997, Tucson. **Proceedings…** New York: ACM, 1997. p.38-49.

PAPADIAS, D.; KALNIS, P.; ZHANG, J.; TAO, Y. Efficient OLAP Operations in Spatial Data Warehouses. INTERNATIONAL SYMPOSIUM ON SPATIAL AND TEMPORAL DATABASES, 7., 2001, Redondo Beach. **Lecture Notes in Computer Science.** v.2121. London: Springer-Verlag, 2001. p.443-459.

PAPADIAS, D.; TAO, Y; KALNIS, P.; ZHANG, J. Indexing Spatio-Temporal Data Warehouses. IEEE INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 18., 2002, San Jose. **Proceedings…** Washington, USA: IEEE Computer Society, 2002. p.166-175.

POESS, M.; FLOYD, C. New TPC Benchmarks for Decision Support and Web Commerce. **ACM SIGMOD Record**, v.29, n.4, p.64-71, 2000.

POURABBAS, E.; RAFANELLI, M. Characterization of hierarchies and some operators in OLAP environment. In: ACM INTERNATIONAL WORKSHOP ON DATA WAREHOUSING AND OLAP, 2., 1999, Kansas City. **Proceedings…** New York: ACM, 1999. p. 54-59.

RAMAKRISHNAN, R.; GEHRKE, J. **Database Management Systems**. McGraw-Hill, 2002. 1104 p.

RAO, F.; ZHANG, L.; YU, X.; LI, Y. Spatial hierarchy and OLAP-favored search in spatial data warehouse. In: INTERNATIONAL WORKSHOP ON DATA WAREHOUSING AND OLAP, 6., 2003, New Orleans. **Proceedings…** New York:ACM, 2003. p. 48-55.

RIGAUX, P.; SCHOLL, M.; VOISARD, A.; **Spatial Databases with Application to GIS.** San Francisco, CA, USA: Morgan Kauffman, 2002.

RIZZI, S. Conceptual Modeling Solutions for the Data Warehouse. In: W. R. e C. Koncilia (Ed.). **Data Warehouses and OLAP: Concepts, Architectures and Solutions**. Hershey, Pennsylvania, USA: IRM Press, 2007. Chapter 1, p.1-26.

RIZZI, S., ABELLÓ, A. Research in data warehouse modeling and design: dead or alive? In: ACM INTERNATIONAL WORKSHOP ON DATA WAREHOUSING AND OLAP, 9., 2006, Arlington. **Proceedings…** New York: ACM, 2006. p.3-10.

SAMPAIO, M. C.; SOUSA, A. G.; BAPTISTA, C. S. Towards a Logical Multidimensional Model for Spatial Data Warehousing and OLAP. In: INTERNATIONAL WORKSHOP ON DATA WAREHOUSING AND OLAP, 9., 2006, Arlington. **Proceedings…** New York: ACM, 2006. p.83-90.

SARAWAGI, S. Indexing OLAP Data. **Bulletin of the Technical Committee on Data Engineering**, v.20, n.1, p.36-43, 1997.

SAS OLAP Server. Disponível em: <http://www.sas.com/resources/factsheet/sas-olap-server-factsheet.pdf>. Acesso em: jan 2008.

SELLIS, T. K.; ROUSSOPOULOS, N.; FALOUTSOS, C. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 13.,1987, Brighton. **Proceedings...** San Francisco: Morgan Kaufmann, 1987. p. 507-518

SHAPEFILE technical description. New York: ESRI White Paper, 1998. Disponível em: <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>. Acesso em: out. 2007.

SILVA, J.; TIMES, V. C.; SALGADO, A. C.; SOUZA, C.; FIDALGO, R. N.; OLIVEIRA, A.G. A Set of Aggregation Functions for Spatial Measures. In: International Workshop on Data Warehousing and OLAP, 11., 2008, Napa Valley. **Proceedings…** New York: ACM, 2008. p. 25-32.

SIQUEIRA, T. L. L.; CIFERRI, C. D. A.; TIMES, V.C.; OLIVEIRA, A. G.; CIFERRI, R. R. The Impact of Spatial Data Redundancy on SOLAP Query Performance. **Journal of the Brazilian Computer Society**. v.15, n.2, p.19-34, 2009b. ISSN 0104-6500.

SIQUEIRA, T. L. L.; CIFERRI, R.R.; TIMES, V.C.; CIFERRI, C.D.A.. A Spatial Bitmap-Based Index for Geographical Data Warehouses. ACM SYMPOSIUM ON APPLIED COMPUTING, 24., 2009a, Honolulu. **Proceedings…** v. 3. New York: ACM, 2009a. p. 1336-1342.

SIQUEIRA, T. L. L.; CIFERRI, R.R.; TIMES, V.C.; CIFERRI, C.D.A. Investigating the Effects of Spatial Data Redundancy in Query Performance over Geographical Data Warehouses. BRAZILIAN SYMPOSIUM ON GEOINFORMATICS, 10., 2008, Rio de Janeiro. **Proceedings…** Disponível em: <http://www.geoinfo.info/geoinfo2008/papers/p27.pdf>. Acesso em: jan. 2009.

STEFANOVIC, N. **Design and Implementation of On-Line Analytical Processing (OLAP) of Spatial Data**. 108 f. Dissertation (Master of Science). Department of Computing Science, Simon Fraser University, 1997.

STEFANOVIC, N.; HAN, J.; KOPERSKI, K. Object-Based Selective Materialization for Efficient Implementation of Spatial Data Cubes. **IEEE Transactions on Knowledge and Data Engineering**, v.12, n.6, p.938-958, 2000.

STOCKINGER, K.; WU, K. Bitmap Indices for Data Warehouses. In: W. R. e C. Koncilia (Ed.). **Data Warehouses and OLAP: Concepts, Architectures and Solutions**. Hershey, Pennsylvania, USA: IRM Press, 2007. Chapter 6, p.157-178.

U.S. CENSUS BUREAU. TIGER: Topologically Integrated Geographic Encoding and Referencing system. Disponível em: <http://www.census.gov/geo/www/tiger>. Acesso em: Dez 2007.

WHITEHORN, M.; ZARE, R.; PASUMANSKY, M. **Fast Track to MDX**. Springer, 2005.

WU, M.-C.; BUCHMANN, A. P. Research Issues in Data Warehousing. In: DATENBANKSYSTEME IN BURO, TECHNIK UND WISSENSCHAFT, 1997, Freiburg. **Proceedings…** Freiburg: Springer, 1997. p.61-82.

WU, M.-C.; BUCHMANN, A. P. Encoded Bitmap Indexing for Data Warehouses. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 14., 1998, Orlando. **Proceedings…** Washington: IEEE Computer Society, 1998. p.220-230.

WU, K.; OTOO, E. J.; ARIE, S. Optimizing bitmap indices with efficient compression. **ACM Transactions on Database Systems**, v.31, n.1, p.1-38, 2006.

WU, K.; STOCKINGER, K.; SHOSHANI, A. Breaking the Curse of Cardinality on Bitmap Indexes. INTERNATIONAL CONFERENCE ON SCIENTIFIC AND STATISTICAL DATABASE MANAGEMENT, 20., 2008, Hong Kong. **Proceedings...** Berlin/Heidelberg: Springer-Verlag, 2008. p.348-365.