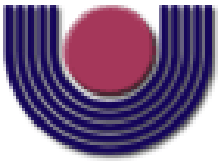

UNIOESTE – Universidade Estadual do Oeste do Paraná

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Colegiado de Ciência da Computação

Curso de Bacharelado em Ciência da Computação

Uma Avaliação Experimental de Desempenho entre Sistemas Gerenciadores de Bancos de Dados Colunares e Relacionais

Bruno Eduardo Soares

CASCAVEL

2012

BRUNO EDUARDO SOARES

UMA AVALIAÇÃO EXPERIMENTAL DE DESEMPENHO ENTRE SISTEMAS GERENCIADORES DE BANCOS DE DADOS COLUNARES E RELACIONAIS

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Dr. Clodis Boscarioli

CASCAVEL

2012

BRUNO EDUARDO SOARES

UMA AVALIAÇÃO EXPERIMENTAL DE DESEMPENHO ENTRE SISTEMAS GERENCIADORES DE BANCOS DE DADOS COLUNARES E RELACIONAIS

Monografia apresentada como requisito parcial para obtenção do Título de *Bacharel em Ciência da Computação*, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Dr. Clodis Boscarioli (Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Dr. Marcio Seiji Oyamada
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Esp. Gustavo Rezende Krüger
Colegiado de Tecnologia em Análise e Desenvolvimento de Sistemas,
UNIVEL

Cascavel, 21 de novembro de 2012.

DEDICATÓRIA

Dedico este trabalho aos meus pais, Octavio Itacir Soares e Palmira Soares (in memorian), por sempre me guiarem no caminho certo e me darem apoio em todas as minhas decisões. Às minhas irmãs, Angélica Soares e Sidinéia Soares (in memorian), por toda força que me deram nessa trajetória e por serem grandes exemplos para mim. Ao meu sobrinho, Daniel Felipe Moretto, que também esteve presente em todos os momentos de minha vida.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Octavio Itacir Soares e Palmira Soares (*in memorian*), por sempre me trataram com todo amor e carinho, mostrando que os estudos e o conhecimento são fundamentais na formação de qualquer pessoa. Do fundo do meu coração, meu sincero obrigado, sem vocês eu não teria chegado até aqui, vou levar seus ensinamentos para o resto de minha vida.

Às minhas irmãs Angélica Soares e Sidinéia Soares (*in memorian*), por serem minhas companheiras e me aconselharem em todos os momentos difíceis que passei.

Ao meu sobrinho Daniel Felipe Moretto, primo Everton Luíz Folador e amigo Thomer Durman, por terem me proporcionado inúmeros momentos de felicidade, desde minha infância, vivenciando cada momento com companheirismo.

Ao meu tio Dionízio A. Zanco, minha tia Fátima D. Zanco e avó Angelina Zanco, por sempre terem me tratado de forma especial, se tornando pessoas essenciais para mim.

À Karen Josiane Soares Pereira, pessoa amada, que esteve ao meu lado e me deu forças para superar todos os momentos difíceis que enfrentei.

Ao meu orientador Prof. Dr. Clodis Boscarioli, que aceitou me orientar neste Trabalho de Conclusão de Curso e em dois projetos de iniciação científica, além de ser grande companheiro e conselheiro fora da vida acadêmica.

Aos professores André Luiz Brun, Aníbal Mantovani Diniz e Márcio Seiji Oyamada, que mais de uma vez tomaram seu tempo para conversar comigo sobre questões extra disciplinares.

Aos meus companheiros Leonardo Zanotto Baggio, Paolo Mautone Romera e Rodolfo Lorbieski, por todas as horas que passamos juntos concluindo trabalhos e estudando para provas, e pelos bons momentos vivenciados dentro e fora da Universidade.

Aos meus amigos Cauê Schulz Lopes, Felipe Augusto Henn e Rodrigo Trage, que tornaram minha passagem na Universidade mais divertida. Obrigado pela companhia nas festas e nos shows que participamos juntos, mesmo naqueles em que nos custaram a noite de sono para chegar a tempo na aula do dia seguinte.

Aos demais colegas da minha turma, que vivenciaram essa jornada comigo e se mantiveram firmes até o final, Alexandre Augusto Giron, André Specian Cardoso, Diego Robles Vieira Ribeiro, Luiz Gustavo de Souza e Maykon Valerio da Silva.

Por fim, agradeço a todos que direta ou indiretamente me ajudaram em algum momento de minha formação. Peço desculpas àqueles que, neste momento, eu possa ter me esquecido de mencionar, mas que acredito estejam cientes de sua importância para mim.

Lista de Figuras

Figura 1: Exemplo de um Banco de Dados Relacional 6
Figura 2: Exemplo da estrutura orientada a objetos 7
Figura 3: Exemplo de banco de dados Objeto-Relacional 8
Figura 4: Relação de depósito em uma conta bancária 11
Figura 5: Forma de armazenamento em (a) linhas e (b) colunas 15
Figura 6: Representação das tabelas Aluno, Disciplina e Aluno_Disciplina no modelo Orientado a Linhas (a) e Orientado a Colunas utilizando a abordagem (Chave, Atributo) (b) 20
Figura 7: Forma de armazenamento em disco dos bancos de dados de modelo colunar (modificando a camada de armazenamento do modelo relacional) 21
Figura 8: Forma de armazenamento em disco dos bancos de dados de modelo relacional 22
Figura 9: Trecho da representação de armazenamento colunar em disco sem utilização do método run-length (a) e com a utilização do método run-length (b) 24
Figura 10: Exemplo de instâncias para a tabela Aluno, apresentada na Figura 1 27
Figura 11: Tuplas geradas sobre o esquema da Tabela 10 para a consulta (6), utilizando a materialização EM 28
Figura 12: Tuplas geradas sobre o esquema da Tabela 10 para a consulta (6), utilizando a materialização LM 28
Figura 13: SSB Esquema - variação do esquema TPC-H. 30
Figura 14: Primeira fase do invisible join referente à consulta (7). 34
Figura 15: Segunda fase, primeiro passo, do invisible join referente à consulta (7). 34
Figura 16: Segunda fase, segundo passo, do invisible join referente à consulta (7). 35
Figura 17: Terceira fase do invisible join referente à consulta (7). 35
Figura 18: Gráficos representando (a) Tabela 4, (b) Tabela 5, (c) Tabela 6, (d) Tabela 7 42
Figura 19: Gráficos representando (a) Tabela 10, (b) Tabela 11, (c) Tabela 12, (d) Tabela 13 45
Figura 20: Gráficos representando (a) Tabela 16, (b) Tabela 18 46
Figura 21: Gráficos representando (a) Tabela 20 e (b) Tabela 22 48
Figura 22: Gráfico representando (a) Tabela 23 e (b) Tabela 24 51
Figura 23: Gráfico representando (a) Tabela 25 e (b) Tabela 26 52

Figura 24: Gráfico representando (a) Tabela 27 e (b) Tabela 28.....54
Figura 25: Gráfico representando (a) Tabela 29 e (b) Tabela 30.....55
Figura 26: Gráfico referente à Tabela 31 57
Figura 27: Gráfico referente à Tabela 32 58
Figura 28: Gráfico referente à Tabela 33 59
Figura 29: Gráfico referente à Tabela 34 60
Figura 30: Gráfico referente à Tabela 35 61
Figura 31: Gráfico referente à Tabela 36 62
Figura 32: Gráfico referente à Tabela 37 63
Figura 33: Gráfico referente à Tabela 38 64
Figura 34: Gráfico referente à Tabela 39 65
Figura 35: Gráfico referente à Tabela 40 66
Figura 36: Gráfico referente à Tabela 41 67
Figura 37: Gráfico referente à Tabela 42 68

Lista de Tabelas

Tabela 1: Modelos de SGBD colunar.....16
Tabela 2: Tempo de consulta, em segundos, para sete diferentes consultas utilizando o C-Store e um SGBD relacional comercial.....23
Tabela 3: Tempo de inserção dos dados no Cenário 1 (SF=1), com restrição de chaves.....39
Tabela 4: Tempo consumido para as três primeiras consultas do Cenário 1, SF=1, (Total/CPU/Saída), com restrição de chaves.....40
Tabela 5: Tempo consumido para cada consulta do Cenário 1, SF=1, (Total/CPU/Saída) com restrições de chaves, para o Teste 1.....40
Tabela 6: Tempo de inserção dos dados no Cenário 1, SF=1, sem restrição de chaves.....41
Tabela 7: Tempo consumido para as três primeiras consultas do Cenário 1, SF=1, (Total/CPU/Saída) sem restrição de chaves.....41
Tabela 8: Tempo consumido para cada consulta do Cenário 1, SF=1, (Total/CPU/Saída) sem restrições de chaves, para o Teste 1.....41
Tabela 9: Tempo de inserção (em segundos) dos dados no Cenário 2, SF=2, com restrição de chaves.....43
Tabela 10: Tempo consumido para as três primeiras consultas do Cenário 2, SF=2, (Total/CPU/Saída), com restrição de chaves.....43
Tabela 11: Tempo consumido para cada consulta do Cenário 2, SF=2, (Total/CPU/Saída) com restrições de chaves, para o Teste 1.....43
Tabela 12: Tempo de inserção dos dados no Cenário 2, SF=2, sem restrição de chaves.....44
Tabela 13: Tempo consumido para as três primeiras consultas do Cenário 2, SF=2, (Total/CPU/Saída), sem restrição de chaves.....44
Tabela 14: Tempo consumido para cada consulta do Cenário 2, SF=2, (Total/CPU/Saída) sem restrições de chaves, para o Teste 1.....44
Tabela 15: Tempo de inserção dos dados no Cenário 3, SF=5, com restrição de chaves.....45
Tabela 16: Tempo consumido para cada consulta do Cenário 3, SF=5, (Total/CPU/Saída) com restrições de chaves, para o Teste 1.....46
Tabela 17: Tempo de inserção dos dados no Cenário 3, SF=5, sem restrição de chaves.....46
Tabela 18: Tempo consumido para cada consulta do Cenário 3, SF=5, (Total/CPU/Saída) sem restrições de chaves, para o Teste 1.....47
Tabela 19: Tempo de inserção dos dados no Cenário 4, SF=10, com restrição de chaves.....47
Tabela 20: Tempo consumido para cada consulta do Cenário 4, SF=10, (Total/CPU/Saída) com restrições de chaves, para o Teste 1.....48

Tabela 21: Tempo de inserção dos dados no Cenário 4, SF=10, sem restrição de chaves 48
Tabela 22: Tempo consumido para cada consulta do Cenário 4, SF=10, (Total/CPU/Saída) sem restrições de chaves, para o Teste 1 49
Tabela 23: Tempo consumido para cada consulta do Cenário 1, SF=1, (Total/CPU/Saída) com restrições de chaves, para o Teste 2 50
Tabela 24: Tempo consumido para cada consulta do Cenário 1, SF=1, (Total/CPU/Saída) sem restrições de chaves, para o Teste 2 50
Tabela 25: Tempo consumido para cada consulta do Cenário 2, SF=2, (Total/CPU/Saída) com restrições de chaves, para o Teste 2 51
Tabela 26: Tempo consumido para cada consulta do Cenário 2, SF=2, (Total/CPU/Saída) sem restrições de chaves, para o Teste 2 52
Tabela 27: Tempo consumido para cada consulta do Cenário 3, SF=5, (Total/CPU/Saída) com restrições de chaves, para o Teste 2 53
Tabela 28: Tempo consumido para cada consulta do Cenário 3, SF=5, (Total/CPU/Saída) sem restrições de chaves, para o Teste 2 53
Tabela 29: Tempo consumido para cada consulta do Cenário 4, SF=10, (Total/CPU/Saída) com restrições de chaves, para o Teste 2 54
Tabela 30: Tempo consumido para cada consulta do Cenário 4, SF=10, (Total/CPU/Saída) sem restrições de chaves, para o Teste 2 55
Tabela 31: Percentagem do tempo consumido no segundo Cenário, SF=2, (utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 1) 56
Tabela 32: Percentagem do tempo consumido no segundo Cenário, SF=2, (não utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 1) 57
Tabela 33: Percentagem do tempo consumido no terceiro Cenário, SF=5, (utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 1) 58
Tabela 34: Percentagem do tempo consumido no terceiro Cenário, SF=5, (não utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 1) 59
Tabela 35: Percentagem do tempo consumido no quarto Cenário (utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 1) 60
Tabela 36: Percentagem do tempo consumido no quarto Cenário, SF=10, (não utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 1) 61
Tabela 37: Percentagem do tempo consumido no segundo Cenário, SF=2, (utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 2) 62
Tabela 38: Percentagem do tempo consumido no segundo Cenário, SF=2, (sem as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 2) 63
Tabela 39: Percentagem do tempo consumido no terceiro Cenário, SF=5, (utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 2) 64
Tabela 40: Percentagem do tempo consumido no terceiro Cenário, SF=5, (sem as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 2) 65

Tabela 41: Percentagem do tempo consumido no quarto Cenário, SF=10, (utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 2) 66
Tabela 42: Percentagem do tempo consumido no quarto Cenário, SF=10, (sem as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 2) 67
Tabela 43: Tamanho em disco requerido pelos bancos de dados de cada SGBD (utilizando restrições de chaves/não utilizando restrições de chaves) 68
Tabela 44: Percentagem do espaço em disco requerido para armazenar os bancos de dados (utilizando restrições de chaves/não utilizando restrições de chaves) 68
Tabela 45: Proporção do tamanho dos bancos de dados em disco em relação ao Infobright (Utilizando chaves/Não utilizando) 70
Tabela 46: Taxa de consultas por hora para o Teste 1, utilizando as restrições de chaves 70
Tabela 47: Taxa de consultas por hora para o Teste 1, sem as restrições de chaves 70
Tabela 48: Taxa de consultas por hora para o Teste 2, utilizando as restrições de chaves 70
Tabela 49: Taxa de consultas por hora para o Teste 2, sem as restrições de chaves 70
Tabela 50: Número de vezes que cada SGBD foi mais rápido que o PostgreSQL ao executar as consultas do Teste 1 (Utilizando chaves/Não utilizando) 72
Tabela 51: Número de vezes que cada SGBD foi mais rápido que o PostgreSQL ao executar as consultas do Teste 2 (Utilizando chaves/Não utilizando) 72
Tabela 52: Número de consultas que cada SGBD executou em menor tempo para o Teste 1 72
Tabela 53: Número de consultas que cada SGBD executou em menor tempo para o Teste 2 72
Tabela 54: Tabela LINEORDER (SF*6.000.000) 79
Tabela 55: Tabela PART (200,000*x(1+log2SF)) 79
Tabela 56: Tabela SUPPLIER (SF*2,000) 80
Tabela 57: Tabela CUSTOMER (SF*30,000) 80
Tabela 58: Tabela DATE (dias referentes a 7 anos) 80

Lista de Abreviaturas e Siglas

ACID	*Atomicity, Consistency, Isolation, Durability*
BASE	*Basically Avaliable, Soft State, Eventual Consistency*
BIT	*Binary Digit*
CAP	*Consistency, Avaliability, Partition Tolerance*
CLR	*Common Language Runtime*
CPU	*Central Processing Unit*
CQL	*Contextual Query Language*
CSV	*Comma Separated Values*
DW	*Data Warehouse*
EM	*Early Materialization*
I/O	*Input/Output*
JDBC	*Java Database Conectivity*
LM	*Late Materialization*
MPP	*Massive Parallel Processing*
MVCC	*Multiversion Concurrency Control*
NoSQL	*Not Only SQL*
ODBC	*Open Database Conectivity*
PHP	*Person Home Page*
RAM	*Random Acess Memory*
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*
TB	*Terabyte*

SUMÁRIO

Lista de Figuras xii
Lista de Tabelas xiv
Lista de Abreviaturas e Siglas xvii
Sumário xviii
Resumo xx
1 Introdução 1
2 Modelo de Banco de Dados 4
2.1 Modelo Relacional 5
2.2 Modelo Orientado a Objetos 7
2.3 Modelo Objeto-Relacional 8
2.4 NoSQL 8
3 Modelos de Banco de Dados NoSQL 9
4 Modelo Colunar de Banco de Dados 15
4.2 Modificação da Camada de Armazenamento 21
4.3 Particionamento Vertical e Modificação da Camada de Armazenamento 22
4.4 Métodos de Compressão 23
4.5 Materialização 26
4.6 Iteração de Bloco 29
4.7 Aplicações para o Modelo Colunar 29
4.7.1 *Join* Otimizado a Aplicações em Esquema Estrela 31
5 Avaliação Experimental 36
5.1 Padrão de Teste 1 39
5.1.1 Cenário 1 – SF=1 39
5.1.2 Cenário 2 – SF=2 41
5.1.3 Cenário 3 – SF=5 45
5.1.4 Cenário 4 – SF=10 47
5.2 Padrão de Teste 2 49
5.3 Análise 55
6 Conclusões 71

Apêndice A 74
Referências 82

RESUMO

A manipulação de grandes quantidades de dados é um desafio amplamente estudado, buscando-se, cada vez mais, alternativas para tratar essas informações. Os bancos de dados de modelo relacional vêm sendo bastante utilizados desde sua concepção, devido, dentre outros fatores, à sua intuitiva maneira de estruturar os dados, confiabilidade e fácil manipulação. No entanto, com o constante crescimento da geração de informações a todo instante, emergem novas abordagens de manipulação de dados, a fim de atender às crescentes necessidades do mercado. Os bancos de dados NoSQL estão ganhando reconhecimento nesse cenário, fornecendo alternativas ao modelo relacional para manipulação de dados. Este trabalho apresenta os bancos de dados NoSQL, em especial o modelo colunar, destacando suas principais características e diferenças em relação ao modelo relacional. Propôs-se também uma forma de comparação experimental entre alguns SGBD de modelo relacional e colunar em diferentes situações de escalabilidade, mostrando que o modelo colunar possui vantagens em tempo de consulta e armazenamento em disco (compressão), porém, apresentando desvantagem na inserção de dados. Esses aspectos são descritos e analisados no texto, discutindo quais razões tornam o modelo colunar mais estável ao crescimento (escalabilidade) do que o modelo relacional.

Palavras-chave: Modelos de Banco de Dados, NoSQL, Banco de Dados Colunar, Banco de Dados Relacional.

Capítulo 1

Introdução

O armazenamento de informações sempre foi um fator fundamental ao se trabalhar com dados de forma digital. Antigamente os dados eram armazenados e recuperados apenas por meio de sistemas de arquivos, que não forneciam recursos confiáveis para evitar problemas como perda de informações, redundância, inconsistência e controle de acesso. Os bancos de dados e SGBD (Sistema de Gerenciamento de Banco de Dados) surgiram como uma forma alternativa para estruturação e manipulação desse tipo de informação, bem como para minimizar os problemas existentes com os recursos mais antigos.

Desde sua concepção até os dias atuais, os SGBD são os sistemas mais utilizados para trabalhar com armazenamento de dados. Existem diversos tipos de SGBD, os quais operam com diferentes tipos de dados, possuem estratégias próprias para organizá-los e fornecem recursos diferenciados aos usuários, como *backup* e restauração, armazenamento paralelizado, etc.

Dentre os modelos de bancos de dados existentes o relacional se destaca, sendo muito utilizado desde sua concepção. É de fácil manipulação - principalmente pela utilização de SQL[1] – *Structured Query Language* – como linguagem de consulta, e possui recursos que garantem consistência e integridade aos dados, a exemplo utilização de chaves primárias, chaves estrangeiras e normalização.

Visto o crescimento e diversidade de informações geradas a todo tempo, os modelos relacionais nem sempre são a melhor alternativa para manipulação de dados. Isso ocorre por diversos fatores, como a forma de armazenamento dos dados em disco, a forma com que o executor de consultas recupera as informações e monta as tabelas e o método de compressão utilizado. Principalmente em aplicações com grandes quantidades de dados, onde consultas complexas são executadas, o modelo relacional enfrenta redução significativa em seu

[1] Linguagem de pesquisa declarativa para bancos de dados de modelo relacional baseada na álgebra relacional.

desempenho na leitura de informações. Os bancos de dados NoSQL (discutidos no Capítulo 3) são uma alternativa para lidar com esses casos, por conta de possuírem uma arquitetura diferenciada, de forma a facilitar o tratamento com alta escalabilidade de dados. No entanto, o modelo relacional possui a vantagem na facilidade de escrita em disco, o que é um problema maior para o modelo colunar. Dentro dos bancos de dados NoSQL, destacam-se os de modelo colunar, que vêm sendo utilizados por grandes empresas, a exemplo a Google, o Facebook e o Twitter, que adotaram bancos de dados deste modelo devido à grande demanda por consultas, vinculada ao elevado número de informações com que trabalham.

A partir dessas considerações, torna-se interessante avaliar em que pontos os bancos de dados de modelo colunar são melhor aplicáveis que os de modelo relacional. Diversos SGBD de modelo colunar possuem suporte a SQL e manipulam informações de forma muito semelhante a outros SGBD de modelo relacional, podendo assim, haver uma comparação adequada entre os modelos. Esse trabalho propõe uma comparação teórica e prática entre os bancos de dados de modelo colunar e relacional, apresentando as principais características do modelo colunar e suas diferenças em relação ao relacional, buscando verificar quais são as situações em que os SGBD de modelo colunar são melhor aplicáveis do que os SGBD de modelo relacional e de quanto é essa diferença.

Este documento está organizado da seguinte forma:

O Capítulo 2 fornece conceitos gerais sobre banco de dados e os modelos mais conhecidos e utilizados.

O Capítulo 3 dá ênfase nos bancos de dados NoSQL, apresentando os conceitos, principais modelos e motivações para utilização desses bancos de dados.

O Capítulo 4 descreve os bancos de dados de modelo colunar, descrevendo suas principais características, arquitetura e métodos de implementação, listando os principais SGBD que suportam esse modelo.

O Capítulo 5 apresenta uma forma de comparação entre SGBD de modelo relacional e colunar, mostrando como cada SGBD se comporta em uma estruturação de dados de formato em estrela, perante diferentes situações de escalabilidade.

No Capítulo 6 apresentam-se conclusões gerais sobre a modelagem de dados colunar e relacional, baseado no levantamento teórico construído e na avaliação experimental realizada, levando em conta os resultados obtidos com os testes aplicados.

O Apêndice A descreve o padrão de *Benchmark* SSB, apresentando sua forma de estruturação, as consultas fornecidas e as alterações em relação ao TPC-H.

Capítulo 2

Modelo de Banco de Dados

O sistema de arquivos foi a primeira das estratégias de persistência de dados a ser utilizada. No entanto, essa forma de armazenamento não possuía recursos para organizar as informações de forma adequada, o que dificultava a manipulação, principalmente, com grandes quantidades de dados. Os bancos de dados e os SGBD surgiram como uma forma alternativa para estruturação e manipulação desse tipo de informação, bem como para minimizar os problemas de perda de dados, redundância, inconsistência e controle de acesso.

- De acordo com Elmasri e Navathe (2003), um banco de dados é uma coleção de dados relacionados, sendo dados fatos que podem ser gravados e têm significado implícito, possuindo as seguintes propriedades:Representam algum aspecto do mundo real, às vezes chamado de minimundo, tal que mudanças no minimundo refletem no banco de dados;
- É uma coleção de dados lógica e coerente com algum significado inerente. Uma quantidade aleatória de dados não pode ser corretamente referenciada como um banco de dados;
- Um banco de dados é projetado, construído e povoado com dados para um propósito específico. Possui um grupo designado de usuários e algumas aplicações pré-concebidas as quais esses usuários estão interessados;

Para facilitar as operações realizadas sobre um banco de dados (como recuperação, alteração e definição), bem como para dar suporte a eles, foram criados os SGBD, que são sistemas de gerenciamento de bancos de dados. São capazes de fornecer recursos que um banco de dados não pode tratar sozinho [Elmasri e Navathe, 2003], a exemplo de:

- Controle de redundância (uma informação não é gravada mais de uma vez no mesmo modelo);

- Restrição de acesso não autorizado (apenas usuários com permissão têm acesso ao banco de dados);
- Persistência dos dados (garante que os dados uma vez escritos permanecerão no modelo);
- *Backup* e restauração (o projetista pode salvar as informações pertencentes ao modelo e recuperá-las em outro momento);
- Restrições de integridade (garantir exatidão e consistência nos dados).

Existem diferentes modelos de bancos de dados, que operam com diferentes tipos de dados e possuem estratégias próprias de organização[2]. Os primeiros a surgir foram os modelos hierárquico [Tsichritzis e Lochovsky, 1976] e de redes.

O modelo hierárquico organiza as informações em uma estrutura de árvore. Nesse tipo de estrutura existem dois componentes principais: os nós e as conexões que ligam um nó a outro. Cada nó contém informação sobre um registro e pode estar ligado a apenas outro, não sendo permitida a formação de ciclos. O modelo de redes surgiu como uma extensão ao modelo hierárquico, mantendo a mesma estrutura de ligação de nós, mas deixando de lado a hierarquia e permitindo que um nó esteja envolvido em várias associações. Os modelos de bancos de dados posteriores e mais comuns (mais utilizados) são descritos a seguir.

2.1 Modelo Relacional

Foi proposto por Codd (1970), com a premissa de que os dados deveriam ser manuseados em estruturas matematicamente desenvolvidas. Representa os dados em uma coleção de tabelas (entidades, associadas a algum objeto do mundo real), contendo um conjunto de atributos, onde cada atributo possui um valor dentro de seu domínio. Ao conjunto de atributos que representa uma linha na tabela é dado o nome de registro ou *tupla*. Tabelas podem ser relacionadas, associando atributos de uma tabela com atributos de outras tabelas.

Também é possível fazer o uso de chaves para garantir a integridade e consistência nos dados. Os dois principais tipos de chaves são a primária e a estrangeira. A chave primária garante que um mesmo valor de atributo (ou conjunto de atributos, caso a chave seja composta) não seja repetido em uma tabela. A chave estrangeira garante a integridade na relação entre duas tabelas, não permitindo referências a valores de atributos não existentes.

[2] Para maior aprofundamento sobre fundamentos, modelagem e estrutura de banco de dados e SGBD rencomenda-se [Heuser, 2008], [Silberchatz, Korth e Sudarshan, 2010] e [Elmasri e Navathe, 2010].

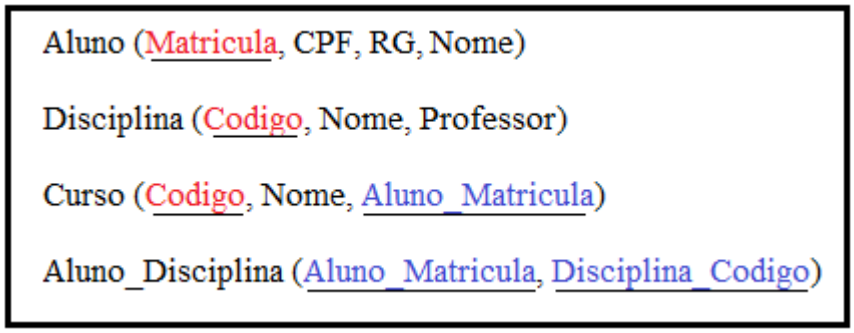

Figura 1: Exemplo de um Banco de Dados Relacional

Para exemplificar os conceitos expostos acima, um pequeno exemplo de banco de dados relacional é apresentado na Figura 1, onde existem quatro tabelas: *Aluno*, *Disciplina*, *Curso* e *Aluno_Disciplina*. Os atributos representados na cor vermelha, sublinhados, compõem a chave primária e os azuis, sublinhados, as chaves estrangeiras.

Na tabela *Aluno_Disciplina*, os atributos "Aluno_Matricula" e "Disciplina_Codigo" são chaves estrangeiras, referenciando as tabelas *Aluno* e *Disciplina*, respectivamente. Cada nova linha na tabela *Aluno_Disciplina* conterá uma referência para um aluno e uma referência para uma disciplina. A chave estrangeira garantirá que a inserção só ocorra caso o número de matrícula aluno inserido exista na tabela *Aluno* e o código da disciplina exista na tabela *Disciplina*. A tabela *Aluno_Disciplina* pode ser entendida como a relação entre as tabelas *Aluno* e *Disciplina*, armazenando informações sobre quais alunos estão matriculados em quais disciplinas.

Além dos recursos mais simples citados, o modelo relacional ainda fornece outros mais complexos, como asserções, gatilhos e funções. Também, segue o conceito ACID (Atomicidade, Consistência, Isolamento e Durabilidade, em inglês *Atomicity*, *Consistency*, *Isolation* e *Durability*), o qual garante atomicidade nas operações, consistência nos dados, operações realizadas de forma isolada e persistência dos dados.

De acordo com Brito (2010) o modelo relacional é amplamente utilizado em praticamente todos os tipos de sistemas de bancos de dados nas últimas décadas. São exemplos de SGBD que suportam esse modelo: PostgreSQL [PostgreSQL, 2012], Firebird [Firebird, 2012], MySQL [Oracle, 2012b], Oracle [Oracle, 2012a] e SQLServer [Microsoft, 2012].

2.2 Modelo Orientado a Objetos

Surgiu na década de 80, principalmente, para tratar de dados mais complexos, não tradicionais, os quais os quais o modelo relacional não tratava de forma adequada. De acordo com Boscarioli et al. (2006) esse modelo de banco possui três pilares principais: herança (permite que uma classe seja estendida por outra classe), polimorfismo (permite comportamento diferenciado por duas classes, desde que sejam especializações de uma mesma classe e o método tenha a mesma assinatura) e encapsulamento (permite que apenas as informações que o objeto julgue apropriadas sejam visualizadas externamente, de acordo com o contexto da aplicação.

Como exemplo na Figura 2, o banco de dados relacional da Figura 1 será transcrito para um orientado a objetos. As entidades *Aluno*, *Disciplina*, *Curso* e *Aluno_Disciplina* são tratadas como objetos e os atributos primitivos são mantidos em cada um deles, da mesma forma que no modelo relacional. As chaves estrangeiras de *Curso* e *Aluno_Disciplina* são utilizadas como atributos, mantidas dentro dos próprios objetos, como mostra a Figura 2:

Class Aluno	**Class** Disciplina	**Class** Curso	**Class** Aluno_Disciplina
int Matricula; **double** CPF; **int** RG; **String** Nome;	**int** Codigo; **String** Nome; **String** Professor;	**int** Codigo; **String** Nome; Aluno Alunos[ ];	Aluno Alunos[ ]; Disciplina Disciplinas[ ];

Figura 2: Exemplo da estrutura orientada a objetos

Nota-se que o esquema de modelo orientado a objeto da Figura 2 é muito parecido com o formato do banco de dados relacional da Figura 1. Porém, a referência da ligação de alunos a disciplinas e alunos a um curso é feita de forma diferente. Como mostra na Figura 2, *Curso* possui uma lista de objetos do tipo *Aluno* e *Aluno_Disciplina* duas listas, uma de objetos do tipo *Aluno* e outra do tipo *Disciplina*. Isso implica que cada novo objeto do tipo *Curso* possuirá um código, um nome e *n* objetos do tipo *Aluno*, e cada novo objeto do tipo *Aluno_Disciplina* possuirá *x* objetos do tipo *Aluno* e *x* objetos do tipo *Disciplina*.

São exemplos de SGBD que suportam esse modelo: Objectivity/DB [Objectivity, 2012], GemStone [GemStone, 2011], Jasmine [Jasmine, 2001] e DB4o [Versant, 2012].

2.3 Modelo Objeto-Relacional

Mescla características do modelo relacional e orientado a objetos e adiciona novas capacidades de armazenamento ao modelo relacional, como músicas, registros, imagens, etc. Além dos tipos primitivos de armazenamento, o modelo permite que o usuário crie novos tipos, como no modelo orientado a objetos, e utilize-os em junto aos atributos primitivos na criação de tabelas.

Ambos os exemplos das Figuras 1 e 2 são também exemplos de bancos de dados do modelo objeto-relacional, como também uma junção destes, apresentada na Figura 3.

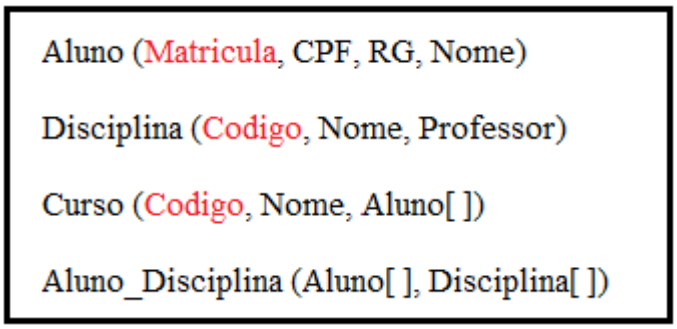

Figura 3: Exemplo de banco de dados Objeto-Relacional

São exemplos de SGBD que suportam esse modelo: PostgreSQL, Oracle 9i [Oracle, 2012a], Informix [Informix, 2012] e DB2 [IBM, 2012].

2.4 NoSQL

Não há uma definição "padrão" para dizer exatamente quais tipos de bancos de dados se encaixam no termo "NoSQL (Não apenas SQL, em inglês *Not Only SQL*)", nem qual estrutura, forma de armazenamento, tipos de dados e operações que comportam.

Marcus (2011) define o termo NoSQL como "*um sistema que apresenta uma interface de consulta que não é apenas SQL"*. Em Han *et al*. (2011) é afirmado que os bancos de dados NoSQL surgiram para satisfazer necessidades como: (i) Armazenamento de grandes volumes de dados de forma eficiente e requerimentos de acesso; (ii) Alta escalabilidade e disponibilidade; e, (iii) Menor custo operacional e de gestão;

Mais informações sobre bancos de dados NoSQL são dadas no Capítulo 3, haja vista o modelo colunar, foco deste estudo, ser classificado como NoSQL.

Capítulo 3

Modelos de Banco de Dados NoSQL

Os modelos de banco de dados NoSQL vêm sendo amplamente estudados e aplicados nos últimos anos (principalmente nos seis últimos). Tornaram-se muito importantes e estão ganhando espaço em grandes empresas, substituindo outros sistemas que prevaleciam há anos. O porquê de esses fatos estarem ocorrendo são explicados neste capítulo.

Desde o início da utilização de banco de dados, uma quantidade massiva de dados é gerada a todo tempo, crescendo cada vez mais. Como dito, o modelo relacional é o mais utilizado para o gerenciamento de dados, visto sua estruturação, confiabilidade e bom desempenho na manipulação de dados. No entanto, a escalabilidade vem se tornando um problema para alguns SGBD que seguem o modelo relacional. Devido ao aumento gradativo de informações a serem armazenadas, há vários casos, em diferentes domínios de aplicação, em que o desempenho é prejudicado e o tempo de resposta começou a se tornar um fator preocupante. Pritchett (2008) afirma que quando um banco de dados cresce além de sua capacidade em um único nó (servidor) é necessário optar por escalabilidade horizontal (paralelismo) ou vertical (reforçar o servidor).

Escalabilidade *versus* desempenho é um fator de motivação para utilização de novas estratégias para lidar com tal massividade de dados, dentre outros, como o uso de *hardware* comum e barato para operação dos sistemas [Diana e Gerosa, 2010]. Uma delas (entre outras, como adicionar *hardware* mais potente ao servidor) foi a aplicação de modelos alternativos de bancos de dados, principalmente a utilização dos modelos NoSQL.

NoSQL não é apenas mais um modelo de banco de dados, mas um termo que define uma classe de modelos, sendo os mais comuns e aplicados:

- Orientado a chaves: A estrutura desse modelo é como em uma tabela *Hash*, ou seja, há diversas chaves na tabela, cada qual referenciando um valor (por valor entende-se um

tipo de dado). Suporta uma grande quantidade de dados e alta concorrência [Jan *et al*., 2011]. São exemplos de SGBD que suportam esse modelo: RIAK [Basho, 2012], Redis [Priogilo, 2012] e MemcacheDB [MemcacheDB, 2009].

- Orientado a colunas: Também chamado de modelo colunar, utiliza-se de tabelas para representação de entidades, porém, não usa de associação. *Os dados são gravados em disco, agrupados por colunas, o que reduz o tempo de leitura e escrita em disco* [Matei, 2010], [Abadi, Madden e Hachem, 2008], [Scalzo, 2010]. O modelo colunar é descrito mais detalhadamente no Capítulo 4, junto com exemplos de SGBD que suportam esse modelo.
- Orientado a documentos: Similar ao modelo orientado a chaves, podendo gerar uma chave secundária para indexar seu valor. São exemplos de SGBD que suportam esse modelo o MongoDB [MongoDB, 2012] e o CouchDB [Apache, 2012b].

Os SGBD NoSQL vêm ganhando cada vez mais importância e sendo aplicados em diversas empresas (Google, Facebook, Twitter, Dig, Amazon, etc.). Dentre as classes de modelos citadas, o modelo colunar é um dos que mais se destaca no conceito NoSQL.

Uma das características chave dos bancos de dados NoSQL é a habilidade de realizar escalonamento horizontal [Cattel, 2010], pois trabalham com dados denormalizados, e se tornam uma alternativa para substituição dos bancos relacionais em situações em que o desempenho é afetado pela escalabilidade. O modelo colunar vem sendo amplamente aplicado na substituição do relacional, devido a ambos tratarem do mesmo tipo de dados, trabalharem bem com SQL [Mcknight, 2011] e pela sua superioridade em desempenho com grandes quantidades de dados para certas aplicações (Seção 4.7).

Além de desempenho, há que considerar outros fatores como a consistência, a disponibilidade e tolerância à partição. De acordo com o Teorema CAP (*Consistency*, *Avaliability*, *Partition Tolerance*), proposto por Gilbert e Lynch (2002), apenas dois dos três itens abaixo podem ser satisfeitos concorrentemente em um modelo de banco de dados:

- Consistência: o cliente percebe que um conjunto de operações ocorreu de uma só vez;
- Disponibilidade: cada operação deve terminar em uma resposta destinada;
- Tolerância à partição: operações serão completadas, mesmo se componentes individuais estiverem indisponíveis.

Seguindo a ideia do Teorema CAP, como a flexibilidade em escalonamento horizontal é promovida pelo modelo NoSQL, é necessário escolher entre consistência ou disponibilidade como segundo fator.

O modelo relacional segue o conceito ACID. No entanto, Pritchett (2008) questiona que se este garante consistência para bancos de dados particionados, então a disponibilidade é deixada em segundo plano (de acordo com o Teorema CAP). Em contraposição a isso, o autor propôs o Teorema BASE (*Basically Avaliable*, *Soft State*, *Eventual Consistency*), sugerindo que o ACID é pessimista e força a consistência no final de cada operação, enquanto o BASE é otimista e aceita que a consistência no banco de dados estará em um estado de fluxo.

A disponibilidade do Teorema BASE é garantida tolerando falhas parciais no sistema, sem que o sistema todo falhe. Por exemplo, se um banco de dados está particionado em cinco nós e um deles falha, apenas os clientes que acessam aquele nó serão prejudicados, pois o sistema como todo não cessará seu funcionamento.

A consistência pode ser "relaxada" permitindo que a persistência no banco de dados não seja efetivada em tempo real (ou seja, logo depois de realizada uma operação sobre o banco). Pelo ACID, quando uma operação é realizada no SGBD (a exemplo *insert*, *update* e *delete*), ela só será finalizada se houver a certeza de que a persistência dos dados foi realizada no mesmo momento. Já no BASE isso não se confirma. Para garantir a disponibilidade, algumas etapas são dissociadas à operação requisitada, sendo executadas posteriormente. O cliente realiza uma operação no banco de dados e, não necessariamente, a persistência será efetivada naquele instante.

Para exemplificar o relaxamento da consistência, considere como exemplo uma pessoa que deseja depositar uma quantia em uma conta bancária (Figura 4).

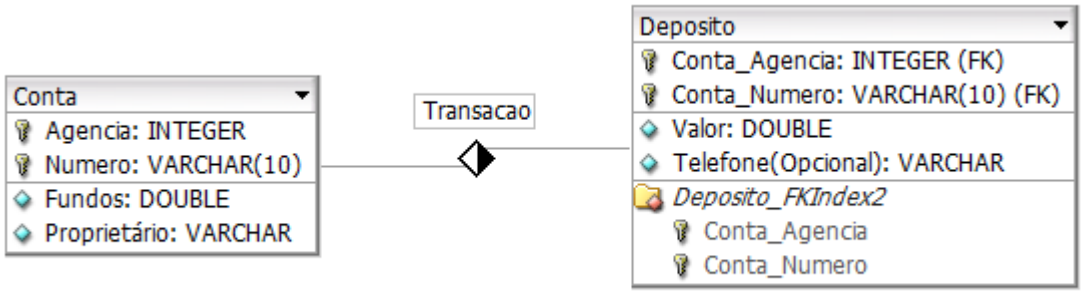

Figura 4: Relação de depósito em uma conta bancária

Quando a operação de depósito é realizada, a quantia inserida deve ser somada ao atributo *Fundos* da entidade *Conta*. Supondo que os números da conta e da agência são 150 e 15, respectivamente, o valor a ser depositado é de 200 e a conta possui fundos igual a 500, as SQL utilizadas para realizar essa operação são mostradas em (1) e (2).

```
*Início da Transação
Insert into Deposito (Conta_Agencia, Conta_Numero, Valor)
                  Values (15, 150, 200);                    (1)
Update Conta Set Fundos = Fundos+200
             Where Agencia = 15 and Conta = 150;            (2)
*Fim da Transação
```

A SQL (1) insere os dados do depósito na entidade *Deposito* e a SQL (2) atualiza os fundos da entidade *Conta*. Da forma que é colocado no trecho de código abaixo a consistência é garantida, visto que ou as duas SQL serão executadas de uma vez ou nenhuma delas será.

Um exemplo de relaxamento na consistência é dado em (3) e (4):

```
*Início da Transação
Insert into Deposito (Conta_Agencia, Conta_Numero, Valor)
                    Values (15, 150, 200);                  (3)
*Fim da Transação
*Início da Transação
Update Conta Set Fundos = Fundos+200
             Where Agencia = 15 and Conta = 150;            (4)
*Fim da Transação
```

Note que as consultas (3) e (4) continuam as mesmas (2) e (3), respectivamente, mas o modo com que são executadas é diferente. As SQL são desmembradas, fazendo com que cada uma seja executada independentemente. Primeiro, os dados do depósito são inseridos na entidade *Deposito* e, depois que esse processo termina, um novo se inicia, atualizando os fundos da entidade *Conta*. Se uma das operações falhar, o banco de dados irá se tornar inconsistente.

A princípio, o modelo proposto por Pritchett (2008) parece ser inviável, pois consistência é fundamental em um banco de dados. No entanto, mesmo que a persistência seja, às vezes, deixada em segundo plano, é possível forçar sua posterior execução, enfileirando mensagens junto à operação realizada pelo cliente e resgatando-as depois, uma a uma, executando a persistência para cada mensagem recuperada, como apresentado no trecho de código abaixo (5):

```
*Início da Operação
Insert into Deposito (Conta_Agencia, Conta_Numero, Valor)
                     Values (15, 150, 200);
Empilhar mensagem "Update Conta set Fundos = Fundos +200
                  Where Agencia = 15 and Conta = 150"
*Fim da Operação

Enquanto (Pilha contém mensagem)
  *Inicio da Operação
    --Verificar mensagem do topo
      --Executar a SQL
    Se (Operação foi realizada com sucesso)
      Remover mensagem da Pilha
    Fim Se
  *Fim da Operação
Fim Enquanto                                                    (5)
```

Para Pritchett (2008), o Teorema BASE pode elevar o sistema a níveis de escalabilidade que não podem ser obtidos com ACID. No entanto, algumas aplicações necessitam que a consistência seja precisamente empregada, como o exemplo acima. Nenhuma aplicação bancária poderá por em risco operações de saque, depósito, transferência, etc. O projetista do banco de dados deverá estar ciente de que se utilizar o Teorema BASE estará ganhando disponibilidade em troca de consistência, o que pode afetar os usuários da aplicação referente ao banco de dados.

A utilização do Teorema BASE não é padrão nos SGBD NoSQL. Alguns seguem o ACID, outros aplicam conceitos referentes ao BASE e há também os que permitem o administrador de banco de dados optar entre um ou outro, como o Cassandra [Apache, 2012a]. A escolha entre ACID ou BASE dependerá do tipo de aplicação com que se irá trabalhar.

Capítulo 4

Modelo Colunar de Banco de Dados

O modelo colunar de banco de dados é um dos modelos pertencentes aos bancos de dados NoSQL. De acordo com Abadi, Boncz e Harizopoulos (2009), o modelo colunar de armazenamento mantém cada coluna do banco de dados separadamente, guardando contiguamente os valores de atributos pertencendo à mesma coluna, de forma densa e comprimida. Segundo Abadi, Madden e Hachem (2008), Matei (2010) e Scalzo (2010), esse modo de armazenamento reduz o tempo de leitura e escrita em disco. Matei (2010) afirma que o modelo colunar é mais eficiente quando é preciso agregar um grande número de linhas, mas um pequeno número de colunas é requisitado, e o desempenho é ainda maior quando o tamanho da linha é pequeno, que pode ser recuperada com apenas um acesso a disco.

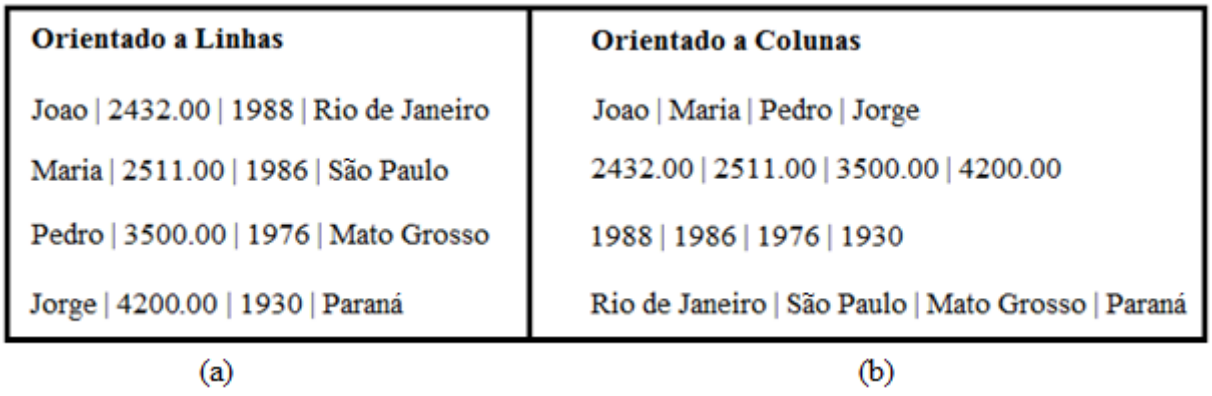

Figura 5: Forma de armazenamento em (a) linhas e (b) colunas

Pela Figura 5 é fácil analisar a diferença, de forma visual, entre a manipulação de dados relacional e colunar. No modelo relacional, cada atributo de uma instância é guardado consecutivamente em uma linha e no modelo orientado a colunas cada linha contêm todos os valores de um mesmo atributo.

Tabela 1: Modelos de SGBD colunar.

SGBD	Empresa	Sistemas Operacionais	Licença	Interfaces	Site	Características Gerais	Linguagem de consulta	Índices
MonetDB	Equipe de Desenvolve-dores MonetDB	Linux, Fedora, RedHat enterprise Linux, Debian, Ubuntu, Gentoo, Mac OS, SUN Solaris, Open Solaris e Windows (XP, Server 2003 e Vista)	*Open Source*	SQL, JDBC, ODBC, PHP, Python, RoR, C/C++ e Pearl	http://www.monetdb.org	ACID, Particionamento Vertical, Permite execução paralelizada	SQL 2003 padrão	*Hash*
Infobright	Enterprise Software & Database Management & Data Warehousing	Windows Server 2003 (32 e 64 bit), Solaris 10 (64 bit), Red Hat Enterprise Linux 5 (64 bit), Novell SUSE Linux Enterprise 10, Red Hat Enterprise Linux 5 Advanced Server (64 bit), Debian 'Lenny' (64 bit), or CentOS 5.2 (64 bit)	*Community* e *Enterprise*	JDBC, ODBC, C, C++, C#, dbExpress (Borland Delphi), Eiffel, SmallTalk, Lisp, Pearl, PHP, conector nativo do Java, Python, Ruby, REALbasic, FreeBasic e Tcl	http://www.infobright.com	ACID e Alta escalabilidade	ANSI SQL-92 com algumas extensões do SQL-99	
Cassandra	Apache Software Fundation	Linux, Mac OS e Windows	*Open Source*	Java, CQL, JDBC, DB-API 2.0, CLI, (Python), PDO (PHP) e DBI-compatible (Ruby)	http://cassandra.apache.org/	MVCC, Opção de escolha entre Consistência e Disponibilidade	CQL	
Vectorwise [Actian, 2012]	Actian	Linux e Windows	*Enterprise*	SQL, JDBC, ODBC e .NET	http://www.actian.com/products/vectorwise	ACID, MPP, Alta Disponibilidade	SQL (não informado padrão)	

BigTable	Google	Linux, Mac OS X e Windows	Proprietária			Alta Disponibilidade, Consistência, Persistência e Tolerante a Partição		
HBase [Apache, 2012c]	Apache Software Foundation	Linux, Mac OS X e Windows	*Open Source*	SQL*, JDBC*	http://hbase.apache.org/	Consistência, Persistência, Tolerante a Partição		
Metakit [Equi4, 2009]	EQUI4 software	Unix,Windows, Macintosh e VMS	*Open Source*	C++, Mk4py (Python) e Mk4tcl (Tcl)	http://equi4.com/metakit/		Tclkit	
InfiniDB [Calpont, 2012]	Calpont	Windows, Ubunto v5, Debian 'lenny' Linux v5, CentOS Linux v5 e Red Hat Enterprise Linux v5 (todos 64 bit)	*Community* e *Enterprise*.	JDBC, ODBC, ADO.NET, C/C++, PHP, Pearl, Python e Ruby	http://infinidb.org	ACID, MVCC e MPP	SQL (não informado padrão)	
C-Store	Colaboração entre MIT, Yale, Universidade Brandeis, Universidade Brown e UMass Boston	Linux (32 bit)	*Open Source*		http://db.csail.mit.edu/projects/cstore/	Alta Disponibilidade, Tolerante a Partição	SQL (não informado padrão)	*Bit map*
Widebase [Widebase, 2012]	Widebase	Unix	*Open Source*	C++, Erland, Go, Haskell, Java, PHP, Python, Ruby e Scala	http://widebase.github.com/	Trabalha com Operações de I/O e suporte a CSV		

Vertica[HP, 2012]	HP	Windows, Linux, Solaris e AIX	*Enterprise*	JDBC, ODBC e ADO.Net	http://www.vertica.com/	Alta Disponibilidade, MPP	SQL (não informado padrão)	
Sybase IQ [Sybase, 2012]	Sap Company	Windows, Linux, Unix, Sun Solaris, HP-UX e IBM AIX	*Enterprise*	JDBC, ODBC, C/C++, Java, Pearl, Python, CLR e Ruby	http://www.sybase.com.br/products/datawarehousing/sybaseiq	MPP	SQL (não informado padrão)	Bit-a-bit

*Operações possíveis com ajuda da API HBql, disponível em: http://hbql.com

Matei (2010) afirma que isso facilita a execução de operações matemáticas, como encontrar o menor valor, o maior valor, a soma dos valores dos atributos, o número de elementos e a média dos elementos de uma coluna.

De acordo com Abadi (2008) há três maneiras de implementar um SGBD de modelo colunar, e isso pode ser feito a partir de um SGBD que suporta banco de dados de modelo relacional:

- **Particionamento Vertical**: Particiona-se cada atributo de cada tabela, formando uma nova tabela com duas colunas (ver Seção 4.1).
- **Modificação da Camada de Armazenamento**: A camada de armazenamento do SGBD é modificada, fazendo com que guarde os dados em disco coluna por coluna (ver Seção 4.2).
- **Particionamento Vertical e Modificação da Camada de Armazenamento**: É uma mesclagem das duas outras abordagens (ver Seção 4.3).

Abadi, Madden e Hachem (2008) afirmam que os bancos de dados colunares possuem três pontos principais em que se destacam sobre os relacionais: Compressão, Materialização e Bloco de Iteração, discutidos ainda neste capítulo.

A Tabelas 1 apresenta os principais SGBD de modelo colunar, juntamente com suas características gerais.

4.1 Particionamento Vertical

As tabelas do banco de dados são particionadas de modo que, para cada atributo da tabela, uma nova tabela com duas colunas seja criada, formando o par (chave da tabela, atributo) [Khoshfian et al., 1987], como mostra a Figura 6.

Pela Figura 6 nota-se o desmembramento dos atributos das três tabelas. Para cada uma, os atributos foram particionados, gerando uma nova tabela a qual contém como primeiro campo a referência para a tabela a qual pertence e como segundo campo o atributo em si. Dessa maneira, quando uma consulta é requisitada ao processador de consultas, apenas são lidos os atributos referentes àquela consulta, não necessitando ler atributos desnecessários. Os atributos requeridos na consulta são unidos pelas respectivas chaves, formando uma projeção da tabela original, contendo apenas os atributos relevantes à consulta [Abadi, 2008].

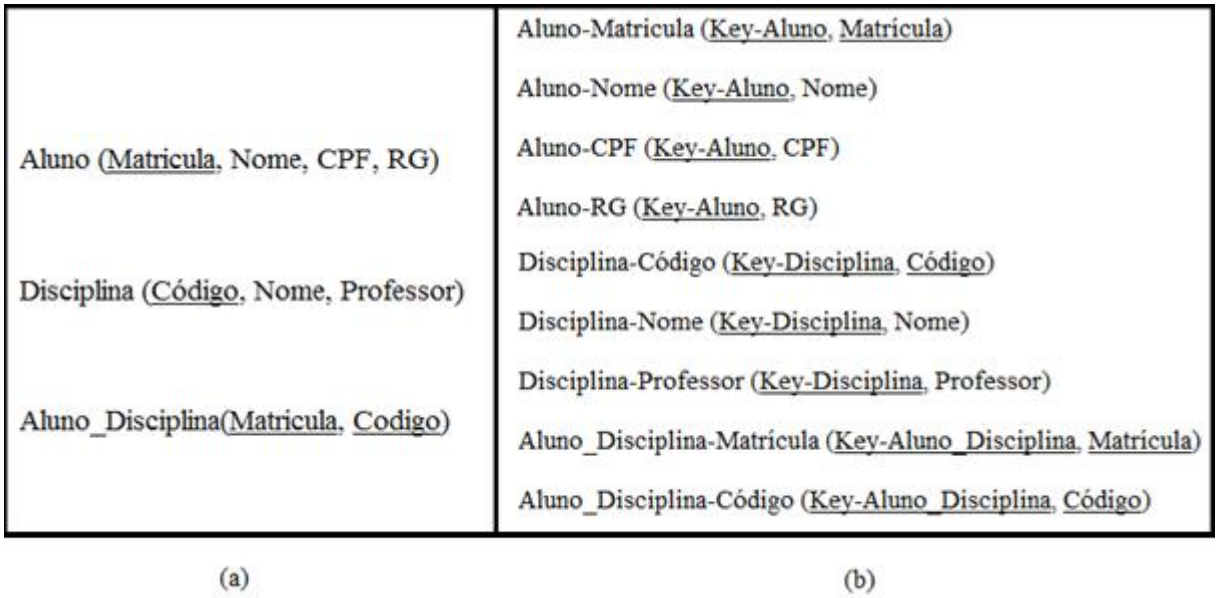

Figura 6: Representação das tabelas Aluno, Disciplina e Aluno_Disciplina no modelo Orientado a Linhas (a) e Orientado a Colunas utilizando a abordagem (Chave, Atributo) (b)

É importante ressaltar que essa abordagem não muda o modo de armazenamento de registros do banco de dados, apenas altera a estrutura de como as tabelas são manipuladas. Por isso, essa abordagem é a forma mais simples de simular um banco de dados de modelo colunar, pois a única mudança feita no banco de dados é no esquema de representação das tabelas e no mecanismo de execução de consultas.

Por não modificar a camada de armazenamento do SGBD, essa abordagem não possui grandes vantagens sobre o modelo relacional e muito menos sobre as demais formas de implementação de bancos de dados colunares. Na verdade, como mostrado em Abadi (2008), em diversas ocasiões o método de particionamento vertical se mostrou mais lento que o relacional. Das treze consultas testadas, em apenas uma o método de particionamento vertical obteve melhor resultado do que o modelo relacional.

Uma das desvantagens do particionamento vertical é não poder fazer uso do método de compressão da mesma forma que em bancos de dados colunares que possuem a camada de armazenamento construída por colunas (ver Seção 4.2). Também, o autor comenta a necessidade de armazenar a posição de cada atributo junto aos registros, para cada coluna (que pode ser feita através de um inteiro contador, ou de uma chave estrangeira), para que seja possível enumerar os atributos de mesma linha e acessá-los consecutivamente em disco.

4.2 Modificação da Camada de Armazenamento

Uma abordagem mais interessante, porém mais complexa, de se construir um banco de dados de modelo colunar é manter o esquema de tabelas e modificar a camada de armazenamento do SGBD. Com a modificação, o SGBD guardará os atributos em disco coluna por coluna (Figura 7), ao invés de armazená-los por linhas (Figura 8).

A diferença no armazenamento dos dados dessa abordagem para o particionamento vertical é que não é necessário armazenar um índice junto a cada atributo. No armazenamento por colunas, o *i-ésimo* atributo de uma coluna corresponderá exatamente com os demais *i-ésimos* atributos das outras colunas, o que economiza espaço em disco.

Nesse modelo, as consultas são geralmente executadas de forma mais rápida do que no particionamento vertical e o espaço em disco necessário para armazenar os registros pode ser reduzido, se utilizado algum método de compressão sobre os dados.

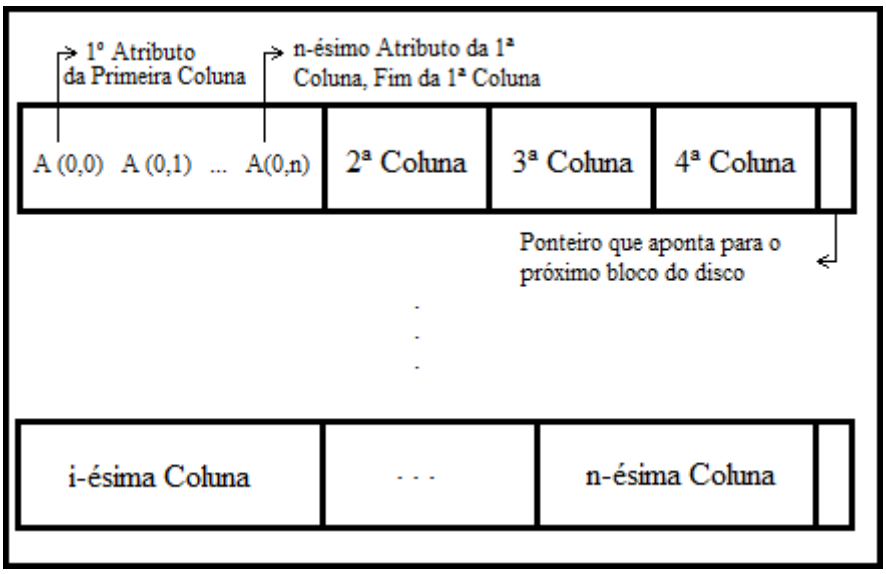

Figura 7: Forma de armazenamento em disco dos bancos de dados de modelo colunar (modificando a camada de armazenamento do modelo relacional)

A grande desvantagem dessa abordagem se dá na reconstrução das *tuplas*, quando acessadas e recuperadas do disco. De fato, o modelo de armazenamento por linhas o faz de forma mais fácil devido a armazenar a linha inteira contiguamente e o modelo de armazenamento por colunas precisar percorrer os blocos recuperando os atributos de cada coluna.

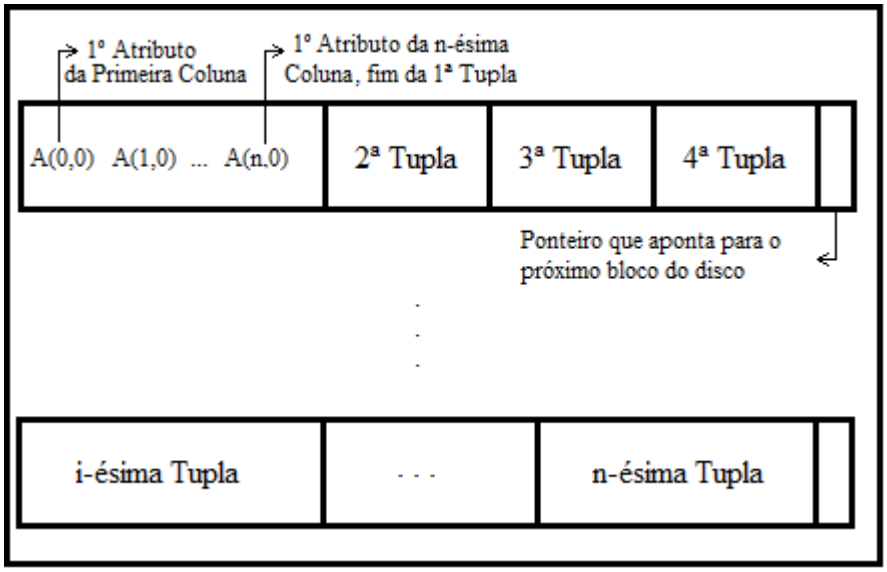

Figura 8: Forma de armazenamento em disco dos bancos de dados de modelo relacional

Harizopoulos et al., (2006) propõem um estudo comparativo sobre a arquitetura de armazenamento colunar e linear (relacional), apresentando o desempenho em questão de acesso a disco, memória e uso de CPU. É concluído que na maioria das vezes os SGBD de modelo colunar, no geral, fazem melhor uso da banda em disco que SGBD de modelo relacional.

4.3 Particionamento Vertical e Modificação da Camada de Armazenamento

Nessa abordagem, tanto o executor de consultas como a camada de armazenamento são modificados no SGBD. Stonebreaker et al., (2005) apresentam um estudo sobre esse modelo, apresentando o C-Store, que é um SGBD de modelo colunar que segue essa abordagem. É o método mais complexo de construir um SGBD de modelo colunar, porém, o que mais apresenta vantagens em tempo de consulta, visto que os dados podem ser armazenados de uma forma altamente compacta (Ver Seção 4.4) e o executor de consultas pode ser modificado para trabalhar sobre os dados compactados. Alguns resultados obtidos pelos autores são mostrados na Tabela 2 (foi utilizada uma versão simplificada do TPC-H [TPC-H, 2012] para os testes, descrita na Seção 4.7).

Tabela 2: Tempo de consulta, em segundos, para sete diferentes consultas utilizando o C-Store e um SGBD relacional comercial. Adaptada de Stonebreaker et al. (2005)

Consulta	C-Store	SGBD relacional
1	0.03	6.80
2	0.36	1.09
3	4.90	93.26
4	2.09	722.90
5	0.31	116.56
6	8.50	652.90
7	2.54	265.80

4.4 Métodos de Compressão

Existem diversos métodos de compressão que podem ser utilizados em bancos de dados, dentre os quais estão os apresentados por Ziv e Lempel (1977), um algoritmo universal para compressão de dados sequenciais: Cormak (1985), um método de compressão para operações relacionais, Westmann et al. (2000), que discutem quatro algoritmos simples de compressão, porém bastante interessantes, sendo eles compressão numérica, compressão de *strings*, compressão baseada em dicionário e compressão de valores nulos e Zukowski et al. (2006), um algoritmo de compressão desenvolvido para a escalabilidade de processadores modernos, utilizando a memória RAM como cache.

No entanto, nem todos os métodos são aplicáveis para todas as arquiteturas presentes nos bancos de dados, principalmente em se tratando de bancos de dados colunares. O ganho no desempenho da compressão depende não só apenas das propriedades dos dados, mas também da forma com que o processador de consultas manipula os atributos [Abadi, Madden e Ferreira, 2006].

No caso do processador de consultas, é necessário que este contenha um mecanismo que permita manipular os dados da forma em que foram comprimidos, ou mesmo que possa realizar uma descompressão para recuperar as informações no formato original, para daí estar apto a manipulá-las. Caso o processador de consultas realize uma descompressão antes de operar sobre os dados, o ganho obtido será medido não só pela redução de espaço em disco, mas também pelo tempo que levará para realizar essa operação. Métodos que gastam tempo

demasiado em descompressão podem não ser interessantes, principalmente quando o processamento sobre os dados é muito afetado.

A forma como os dados são armazenados em disco também influencia na qualidade da compressão. Bancos de dados de modelo colunar possuem vantagens sobre bancos de dados de modelo relacional nessa questão. Abadi (2008) exemplifica esse fato sugerindo que, pelo método de armazenamento colunar guardar informações por colunas, elas se tornam mais aptas à compressão, por informações semelhantes serem armazenadas sequencialmente. Um exemplo é a utilização do algoritmo *run-length*, a ser descrito na pag. 20.

Considere um banco de dados que contenha uma tabela com uma de suas colunas sendo "sexo" e seus possíveis valores sejam *M* ou *F*. Considerando que diversos registros tenham sido inseridos na tabela, um número grande de *M*s ou *F*s podem ser armazenados de forma sequencial (Figura 9(a)). Dessa maneira, o trabalho de ler cada *M* ou *F* de forma individual seria muito maior do que utilizar um contador que indique quantos *M*s ou *F*s seguidos aparecem de uma só vez (Figura 9(b)). Nota-se que 113 caracteres puderam ser reescritos em 101. Talvez não pareça um número tão significativo, mas pensando que possam existir milhões de registros em disco, então um número grande de espaço pode ser economizado.

(a)	(b)
. . . . M F M M M M M F M M M M M M F F F F F F F M F M M M F F F M M F M M M M F F M M F F F M M M M M M M M F M F F M M M M F M M F F F F M F F M M F F F F M M M F F M F M F M F M F F M M M F M M F F F F F F F F F M F F M M F M F M	. . . . M F @5M F @6M @8F M F @2M @3F @2M F @4M @2F @ 2M @3F @8M F M @2F @4M F @2M @4F M @2F @2M @4F @ 3M @2F M F M F M F M @2F @ 3M F @2M @9F M @2F @2M F M F M . . . .

Figura 9: Trecho da representação de armazenamento colunar em disco sem utilização do método run-length (a) e com a utilização do método run-length (b)

Há também outras possibilidades para otimizar ainda mais o algoritmo, como utilizar um caractere especial diferente para cada possível valor, (como # para *M* e @ para *F*) dessa forma cada cadeia codificada gastaria um caractere a menos. Pode-se também utilizar o critério de que apenas cadeias de tamanho superior a três devam ser codificadas. No exemplo acima, houve certa perda com a compressão, no sentido de que dois caracteres consecutivos necessitaram de três outros para codificá-los.

Abadi, Madden e Ferreira (2006) utilizam-se de quatro métodos de compressão no C-Store, apresentando significativo ganho de desempenho:

- ***Null Supression***: É uma variação da abordagem apresentada em Westmann et al. (2000), consistindo em deletar zeros e valores nulos nos dados e substituí-los por uma descrição de quantos eles eram e onde existiam. No caso de números inteiros, utiliza-se 4 bytes para sua representação (32 bits). Caso o número 4 necessite ser representado, por exemplo, 29 bits seriam utilizados para guardar zeros e apenas 3 bits, de fato, conteriam a representação do número 4. Utilizando a compressão, os zeros poderiam ser anulados e o número representado apenas nos 3 bits necessários.
- **Codificação por Dicionário:** Consiste em representar todos os possíveis valores de um atributo em uma estrutura separada. Por exemplo, se um atributo pode assumir os valores {1, 5, 15, 25, 50, 100}, estes podem ser representados no dicionário por 3 bits, codificando cada um dos possíveis valores para representa-los por um bit diferente. Este é outro exemplo de compressão em que o modelo colunar se sobressai em relação ao relacional. Abadi, Madden e Ferreira (2006) afirmam que os bancos de dados de modelo relacional tem a limitação de só poder mapear valores de atributos para uma única *tupla*, pois não têm a capacidade de combinar os atributos de diversos registros em uma única entrada se não incluir os demais atributos também.
- **Codificação *Run-length***: Quando um valor lido é repetido sequencialmente, torna-se mais conveniente contar quantas vezes esse valor é repetido ao invés de ter que ler cada cópia individualmente, economizando leitura em disco. Abadi, Madden e Ferreira (2006) utilizam a tripla (valor, posição inicial, tamanho) para indicar para qual valor do atributo foi aplicada a compressão, a posição inicial e o número de caracteres que foram comprimidos.
- **Codificação por Vetor de bit (*Bitmap*)**: Consiste em montar uma cadeia de bit associando o valor 1 à posição correspondente caso aquele valor apareça naquela posição e 0 caso não. Os autores exemplificam a partir dos dados: 1 1 3 2 2 3, representando-os da seguinte forma: cadeia de bit para o valor 1: 1100001, para o valor 2: 0001100 e para o valor 3: 0010010.

De uma maneira geral, métodos de compressão são muito úteis e capazes de incrementar consideravelmente o desempenho de um SGBD. No entanto, alguns métodos de compressão devem ser modificados para adaptar-se bem a certos SGBD, devido às diferenças de

arquiteturas entre esses sistemas. Como dito, dois principais componentes que influenciam no desempenho da compressão são o executor de consultas e a camada de armazenamento do SGBD. O executor de consultas deve estar apto a trabalhar com dados comprimidos ou conseguir descompactá-los para recuperar as informações originais e a camada de armazenamento deve substituir informações comprimidas por referências que possam transmitir a essência dos dados descomprimidos.

A compressão de dados é, com certeza, um fator fundamental ao trabalhar com uma grande quantidade de dados, devido à redução de espaço em disco e ao melhor desempenho obtido. Diversos SGBD colunares aplicam métodos de compressão, apresentando ótimos resultados em redução de espaço em disco. Esse fator se tornou um dos quesitos principais na escolha de um SGBD, principalmente para grandes empresas. Como dito, SGBD de modelo colunar podem comprimir informações com uma proporção maior do que os modelos relacionais, tornando-se uma importante vantagem sobre SGBD desse modelo.

4.5 Materialização

Da mesma forma com que existe uma metodologia para armazenar os dados em disco nos bancos de dados, também deve existir outra que recupere as informações armazenadas e as transforme novamente em *tuplas*. Essa operação é chamada de materialização ou reconstrução de *tuplas*. Abadi et al. (2007) apresentam duas formas de materialização:

- *Early Materialization* (*EM*): Consiste em adicionar uma coluna a uma *tupla* intermediária de saída, caso a coluna acessada seja requerida posteriormente por algum operador ou esteja incluída na saída da consulta. Metodologia adotada pelo modelo relacional de banco de dados.
- *Late Materialization* (*LM*): Consiste em não adicionar a coluna no mesmo instante em que é requerida. O executor de consultas espera um instante para que, primeiramente, cada predicado seja aplicado à sua respectiva coluna, e uma lista para cada coluna contendo as posições que atenderam ao predicado é gerada. Cada *i-ésimo* valor de cada coluna é comparado e apenas os *i-ésimos* atributos que atenderam a todos os predicados são adicionados à *tupla* de resposta. Essa metodologia é desenvolvida para o modelo colunar de banco de dados.

A Figura 10 apresenta um exemplo de base de dados para a tabela *Aluno* da Figura 1. A SQL (6) apresenta uma consulta sobre a tabela *Aluno*, selecionando os atributos *CPF* e *RG* de

cada *tupla* que possuir o atributo *Matricula* de valor maior que 2 e o atributo *Nome* começando com a letra *A*. As Figuras 11 e 12 exemplificam os métodos de materialização *EM* e *LM*, respectivamente, para a consulta (6) sobre a base de dados da Figura 10:

```
SELECT CPF, RG FROM ALUNO
WHERE Matricula > 2
AND Nome LIKE 'A%'                                    (6)
```

Pela Figura 11 nota-se que foi necessário gerar uma *tupla* para cada linha da tabela, mesmo para as linhas que não atenderam ao predicado, pois a verificação do predicado é feita após a montagem das *tuplas*.

É importante notar que, embora todas as *tuplas* tenham sido montadas na Figura 11, nem todas aparecerão na saída. Isso ocorre, pois esse método de materialização primeiramente monta as *tuplas* a partir de cada coluna requisitada na consulta e só depois verifica os predicados, descartando as que são desnecessárias à consulta (que não atenderam ao predicado).

TABELA ALUNO

Matricula	CPF	RG	Nome
1	678.Y31.X33-40	4.03X.894 Y	Adriano P. S.
2	6X4.616.Y86-83	2.Y77.26X	Roberto M. G.
3	483.704.8Y5-3X	91.X2Y.53Z 1	Augusto S. F.
4	Y87.828.28X-63	8.456.XY2 Z	Mariana F. B.
5	711.2X2.Y78-28	12.37X.Y23 4	Carlos A. E.
6	1XY.064.415-12	6.472.33X Y	Amanda C.

Figura 10: Exemplo de instâncias para a tabela Aluno, apresentada na Figura 1

Na Figura 12 é mostrado que isso não acontece utilizando a materialização *LM*. Primeiramente, as colunas referentes ao predicado são analisadas, verificando quais são as posições que o atendem. A partir das posições obtidas é que as *tuplas* são construídas, não necessitando criar *tuplas* desnecessárias.

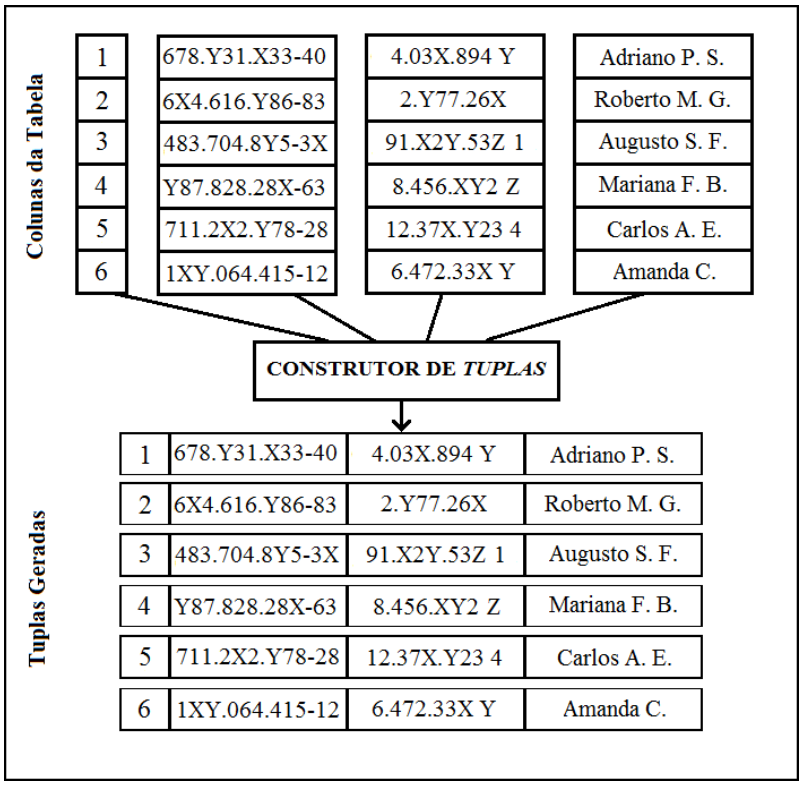

Figura 11: Tuplas geradas sobre o esquema da Tabela 10 para a consulta (6), utilizando a materialização EM

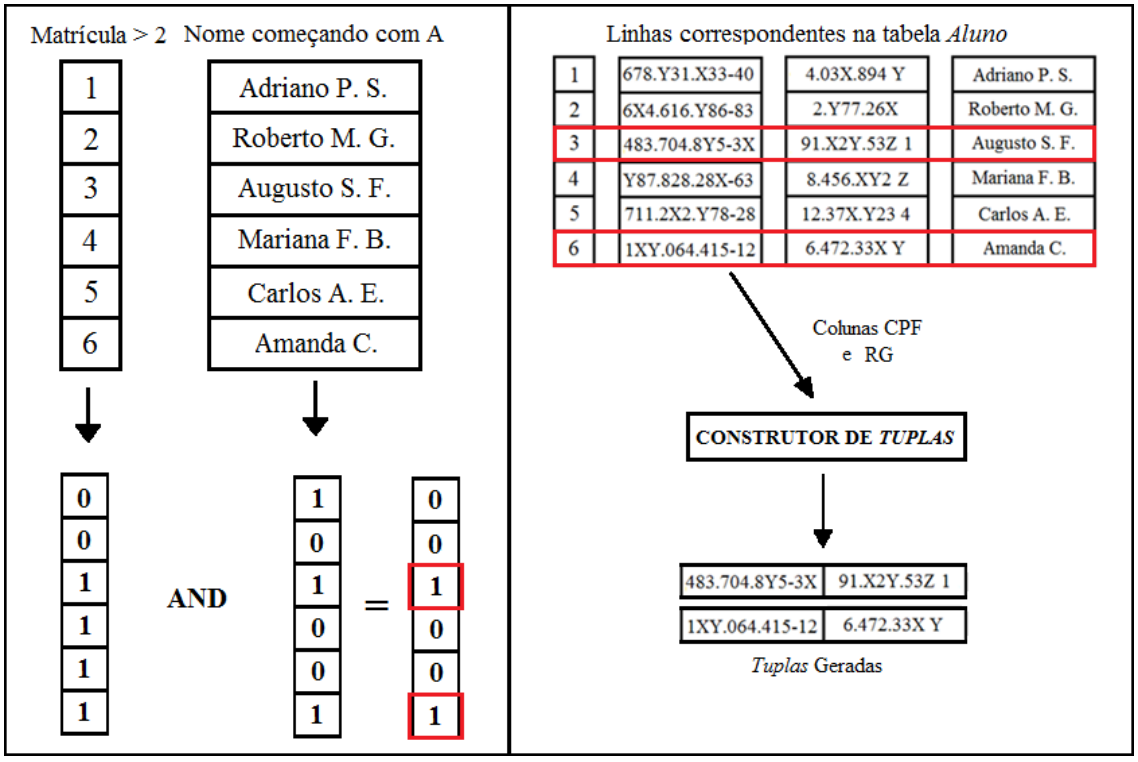

Figura 12: Tuplas geradas sobre o esquema da Tabela 10 para a consulta (6), utilizando a materialização LM

4.6 Iteração de Bloco

De acordo com Zukowski et al. (2005), o executor de consultas de SGBD relacionais como o MySQL 4.1 [Oracle, 2012b] necessitam de tempo demasiado para interpretar *tuplas* antes de fornecer o resultado de uma consulta. Isso ocorre pois todas as *tuplas* devem ser percorridas e interpretadas uma-a-uma, lendo-se atributos de forma desnecessária e utilizando um alto número de instruções de CPU para verificar todos os registros.

SGBD de modelo colunar, possuem uma estrutura que minimiza este problema. Visto que armazenam em disco os elementos de cada coluna de forma contígua, é possível armazenar todos os valores de uma coluna em um vetor e manda-lo para o executor de consultas com apenas uma instrução de CPU. Dessa forma, o executor de consultas pode operar sobre o vetor contendo todos os elementos da coluna lida de uma só vez, ao invés de requisitar uma instrução para cada *tupla*.

4.7 Aplicações para o Modelo Colunar

Como dito no Capítulo 3, a escalabilidade *versus* desempenho é um fator fundamental para busca de novas estratégias de armazenamento de dados. Harizopoulos et al. (2006) e Stonebreaker et al. (2005) discutem a questão de bancos de dados colunares serem otimizados à leitura (*read-optimized*), enquanto bancos de dados relacionais são otimizados à escrita (*write-optimized*), tornando-os uma boa alternativa às aplicações que possuem grande densidade de dados e que são frequentemente requeridos para leitura.

Data warehouses (DW) são exemplos desse tipo de aplicação, possuem imensa quantidade de informações e são requeridas a todo tempo. Inmon (2000) afirma que DW é um esquema orientado a tema/assunto, integrado, variante de tempo e não volátil de coleção de dados, utilizada no apoio de gestão do processo de tomada de decisão. De acordo com Golfarelli, Maio e Rizzi (1998), *data wharehouse* é um repositório de dados que compreende ferramentas de aplicação, arquiteturas, serviços de informação e infra-estruturas de comunicação para sintetizar informações úteis para tomada de decisão.

DW tendem a ser naturalmente grandes (grande quantidade de dados) por não apagarem ou modificarem informação alguma, armazenando diariamente novas grandes massas de dados. Apesar de esses repositórios receberem grande número de informações diariamente, a velocidade de escrita não é um fator preocupante. A estratégia de inserção é armazenar,

durante o período de maior atividade, toda informação necessária a ser guardada no DW e inseri-la de uma só vez, em algum período em que haja o menor número de acessos possível. Por essa razão, DW são caracterizados como sistemas otimizados à leitura.

A Figura 13 mostra um exemplo de esquema em estrela [Red Brick Systems, 1995], chamado SSB, uma variação do esquema TPC-H, descrito em Abadi (2008) e Neil, Neil e Chen (2009). O esquema em estrela é um dos meios de estruturação de DWs e, de acordo com Kimball (2011), sistemas de suporte à decisão como DWs são frequentemente construídos desta forma, o que facilita a manipulação dos dados no DW. Esse tipo de esquema se caracteriza por possuir uma tabela fato, que é a tabela que armazena a maior quantidade de informação e várias tabelas de dimensão, que possuem menos dados que a tabela fato e também crescem consideravelmente mais lentamente. A tabela fato possuirá chaves estrangeiras para as tabelas de dimensão e nenhuma tabela de dimensão é unida a outra. Essas possuem apenas chaves primárias, que servem para "ligar-se" à tabela fato.

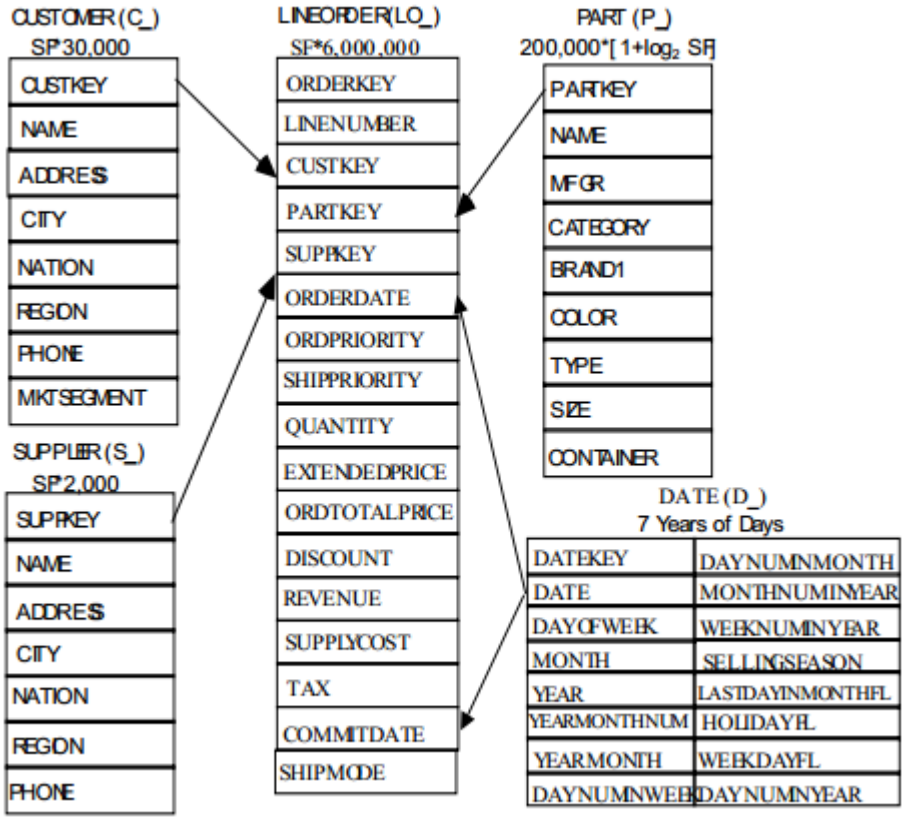

Figura 13: SSB Esquema - variação do esquema TPC-H.

Adaptada de [Neil, Neil Chen e 2009]

De acordo com Weininger (2002), o esquema em estrela gera uma navegação mais intuitiva pelo usuário e as consultas geralmente consistem em fazer uma filtragem nas tabelas de dimensão, unir as tabelas de dimensão à tabela fato e fazer algum agrupamento ou ordenação. Esse tipo de operação realiza diversas junções (*joins*), que é um tipo de operação de alto custo de processamento. Esse é um ponto em que os bancos de dados colunares podem obter vantagem sobre os relacionais. Abadi (2008) e Abadi, Madden e Hachem (2008) propõe uma metodologia de execução de *joins* para o modelo colunar de banco de dados aplicado a consultas em modelos do tipo estrela (Seção 4.7.1).

É importante ressaltar que, quando mais de um atributo é requerido em uma consulta, algum mecanismo de construção de *tuplas* deve ser utilizado. Considerando que os tipos de aplicações discutidas nesta seção utilizam consultas que geralmente selecionam mais de um atributo e possuem predicados, o método de materialização adotado nos bancos de dados de modelo colunar garantem outra vantagem sobre os relacionais (ver Seção 4.5).

4.7.1 *Join* Otimizado a Aplicações em Esquema Estrela

O uso de *joins* nos bancos de dados de modelo colunar tem o mesmo objetivo que nos bancos de dados relacionais, que segundo [Mishra e Eich, 1992], é combinar registros de duas relações diferentes, baseado em alguma informação em comum, como uma operação de produto cartesiano. Por exemplo, seguindo o esquema da Figura 1, deseja-se exibir o código de todos os alunos matriculados em um determinado curso. A entidade *Curso* possui uma chave estrangeira para a entidade *Aluno*, que é o atributo "Aluno_Matricula", ou seja, o atributo referente à matrícula do aluno é a informação em comum nesse caso, e a junção seria feita a partir deste atributo.

Há diversas maneiras de implementação da operação de junção, utilizando diferentes estruturas. Mishra e Eich (1992) apresentam os modos de implementação de junção referentes a banco de dados relacionais, sendo as principais estruturas utilizadas: *Nested-Loops*, *Sort-Merge*, tabelas *Hash,* Índices, *B-Trees* e *T-Trees*. De acordo com o autor, *joins* executados a partir de índices fornecem bons resultados em consultas complexas, que envolvem várias junções e seleções. O'Neil e Graefe (1995) apresentam o conceito da utilização de índices *Bitmap* no auxílio de operações de junção de tabelas, enfatizando a otimização das consultas sobre o esquema em estrela. Abadi (2008) e Abadi, Madden e Hachem (2008) põe em prática a utilização de índices *Bitmap* para operações de *joins*, adaptando o conceito apresentado por

O'Neil e Graefe (1995) para o modelo colunar (*Invisible Join*), descrito em três partes principais:

- 1ª Fase: Cada predicado é aplicado à sua tabela dimensão referente, informando quais são as chaves que o satisfazem. Essas chaves serão utilizadas para construir uma tabela *Hash*, para cada predicado, que indicará se uma chave particular satisfaz ou não ao predicado.
- 2ª Fase: Cada tabela *Hash* é utilizada para extrair as posições, de cada coluna da tabela fato que é chave estrangeira para as tabelas de dimensão utilizadas na Fase 1, que possuem a chave referente a algum dos valores encontrados na Fase 1. Uma nova tabela *Hash* (diga-se *hf*, para facilitar o entendimento) é criada, indicando quais chaves da tabela fato atendem a todos os predicados das tabelas *Hash* vindas da Fase 1.
- 3ª Fase: Para cada coluna (que guarda uma chave) da tabela *Hash hf* que atendeu aos requisitos impostos na Fase 2 e que é chave estrangeira para uma tabela dimensão requisitada na consulta, a chave da tabela dimensão é retornada e adicionada à resposta da consulta.

Para uma compreender os passos descritos acima de uma forma mais simples, considere o exemplo abaixo, descrito por Abadi, Madden e Hachem (2008), referente ao esquema da Figura 13:

```
SELECT c.nation, s.nation, d.year,
SUM(lo.revenue) AS revenue
FROM customer AS c, lineorder AS lo,
supplier AS s, dwdate AS d
WHERE lo.custkey = c.custkey
AND lo.suppkey = s.suppkey
AND lo.orderdate = d.datekey
AND c.region = 'ASIA'
AND s.region = 'ASIA'
AND d.year >= 1992 AND d.year <= 1997
GROUP BY c.nation, s.nation, d.year
ORDER BY d.year asc, revenue desc;                    (7)
```

De acordo com a consulta (7), há três tabelas de dimensão que serão unidas à tabela fato, *customer*, *supplier* e *dwdate*. Dessa forma, de acordo com a Fase 1, três tabelas de dimensão vão ser consultadas, verificando quais chaves de cada uma atendem ao predicado. Para uma base de dados qualquer, suponha que as tabelas da Figura 14 representem as tabelas de dimensão requisitadas. O próximo passo é criar uma tabela *Hash* para cada coluna da tabela fato com chave estrangeira correspondente às tabelas de dimensão utilizadas na Fase 1.
Portanto, haverão três tabelas *Hash*, uma para a coluna de chave estrangeira à tabela *customer*, outra para a coluna de chave estrangeira à tabela *supplier* e outra para a coluna de chave estrangeira à tabela *date* (Figura 15). O próximo passo da Fase 2 é montar a tabela *Hash hf*, que representará as posições da tabela fato em que todas as restrições foram satisfeitas. Isso é feito aplicando a operação *AND* sobre os *bits* de cada tabela sobre cada posição correspondente (Figura 16).

Depois de obtida a tabela *hf*, a terceira Fase consiste em recuperar, a partir das chaves da tabela fato, as chaves nas tabelas de dimensão sobre as posições da tabela *hf* que contém valor 1 (*true*) e que são requisitadas para resposta da consulta (Figura 14).

Assim, a saída da consulta são as 3 colunas resposta apresentadas na Figura 17, junto à soma da coluna *revenue* da tabela fato.

Vale ressaltar que o uso de *invisible joins* foi utilizado apenas no C-Store, não sendo o método padrão de *join* dos bancos colunares, porém, é aplicável para quaisquer bancos de dados desse modelo que armazenem os dados em disco por colunas.

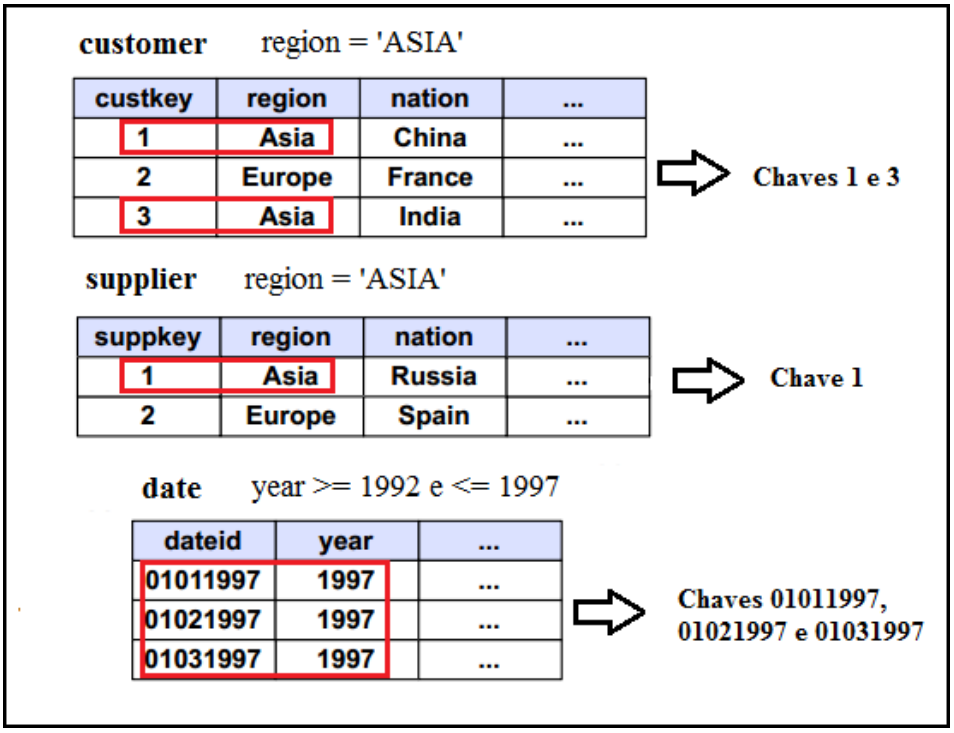

Figura 14: Primeira fase do invisible join referente à consulta (7). Adaptada de Abadi, Madden e Hachem (2008)

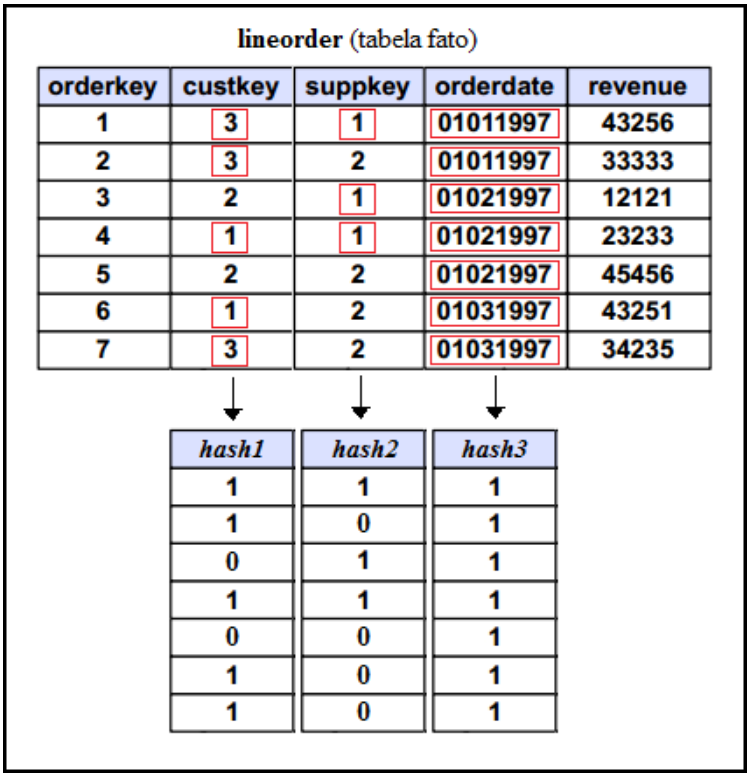

Figura 15: Segunda fase, primeiro passo, do invisible join referente à consulta (7).

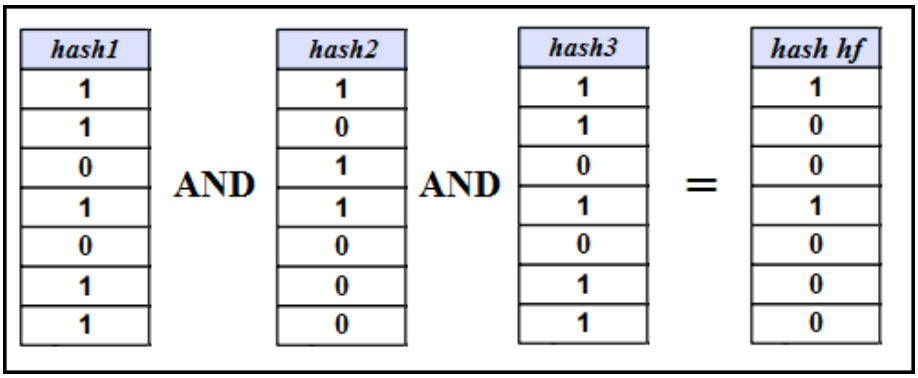

Figura 16: Segunda fase, segundo passo, do invisible join referente à consulta (7).

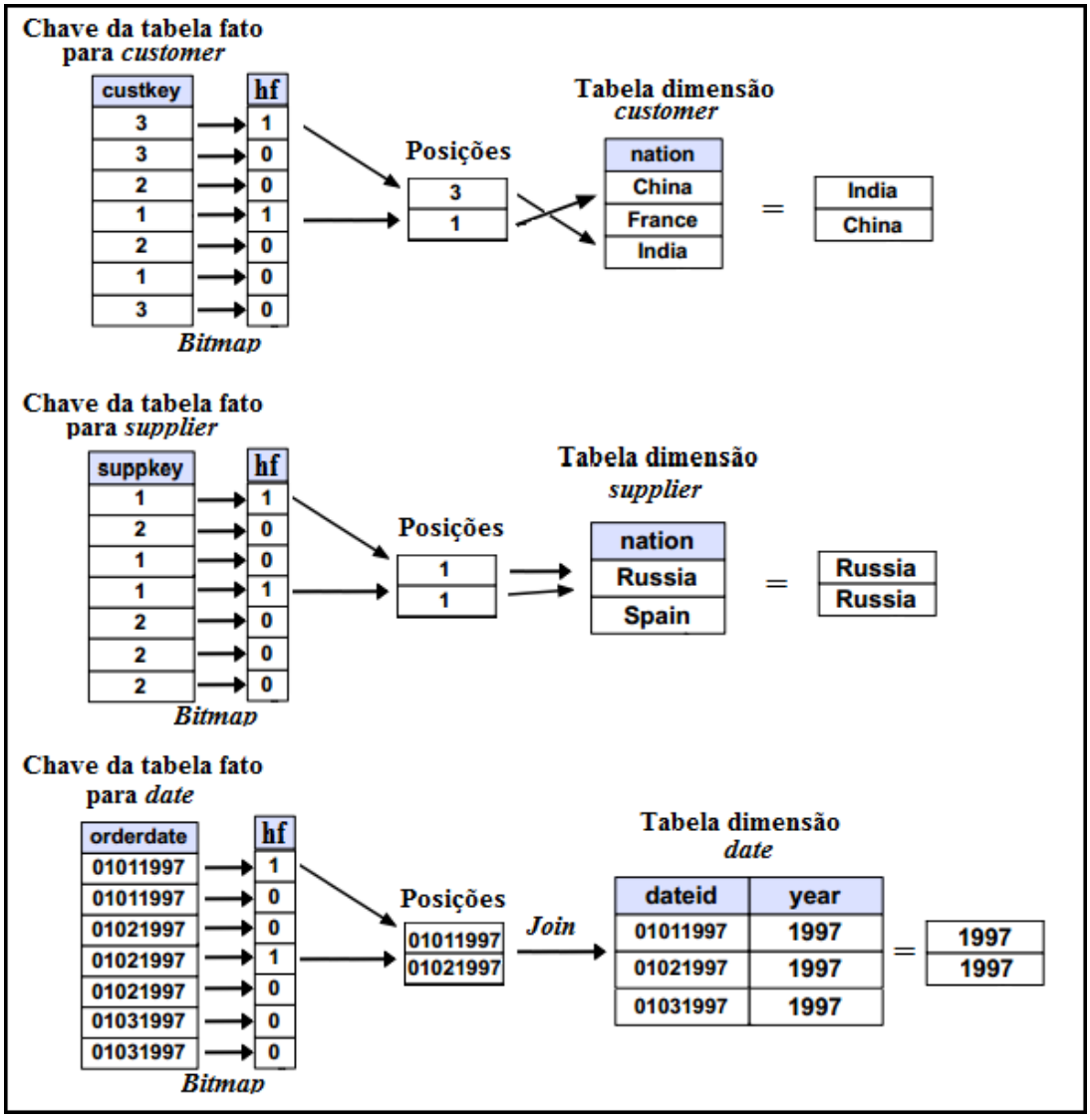

Figura 17: Terceira fase do invisible join referente à consulta (7).

Capítulo 5

Avaliação Experimental

A partir da comparação teórica e do levantamento bibliográfico construídos nos capítulos anteriores, chegou-se a conclusão de que, para que uma comparação prática "proporcional" seja feita, é essencialmente necessário que os SGBD manipulem os mesmos tipos de dados e utilizem a mesma linguagem de consulta. Portanto, pesquisou-se na literatura SGBD de modelo colunar, visando encontrar os que atendessem a esses requisitos de comparação. A Tabela 1, apresentada no Capítulo 4, descreve as principais características de diversos SGBD de modelo colunar encontrados.

Como dito no Capítulo 1, SGBD de modelo relacional utilizam SQL como linguagem de consulta. Por tanto, a comparação deve ser feita a partir de SGBD de modelo colunar que também utilizem a SQL como linguagem de consulta, além de que apenas bancos de dados de licença livre, comunitária ou de teste poderão ser utilizados por definição de projeto.

Os SGBD MonetDB, Infobright, InfiniDB e C-Store atendem a essas características. Quanto aos SGBD relacionais, escolheu-se o PostgreSQL, por ser um dos SGBD de licença livre mais comuns e utilizado e o MySQL, por ser parte da composição do Infobright e InfiniDB, possibilitando assim verificar quais os pontos de destaque da abordagem colunar sobre um SGBD relacional.

Para realizar a conexão com os servidores dos bancos de dados foi decidido utilizar o SquirreL SQL v.3.3.0 [SquirreL, 2012], um cliente genérico, para que o tempo de consulta não seja influenciado pela interface do *client* dos SGBD e todos estejam em uma aplicação padrão. Os SGBD (com exceção do C-Store, por não ser compatível com este sistema operacional) foram instalados no servidor do Laboratório de Engenharia de Software e Banco de Dados/Laboratório de Inteligência Artificial e Computação Gráfica (LEB/LIC), no bloco F da Unioeste - Universidade Estadual do Oeste do Paraná - do campus de Cascavel, que possui os seguintes recursos:

- Sistema Operacional: Windows Server 2008 R2 Datacenter. Service Pack 1 (64 Bits).
- Processador: 2 Processadores Intel® Xeon® E5620 @ 2.4 GHz 2.4 GHz.
- Memória RAM: 16 GB.

A conexão com os SGBD foi estabelecida por meio do driver JDBC de cada um deles, sendo que todos implementam acesso concorrente. A ideia inicial era utilizar o padrão TPC-H para realizar os testes com os SGBD, contudo, alguns fatores impossibilitaram sua adoção:

- O fato de cada tabela do modelo utilizar pelo menos uma restrição de chave, sendo que o InfiniDB e o Infobright não utilizam restrições de chave;
- Não permitir uso de driver JDBC diferenciado para cada SGBD a ser testado;

Além das restrições impostas pelo TPC-H, há que considerar que o modelo SSB (Seção 4.7) possui uma modelagem mais interessante de navegação sobre os dados, fornecendo uma estrutura denormalizada, o que, de acordo com Kimball (2011), é padrão em DW. Por tanto, optou-se por utilizar esse modelo. A descrição das tabelas e consultas do SSB é apresentada no Apêndice A. Os testes foram realizados em diferentes cenários, cada um contendo um diferente número de registros em cada tabela. O primeiro teste foi realizado com os números informados na descrição do modelo SSB (SF=1). Em seguida, a escalabilidade foi aumentada para 2 (SF=2), 5 (SF=5) e 10 (SF=10) vezes, para averiguar o comportamento dos SGBD com o aumento no número de informações.

As consultas utilizadas foram as descritas pelos autores do SSB (Apêndice A), sendo empregado o processo de execução denominado *power test*, que executa todas as consultas de uma só vez, calculando o tempo gasto em CPU, o tempo de construção da saída (tempo de leitura do *ResultSet* do JDBC mais o tempo de armazenamento da saída no SquirreL) e o tempo total gasto em cada consulta. Também foi calculada a taxa de consultas por hora (*Cph*), descrita através da fórmula (8) pelo TPC-H, adaptado para as consultas do SSB (*CphSSB*) conforme (9).

$$Cph = \frac{3600 * SF}{\sqrt[24]{\prod_{i=1}^{i=22} QI(i,0) * \prod_{j=1}^{j=2} RI(j,0)}} \quad (8)$$

$$CphSSB = \frac{3600 * SF}{\sqrt[15]{\prod_{i=1}^{i=13} QI(i,0) * \prod_{j=1}^{j=2} RI(j,0)}} \quad (9)$$

Em (8) e (9), *QI* (*i*,0) representa o tempo consumido para executar a consulta *QIi*, *RI* (*j*,0) representa o tempo consumido para executar a função de *refresh RIj* (explicada a seguir) e *SF* é o fator de escala.

No *power test* descrito no SSB, há duas funções apresentadas no modelo, uma executada previamente (RF1) e outra posteriormente às consultas (RF2), sendo elas funções de inserção de linhas na tabela *lineorder* e de deleção das linhas inseridas, respectivamente. Essas funções foram construídas em Java, utilizando o mesmo *driver* de conexão de cada SGBD conectado ao SquirreL. A execução do *power test* se dá da seguinte maneira:

```
Função de inserção na tabela LINEORDER – RF1
        Execução das 13 consultas
Função de remoção das linhas inseridas – RF2
```

Optou-se por seguir estes passos de duas formas diferentes:

- Teste 1: Executar cada etapa imediatamente após o término da etapa anterior. Dessa forma há a possibilidade de verificar o comportamento de cada SGBD na inserção, na consulta e na remoção de dados, sem que uma etapa interfira na outra.
- Teste 2: Executar a função RF1 e as 13 consultas, uma a uma, simultaneamente (aplicação paralela) e, imediatamente após o término da função RF1, executar a função RF2. Dessa forma há a possibilidade de verificar o comportamento de cada SGBD com requisições simultâneas.

O InfiniDB apresentou problemas no cálculo de operações com plano cartesiano, não sendo possível incluí-lo nos testes, visto que todas as consultas possuem esse tipo de operação. O povoamento dos bancos de dados se deu a partir da ferramenta *dbgen*, padrão do SSB. Todas as variáveis numéricas foram tratadas como *Integer* e as de texto como *Character Varying*.

Os SGBD foram testados em questão do tempo de carregamento de registros, com o comando *infile* (visto que os arquivos gerados pelo *dgben* atendem a este padrão de inserção),

tempo de consulta para cada consulta e no espaço em disco necessário para armazenar todas as informações de cada banco de dados, para cada cenário.

Pelo fato do Infobright não fazer uso de chaves (tanto primárias quanto estrangeiras), optou-se por realizar dois testes para cada cenário, um inserindo todas as restrições de chave no PostgreSQL, Monet e MySQL e outro não utilizando nenhuma.

5.1 Padrão de Teste 1

Os testes apresentados nessa seção foram executados conforme descrito o Teste 1. O MySQL levou muito tempo para executar as consultas (mais de 24 horas), com exceção das três primeiras e, por isso, apenas estas foram mantidas nos testes, apresentando esses resultados separadamente dos demais. Escolheu-se essa abordagem, pois, embora as três primeiras consultas tenham executado, o tempo consumido na execução foi consideravelmente mais alto do que em todos os outros SGBD, o que torna o gráfico pouco detalhado, visto que haveria muitos dados discrepantes. Também, a partir do Cenário 3, a inserção dos dados no MySQL excedeu 96 horas e decidiu-se então cancelar os testes com esse SGBD nos dois últimos cenários.

5.1.1 Cenário 1 – SF=1

No Cenário 1, o número de registros referentes à cada tabela é o apresentado na Figura 13, utilizando SF=1. Dessa forma, as tabelas Part, Date, Lineorder, Customer e Supplier continham, respectivamente, 200.000, 2.556, 6.001.171, 30.000 e 2.000 registros.

Tabela 3: Tempo de inserção dos dados no Cenário 1 (SF=1), com restrição de chaves

	Monet	**Infobright**	**MySQL**	**PostgreSQL**
Supplier	0,171	0,125	0,171	0,016
Customer	1,997	0,438	0,562	0,437
Date	3,104	0,157	0,171	0,032
Part	2,793	0,203	3,136	3,900
Lineorder	73,929	1,014	13476,598	657,839

A Tabela 3 apresenta o tempo consumido, em segundos, com a inserção dos dados deste Cenário, para cada SGBD, utilizando as restrições de chave (com exceção do Infobright).

O tempo consumido para as três primeiras consultas é apresentado na Tabela 4 (Total/CPU/Saída).

Tabela 4: Tempo consumido para as três primeiras consultas do Cenário 1, SF=1, (Total/CPU/Saída), com restrição de chaves

	Monet	**Infobright**	**MySQL**	**PostgreSQL**
RF1	3370,662	1985,345	509,988	13,052
1.1	0,468/0,296/0,172	0,499/0,468/0,031	1191,656/1191,625/0,031	2,558/2,543/0,015
1.2	0,374/0,218/0,156	0,234/0,218/0,016	101,509/101,478/0,031	2,496/2,481/0,015
1.3	0,452/0,296/0,156	0,218/0,203/0,015	23,088/23,073/0,015	2,465/2,434/0,031
RF2	0,016	0,031	3,978	0,016

Para o conjunto completo das consultas, é apresentado o tempo consumido na Tabela 5 (Total/CPU/Saída). As Tabelas 6, 7 e 8 representam as mesmas características das Tabelas 3, 4 e 5, sem a utilização das restrições de chaves e a Figura 18 representa essas informações de forma gráfica.

O número de linhas retornadas para cada consulta, ordenadas de acordo com a Tabela 5, foi 1, 1, 1, 280, 56, 7, 150, 599, 24, 4, 35, 100, 339, respectivamente.

Tabela 5: Tempo consumido para cada consulta do Cenário 1, SF=1, (Total/CPU/Saída) com restrições de chaves, para o Teste 1

	Monet	**Infobright**	**PostgreSQL**
RF1	3370,662	1985,345	13,052
1.1	0,468/0,296/0,172	0,499/0,468/0,031	2,558/2543/0,015
1.2	0,374/0,218/0,156	0,234/0,218/0,016	2,558/2,543/0,015
1.3	0,452/0,296/0,156	0,218/0,203/0,015	2,496/2,481/0,015
2.1	0,687/0,266/0,421	0,905/0,858/0,047	2,808/2,792/0,016
2.2	0,561/0,125/0,436	0,702/0,670/0,032	2,137/2,106/0,031
2.3	0,546/0,094/0,452	0,514/0,499/0,015	1,779/1,747/0,032
3.1	1,139/0,531/0,608	0,702/0,687/0,015	3,010/2,979/0,031
3.2	0,764/0,156/0,608	0,500/0,453/0,047	1,935/1,904/0,031
3.3	0,593/0,109/0,484	0,702/0,686/0,016	1,778/1,747/0,031
3.4	0,562/0,094/0,468	0,405/0,390/0,015	1,888/1,857/0,031
4.1	0,920/0,483/0,437	0,843/0,827/0,016	2,995/2,964/0,031
4.2	1,014/0,468/0,546	0,780/0,764/0,016	2,714/2,683/0,031
4.3	0,842/0,234/0,608	0,577/0,546/0,031	2,013/1,997/0,016
RF2	0,016	0,031	0,016

Tabela 6: Tempo de inserção dos dados no Cenário 1, SF=1, sem restrição de chaves

	Monet	Infobright	MySQL	PostgreSQL
Supplier	1,669	0,125	0,079	0,001
Customer	2,013	0,438	0,532	0,376
Date	3,105	0,157	0,095	0,016
Part	2,825	0,203	3,277	3,823
Lineorder	43,869	1,014	152,039	85,006

Tabela 7: Tempo consumido para as três primeiras consultas do Cenário 1, SF=1, (Total/CPU/Saída) sem restrição de chaves

	Monet	Infobright	MySQL	PostgreSQL
RF1	291,915	1985,345	468,889	9,687
1.1	0,250/0,172/0,078	0,499/0,468/0,031	31,716/31,684/0,032	3,225/3,193/0,032
1.2	0,250/0,140/0,110	0,234/0,218/0,016	10,311/10,280/0,031	2,995/2,979/0,016
1.3	0,250/0,156/0,094	0,218/0,203/0,015	9,688/9,672/0,016	2,090/2,075/0,015
RF2	0,047	0,031	13,650	1,326

Tabela 8: Tempo consumido para cada consulta do Cenário 1, SF=1, (Total/CPU/Saída) sem restrições de chaves, para o Teste 1

	Monet	Infobright	PostgreSQL
RF1	291,915	1985,345	9,687
1.1	0,250/0,172/0,078	0,499/0,468/0,031	3,225/3,193/0,032
1.2	0,250/0,140/0,110	0,234/0,218/0,016	2,995/2,979/0,016
1.3	0,250/0,156/0,094	0,218/0,203/0,015	2,090/2,075/0,015
2.1	0,343/0,109/0,234	0,905/0,858/0,047	2,386/2,355/0,031
2.2	0,281/0,094/0,187	0,702/0,670/0,032	2,075/2,060/0,015
2.3	0,312/0,062/0,250	0,514/0,499/0,015	1,701/1,669/0,032
3.1	0,405/0,140/0,265	0,702/0,687/0,015	2,964/2,932/0,032
3.2	0,359/0,063/0,296	0,500/0,453/0,047	1,950/1,918/0,032
3.3	0,328/0,047/0,281	0,702/0,686/0,016	1,653/1,638/0,015
3.4	0,296/0,047/0,249	0,405/0,390/0,015	1,545/1,529/0,016
4.1	0,375/0,172/0,203	0,843/0,827/0,016	2,589/2,574/0,015
4.2	0,468/0,172/0,296	0,780/0,764/0,016	2,605/2,590/0,015
4.3	0,421/0,125/0,296	0,577/0,546/0,031	1,935/1,904/0,031
RF2	0,047	0,031	1,326

5.1.2 Cenário 2 – SF=2

No Cenário 2, o número de registros referentes à cada tabela é o apresentado na Figura 13, utilizando SF=2. Dessa forma, as tabelas Part, Date, Lineorder, Customer e Supplier

continham, respectivamente, 400.000, 2.556, 11.998.051, 60.000 e 4.000 registros. A Tabela 9 apresenta o tempo consumido, em segundos, com a inserção dos dados deste cenário, para cada SGBD, utilizando as restrições de chave (com exceção do Infobright). O tempo consumido para as três primeiras consultas é apresentado na Tabela 10 (Total/CPU/Saída). Para o conjunto completo das consultas, o tempo consumido é apresentado na Tabela 11 (Total/CPU/Saída).

As Tabelas 12, 13 e 14 representam as mesmas características das Tabelas 9, 10 e 11, sem a utilização das restrições de chaves e a Figura 18 representa essas informações de forma gráfica. O número de linhas retornadas para cada consulta, ordenadas de acordo com a Tabela 14, foi 1, 1, 1, 280, 56, 7, 150, 600, 24, 2, 35, 100, 529, respectivamente.

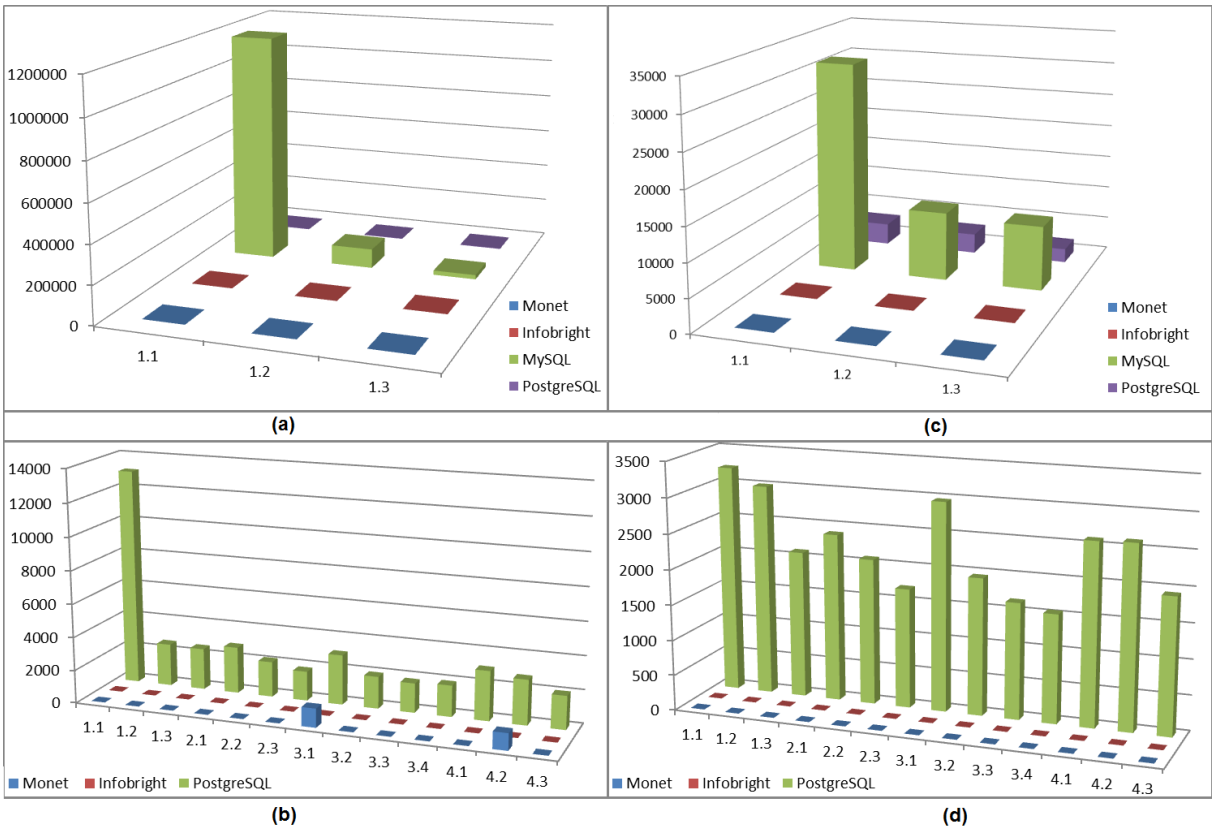

Figura 18: Gráficos representando (a) Tabela 4, (b) Tabela 5, (c) Tabela 6, (d) Tabela 7

Tabela 9: Tempo de inserção (em segundos) dos dados no Cenário 2, SF=2, com restrição de chaves

	Monet	**Infobright**	**MySQL**	**PostgreSQL**
Supplier	1,717	0,812	0,390	0,033
Customer	2,137	0,905	4,072	0,797
Date	3,104	0,156	0,469	0,032
Part	3,433	2,947	27,145	9,049
Lineorder	144,285	116,923	45019,902	1408,932

Tabela 10: Tempo consumido para as três primeiras consultas do Cenário 2, SF=2, (Total/CPU/Saída), com restrição de chaves

	Monet	**Infobright**	**MySQL**	**PostgreSQL**
RF1	10958,132	2602,600	3178,360	27,863
1.1	0,951/0,811/0,14	1,030/0,998/0,032	102,773/102,679/0,094	7,207/7,145/0,062
1.2	0,843/0,733/0,11	0,234/0,219/0,015	63,336/63,305/0,031	5,070/5,039/0,031
1.3	0,811/0,670/0,141	0,297/0,281/0,016	65,333/65,301/0,032	4,680/4,649/0,031
RF2	0,218	0,015	57,002	0,094

Tabela 11: Tempo consumido para cada consulta do Cenário 2, SF=2, (Total/CPU/Saída) com restrições de chaves, para o Teste 1

	Monet	**Infobright**	**PostgreSQL**
RF1	10958,132	2602,600	27,863
1.1	0,951/0,811/0,14	1,030/0,998/0,032	7,207/7,145/0,062
1.2	0,843/0,733/0,11	0,234/0,219/0,015	5,070/5,039/0,031
1.3	0,811/0,67/0,141	0,297/0,281/0,016	4,680/4,649/0,031
2.1	1,388/0,983/0,405	1,560/1,528/0,032	6,053/6,006/0,047
2.2	1,061/0,655/0,406	1,186/1,170/0,016	5,101/5,070/0,031
2.3	0,998/0,608/0,39	0,920/0,905/0,015	4,103/4,072/0,031
3.1	1,514/0,983/0,531	1,482/1,451/0,031	6,069/6,022/0,047
3.2	1,232/0,702/0,530	0,843/0,796/0,047	4,180/4,134/0,046
3.3	1,045/0,608/0,437	0,842/0,811/0,031	3,495/3,464/0,031
3.4	0,983/0,577/0,406	0,686/0,671/0,015	3,791/3,759/0,032
4.1	1,825/1,451/0,374	1,389/1,358/0,031	6,427/6,396/0,031
4.2	1,810/1,264/0,546	1,388/1,357/0,031	5,491/5,444/0,047
4.3	1,419/0,998/0,421	1,077/1,030/0,047	3,916/3,869/0,047
RF2	0,218	0,015	0,094

Tabela 12: Tempo de inserção dos dados no Cenário 2, SF=2, sem restrição de chaves

	Monet	**Infobright**	**MySQL**	**PostgreSQL**
Supplier	1,717	0,812	0,188	0,172
Customer	2,294	0,905	4,420	2,637
Date	3,338	0,156	0,220	0,063
Part	3,839	2,947	28,189	14,727
Lineorder	113,398	116,923	936,829	424,462

Tabela 13: Tempo consumido para as três primeiras consultas do Cenário 2, SF=2, (Total/CPU/Saída), sem restrição de chaves

	Monet	**Infobright**	**MySQL**	**PostgreSQL**
RF1	1201,418	2602,600	3624,837	149,776
1.1	0,421/0,312/0,109	1,030/0,998/0,032	158,684/158,653/0,031	4,868/4,836/0,032
1.2	0,296/0,203/0,093	0,234/0,219/0,015	66,019/66,003/0,016	4,820/4,789/0,031
1.3	0,343/0,234/0,109	0,297/0,281/0,016	62,447/62,416/0,031	4,337/4,306/0,031
RF2	0,780	0,015	192,249	2,465

Tabela 14: Tempo consumido para cada consulta do Cenário 2, SF=2, (Total/CPU/Saída) sem restrições de chaves, para o Teste 1

	Monet	**Infobright**	**PostgreSQL**
RF1	1201,418	2602,600	149,776
1.1	0,421/0,312/0,109	1,030/0,998/0,032	4,868/4,836/0,032
1.2	0,296/0,203/0,093	0,234/0,219/0,015	4,820/4,789/0,031
1.3	0,343/0,234/0,109	0,297/0,281/0,016	4,337/4,306/0,031
2.1	0,422/0,125/0,297	1,560/1,528/0,032	5,413/5,366/0,047
2.2	0,390/0,109/0,281	1,186/1,170/0,016	4,898/4,867/0,031
2.3	0,374/0,093/0,281	0,920/0,905/0,015	3,900/3,869/0,031
3.1	0,562/0,156/0,406	1,482/1,451/0,031	6,303/6,272/0,031
3.2	0,499/0,109/0,390	0,843/0,796/0,047	3,776/3,744/0,032
3.3	0,437/0,062/0,375	0,842/0,811/0,031	3,510/3,478/0,032
3.4	0,468/0,093/0,375	0,686/0,671/0,015	3,525/3,494/0,031
4.1	0,624/0,327/0,297	1,389/1,358/0,031	6,193/6,162/0,031
4.2	0,639/0,265/0,374	1,388/1,357/0,031	6,131/6,100/0,031
4.3	0,593/0,234/0,359	1,077/1,030/0,047	4,212/4,181/0,031
RF2	0,078	0,015	2,465

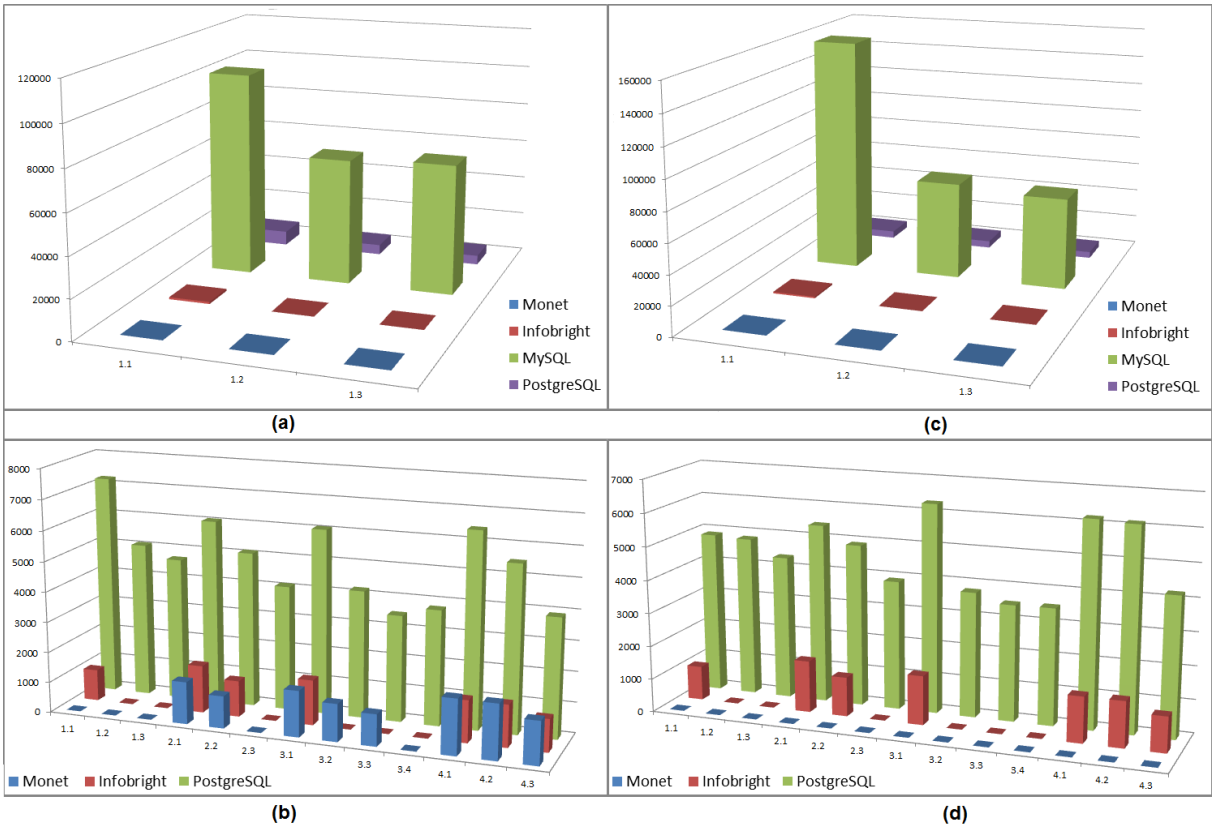

Figura 19: Gráficos representando (a) Tabela 10, (b) Tabela 11, (c) Tabela 12, (d) Tabela 13

5.1.3 Cenário 3 – SF=5

No Cenário 3, o número de registros referentes à cada tabela é o apresentado na Figura 13, utilizando SF=5. Dessa forma, as tabelas Part, Date, Lineorder, Customer e Supplier continham, respectivamente, 600.000, 2.556, 29.999.810, 150.000 e 10.000 registros.

A Tabela 15 apresenta o tempo consumido, em segundos, com a inserção dos dados deste cenário, para cada SGBD, utilizando as restrições de chave (com exceção do Infobright). O tempo consumido para o conjunto completo das consultas é apresentado na Tabela 16 (Total/CPU/Saída). As Tabelas 17 e 18 representam as mesmas características das Tabelas 15 e 16, sem a utilização das restrições de chaves, e a Figura 20 representa essas informações de forma gráfica.

Tabela 15: Tempo de inserção dos dados no Cenário 3, SF=5, com restrição de chaves

	Monet	**Infobright**	**PostgreSQL**
Supplier	1,717	1,311	0,156
Customer	2,527	1,420	2,841
Date	3,090	0,173	0,063
Part	5,585	3,432	16,162
Lineorder	1419,135	264,452	4082,839

O número de linhas retornadas para cada consulta, ordenadas de acordo com a Tabela 16, foi 1, 1, 1, 280, 56, 7, 150, 600, 24, 4, 35, 100, 746, respectivamente.

Tabela 16: Tempo consumido para cada consulta do Cenário 3, SF=5, (Total/CPU/Saída) com restrições de chaves, para o Teste 1

	Monet	Infobright	PostgreSQL
RF1	19491,679	7777,029	124,209
1.1	1,248/1,014/0,234	2,450/2,387/0,063	18,299/18,19/0,109
1.2	0,858/0,639/0,219	0,515/0,499/0,016	13,104/13,073/0,031
1.3	0,951/0,733/0,218	0,577/0,546/0,031	12,511/12,48/0,031
2.1	1,576/0,921/0,655	4,805/4,758/0,047	15,21/15,164/0,046
2.2	1,076/0,437/0,639	6,802/6,770/0,032	12,184/12,153/0,031
2.3	1,045/0,406/0,639	5,475/5,444/0,031	9,298/9,266/0,032
3.1	1,576/0,749/0,827	4,103/4,056/0,047	16,894/16,863/0,031
3.2	1,279/0,39/0,889	2,465/2,418/0,047	10,140/10,078/0,062
3.3	1,232/0,405/0,827	3,385/3,369/0,016	9,235/9,204/0,031
3.4	1,092/0,234/0,858	1,544/1,513/0,031	8,065/8,034/0,031
4.1	1,841/1,217/0,624	3,698/3,666/0,032	16,833/16,802/0,031
4.2	1,966/1,139/0,827	2,886/2,854/0,032	14,477/14,445/0,032
4.3	1,263/0,483/0,780	4,914/4,867/0,047	10,389/10,343/0,046
RF2	0,125	0,035	0,133

Tabela 17: Tempo de inserção dos dados no Cenário 3, SF=5, sem restrição de chaves

	Monet	Infobright	PostgreSQL
Supplier	1,747	1,311	0,063
Customer	2,527	1,420	1,670
Date	3,090	0,173	0,016
Part	5,585	3,432	9,080
Lineorder	353,155	264,452	421,919

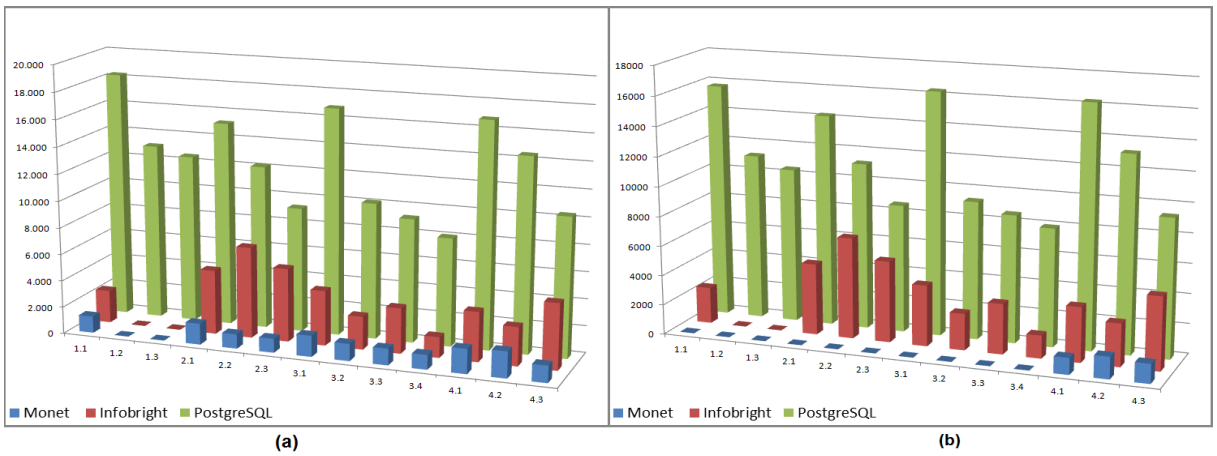

Figura 20: Gráficos representando (a) Tabela 16, (b) Tabela 18

Tabela 18: Tempo consumido para cada consulta do Cenário 3, SF=5, (Total/CPU/Saída) sem restrições de chaves, para o Teste 1

	Monet	Infobright	PostgreSQL
RF1	5854,439	7777,029	38,033
1.1	0,812/0,656/0,156	2,450/2,387/0,063	15,741/15,647/0,094
1.2	0,670/0,530/0,140	0,515/0,499/0,016	11,169/11,138/0,031
1.3	0,765/0,593/0,172	0,577/0,546/0,031	10,421/10,39/0,031
2.1	0,795/0,234/0,561	4,805/4,758/0,047	14,243/14,196/0,047
2.2	0,718/0,219/0,499	6,802/6,770/0,032	11,201/11,170/0,031
2.3	0,718/0,187/0,531	5,475/5,444/0,031	8,612/8,580/0,032
3.1	0,873/0,343/0,530	4,103/4,056/0,047	16,349/16,302/0,047
3.2	0,718/0,141/0,577	2,465/2,418/0,047	9,282/9,235/0,047
3.3	0,842/0,187/0,655	3,385/3,369/0,016	8,611/8,580/0,031
3.4	0,780/0,187/0,593	1,544/1,513/0,031	7,956/7,925/0,031
4.1	1,123/0,655/0,468	3,698/3,666/0,032	16,302/16,271/0,031
4.2	1,482/0,796/0,686	2,886/2,854/0,032	13,213/13,182/0,031
4.3	1,31/0,624/0,686	4,914/4,867/0,047	9,329/9,282/0,047
RF2	0,125	0,035	5,788

5.1.4 Cenário 4 – SF=10

No Cenário 4, o número de registros referentes à cada tabela é o apresentado na Figura 13, utilizando SF=10. Dessa forma, as tabelas Part, Date, Lineorder, Customer e Supplier continham, respectivamente, 800.000, 2.556, 59.986.052, 300.000 e 20.000 registros.

A Tabela 19 apresenta o tempo consumido, em segundos, com a inserção dos dados deste cenário, para cada SGBD, utilizando as restrições de chave (com exceção do Infobright). O tempo consumido para o conjunto completo das consultas é apresentado na Tabela 20 (Total/CPU/Saída). As Tabelas 21 e 22 representam as mesmas características das Tabelas, 19 e 20, sem a utilização das restrições de chaves, e a Figura 21 representa essas informações de forma gráfica. O número de linhas retornadas para cada consulta, ordenadas de acordo com a Tabela 23, foi 1, 1, 1, 280, 56, 7, 150, 600, 24, 4, 35, 100, 791, respectivamente.

Tabela 19: Tempo de inserção dos dados no Cenário 4, SF=10, com restrição de chaves

	Monet	Infobright	PostgreSQL
Supplier	1,873	1,344	0,312
Customer	3,854	1,888	6,225
Date	3,152	0,188	0,032
Part	8,784	4,056	17,535
Lineorder	2469,298	607,653	9520,416

Tabela 20: Tempo consumido para cada consulta do Cenário 4, SF=10, (Total/CPU/Saída) com restrições de chaves, para o Teste 1

	Monet	Infobright	PostgreSQL
RF1	13373,186	16995,326	347,335
1.1	3,401/3,261/0,14	4,166/4,103/0,063	23,673/23,642/0,031
1.2	3,089/3,027/0,062	1,108/1,077/0,031	21,076/21,044/0,032
1.3	3,229/3,151/0,078	1,139/1,123/0,016	20,556/20,524/0,032
2.1	2,964/2,792/0,172	9,906/9,844/0,062	31,512/31,465/0,047
2.2	2,793/2,606/0,187	17,706/17,675/0,031	27,939/27,908/0,031
2.3	2,652/2,48/0,172	14,165/14,149/0,016	16,895/16,864/0,031
3.1	3,604/3,37/0,234	8,33/8,284/0,046	35,615/35,568/0,047
3.2	2,948/2,714/0,234	6,724/6,677/0,047	19,172/19,125/0,047
3.3	2,808/2,605/0,203	8,377/8,346/0,031	17,191/17,16/0,031
3.4	2,714/2,512/0,202	3,323/3,292/0,031	16,645/16,614/0,031
4.1	5,18/5,008/0,172	7,067/7,035/0,032	36,102/36,07/0,032
4.2	4,68/4,477/0,203	10,28/10,249/0,031	26,811/26,778/0,033
4.3	4,165/3,9/0,265	10,624/10,577/0,047	18,754/18,707/0,047
RF2	0,250	0,047	0,666

Tabela 21: Tempo de inserção dos dados no Cenário 4, SF=10, sem restrição de chaves

	Monet	Infobright	PostgreSQL
Supplier	1,747	1,344	0,249
Customer	1,749	1,888	4,992
Date	3,526	0,1888	0,064
Part	3,168	4,056	15,882
Lineorder	441,208	607,653	860,593

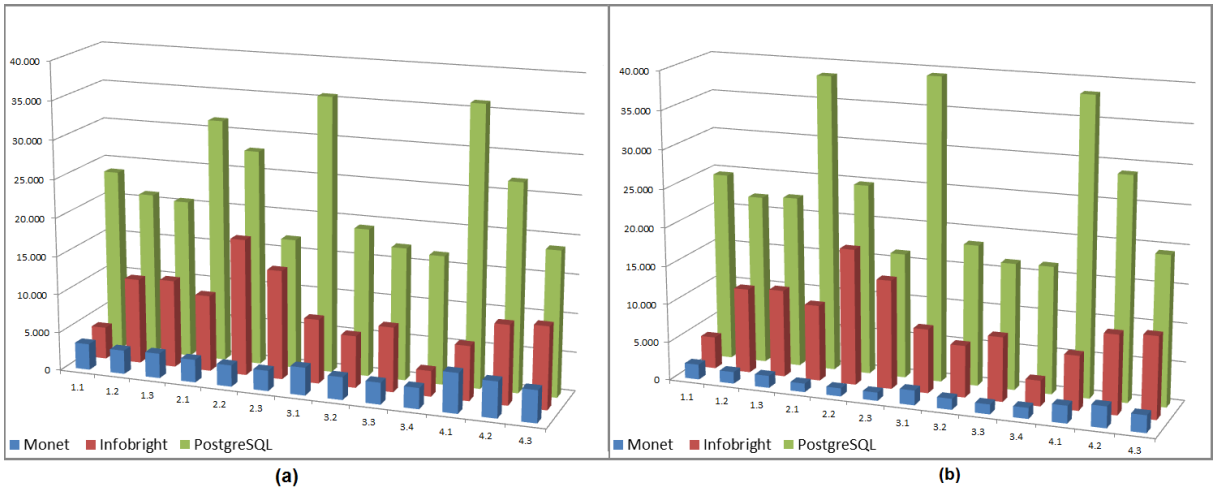

Figura 21: Gráficos representando (a) Tabela 20 e (b) Tabela 22

Tabela 22: Tempo consumido para cada consulta do Cenário 4, SF=10, (Total/CPU/Saída) sem restrições de chaves, para o Teste 1

	Monet	**Infobright**	**PostgreSQL**
RF1	17889,917	16995,326	80,754
1.1	1,966/1,623/0,343	4,166/4,103/0,063	24,586/24,398/0,188
1.2	1,529/1,248/0,281	1,108/1,077/0,031	22,043/22,012/0,031
1.3	1,576/1,326/0,25	1,139/1,123/0,016	22,339/22,324/0,015
2.1	1,123/0,437/0,686	9,906/9,844/0,062	38,485/38,423/0,062
2.2	1,061/0,312/0,749	17,706/17,675/0,031	24,82/24,789/0,031
2.3	1,076/0,249/0,827	14,165/14,149/0,016	16,302/16,286/0,016
3.1	1,919/0,796/1,123	8,330/8,284/0,046	39,39/39,359/0,031
3.2	1,42/0,25/1,17	6,724/6,677/0,047	18,346/18,315/0,031
3.3	1,279/0,265/1,014	8,377/8,346/0,031	16,442/16,411/0,031
3.4	1,419/0,265/1,154	3,323/3,292/0,031	16,537/16,505/0,032
4.1	2,278/1,42/0,858	7,067/7,035/0,032	38,391/38,376/0,015
4.2	2,777/1,763/1,014	10,28/10,249/0,031	28,923/28,907/0,016
4.3	2,246/1,373/0,873	10,624/10,577/0,047	19,406/19,375/0,031
RF2	0,078	0,047	11,934

5.2 Padrão de Teste 2

Os testes apresentados nessa seção foram executados conforme descrito o Teste 2 . Visto a problemática do MySQL na realização da primeira bateria de testes (Seção 5.1) optou-se por excluí-lo dessa segunda parte. O número de linhas retornadas para cada consulta se manteve o mesmo, em todos os casos, trabalhando-se sobre os mesmos dados inseridos no outro teste. As Tabelas 23, 25, 27 e 29 apresentam o tempo consumido para execução das consultas nos Cenários 1, 2, 3 e 4, respectivamente, com a utilização das restrições de chave e as Tabelas 24, 26, 28 e 30 apresentam o tempo consumido para execução das consultas nos Cenários 1, 2, 3 e 4, respectivamente, sem a utilização das restrições de chave.

Tabela 23: Tempo consumido para cada consulta do Cenário 1, SF=1, (Total/CPU/Saída) com restrições de chaves, para o Teste 2

	Monet	**Infobright**	**PostgreSQL**
RF1	519,720	1510,128	13,589
1.1	0,764/0,671/0,093	0,188/0,173/0,015	2,761/2,730/0,031
1.2	0,484/0,390/0,094	0,187/0,171/0,016	2,480/2,465/0,015
1.3	0,468/0,390/0,078	0,249/0,218/0,031	2,543/2,511/0,032
2.1	0,530/0,281/0,249	4,992/4,961/0,031	2,714/2,698/0,016
2.2	0,500/0,312/0,188	0,484/0,453/0,031	2,496/2,465/0,031
2.3	0,499/0,249/0,250	0,374/0,359/0,015	1,763/1,747/0,016
3.1	0,686/0,390/0,296	1,841/1,810/0,031	3,104/3,073/0,031
3.2	0,593/0,312/0,281	0,515/0,499/0,016	2,122/2,106/0,016
3.3	0,577/0,296/0,281	0,421/0,405/0,016	1,919/1,887/0,032
3.4	0,593/0,265/0,328	0,421/0,405/0,016	1,762/1,747/0,015
4.1	0,749/0,515/0,234	2,340/2,324/0,016	2,808/2,793/0,015
4.2	0,749/0,452/0,297	0,671/0,655/0,016	2,777/2,746/0,031
4.3	0,702/0,405/0,297	0,639/0,608/0,031	1,887/1,856/0,031
RF2	0,063	0,175	0,063

Tabela 24: Tempo consumido para cada consulta do Cenário 1, SF=1, (Total/CPU/Saída) sem restrições de chaves, para o Teste 2

	Monet	**Infobright**	**PostgreSQL**
RF1	335,415	1510,128	8,270
1.1	0,359/0,249/0,110	0,188/0,173/0,015	2,683/2,652/0,031
1.2	0,327/0,218/0,109	0,187/0,171/0,016	2,699/2,684/0,015
1.3	0,359/0,250/0,109	0,249/0,218/0,031	2,481/2,465/0,016
2.1	0,484/0,140/0,344	4,992/4,961/0,031	2,511/2,496/0,015
2.2	0,499/0,124/0,375	0,484/0,453/0,031	2,075/2,059/0,016
2.3	0,437/0,109/0,328	0,374/0,359/0,015	1,747/1,732/0,015
3.1	0,592/0,171/0,421	1,841/1,810/0,031	2,761/2,746/0,015
3.2	0,671/0,156/0,515	0,515/0,499/0,016	1,966/1,950/0,016
3.3	0,609/0,125/0,484	0,421/0,405/0,016	1,794/1,763/0,031
3.4	0,608/0,140/0,468	0,421/0,405/0,016	1,591/1,576/0,015
4.1	0,562/0,218/0,344	2,340/2,324/0,016	2,902/2,886/0,016
4.2	0,686/0,249/0,437	0,671/0,655/0,016	2,636/2,621/0,015
4.3	0,640/0,172/0,468	0,639/0,608/0,031	1,872/1,841/0,031
RF2	0,109	0,175	1,279

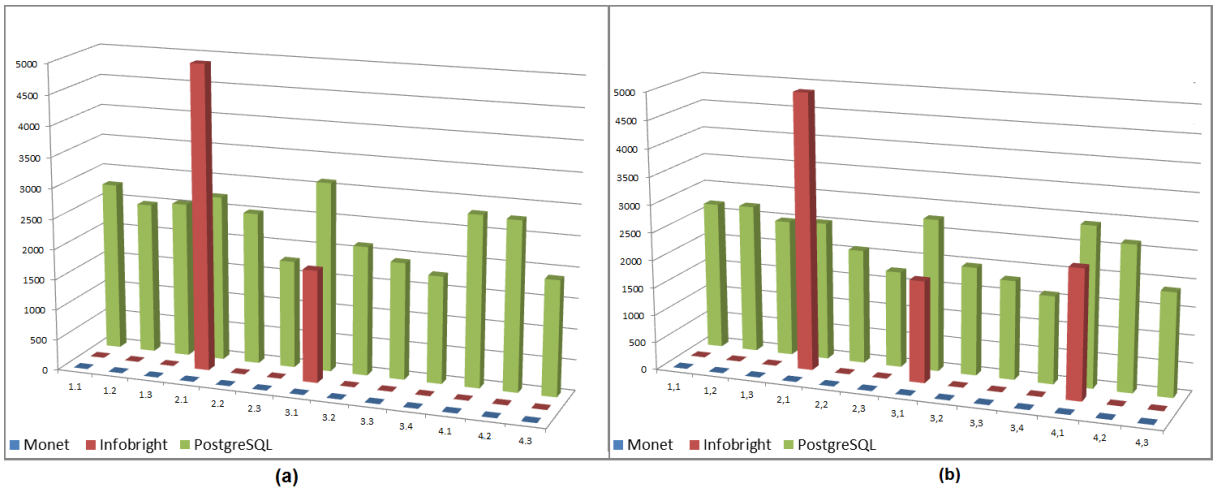

Figura 22: Gráfico representando (a) Tabela 23 e (b) Tabela 24

Tabela 25: Tempo consumido para cada consulta do Cenário 2, SF=2, (Total/CPU/Saída) com restrições de chaves, para o Teste 2

	Monet	**Infobright**	**PostgreSQL**
RF1	1093,232	3893,414	46,102
1.1	0,765/0,671/0,094	0,359/0,328/0,031	5,444/5,429/0,015
1.2	0,733/0,671/0,062	0,250/0,234/0,016	4,899/4,883/0,016
1.3	0,718/0,640/0,078	0,296/0,280/0,016	4,742/4,726/0,016
2.1	0,780/0,608/0,172	0,874/0,842/0,032	9,594/9,563/0,031
2.2	0,624/0,514/0,110	0,811/0,795/0,016	4,758/4,727/0,031
2.3	0,639/0,483/0,156	0,983/0,951/0,032	3,417/3,401/0,016
3.1	0,889/0,687/0,202	1,466/1,435/0,031	6,177/6,146/0,031
3.2	0,765/0,546/0,219	0,827/0,796/0,031	3,978/3,947/0,031
3.3	0,780/0,577/0,203	0,686/0,655/0,031	3,588/3,557/0,031
3.4	0,717/0,530/0,187	0,796/0,765/0,031	3,370/3,354/0,016
4.1	1,186/1,014/0,172	1,466/1,451/0,015	6,115/6,099/0,016
4.2	1,108/0,920/0,188	1,279/1,264/0,015	5,413/5,398/0,015
4.3	1,076/0,826/0,250	1,061/1,046/0,015	4,009/3,994/0,015
RF2	0,265	0,031	0,063

Tabela 26: Tempo consumido para cada consulta do Cenário 2, SF=2, (Total/CPU/Saída) sem restrições de chaves, para o Teste 2

	Monet	Infobright	PostgreSQL
RF1	655,662	3893,414	15,953
1.1	0,591/0,528/0,063	0,359/0,328/0,031	5,273/5,257/0,016
1.2	0,561/0,468/0,093	0,250/0,234/0,016	4,898/4,882/0,016
1.3	0,577/0,484/0,093	0,296/0,280/0,016	4,758/4,742/0,016
2.1	0,598/0,410/0,188	0,874/0,842/0,032	5,725/5,710/0,015
2.2	0,374/0,187/0,187	0,811/0,795/0,016	4,571/4,555/0,016
2.3	0,374/0,172/0,202	0,983/0,951/0,032	3,463/3,448/0,015
3.1	2,169/1,919/0,250	1,466/1,435/0,031	5,679/5,663/0,016
3.2	0,608/0,296/0,312	0,827/0,796/0,031	3,510/3,494/0,016
3.3	0,468/0,172/0,296	0,686/0,655/0,031	3,447/3,432/0,015
3.4	0,422/0,188/0,234	0,796/0,765/0,031	2,980/2,964/0,016
4.1	2,200/1,997/0,203	1,466/1,451/0,015	5,663/5,647/0,016
4.2	0,749/0,421/0,328	1,279/1,264/0,015	5,428/ 5,413/ 0,015
4.3	0,639/0,358/0,281	1,061/1,046/0,015	4,041/ 4,025/ 0,016
RF2	0,187	0,031	2,403

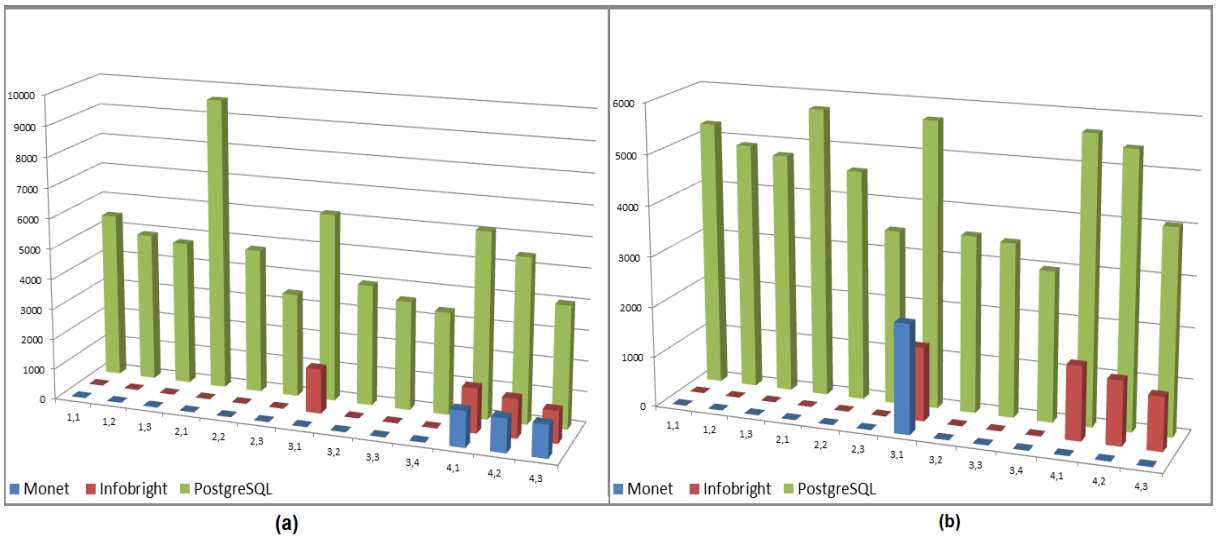

Figura 23: Gráfico representando (a) Tabela 25 e (b) Tabela 26

Tabela 27: Tempo consumido para cada consulta do Cenário 3, SF=5, (Total/CPU/Saída) com restrições de chaves, para o Teste 2

	Monet	**Infobright**	**PostgreSQL**
RF1	2801,802	8513,640	63,033
1.1	0,936/6,802/0,156	0,718/0,704/0,140	18,533/18,486/0,047
1.2	0,780/0,718/0,062	0,592/0,546/0,046	17,066/17,035/0,031
1.3	0,749/0,655/0,094	0,640/0,624/0,016	19,297/19,266/0,031
2.1	0,499/0,327/0,172	2,238/2,191/0,047	21,107/21,076/0,031
2.2	0,343/0,203/0,140	2,871/2,839/0,032	19,578/19,547/0,031
2.3	0,234/0,109/0,125	3,120/3,089/0,031	14,384/14,352/0,032
3.1	0,671/0,453/0,218	2,334/2,287/0,047	21,309/21,278/0,031
3.2	0,343/0,172/0,171	2,434/2,387/0,047	14,508/14,477/0,031
3.3	0,281/0,140/0,141	2,122/2,091/0,031	13,525/13,494/0,031
3.4	0,390/0,140/0,250	1,887/1,856/0,031	14,259/14,228/0,031
4.1	0,967/0,826/0,141	3,267/3,235/0,032	21,481/21,45/0,031
4.2	0,780/0,530/0,250	3,541/3,510/0,031	18,595/18,564/0,031
4.3	0,499/0,296/0,203	2,840/2,777/0,063	15,101/15,070/0,031
RF2	0,093	0,016	0,359

Tabela 28: Tempo consumido para cada consulta do Cenário 3, SF=5, (Total/CPU/Saída) sem restrições de chaves, para o Teste 2

	Monet	**Infobright**	**PostgreSQL**
RF1	1603,913	8513,640	36,612
1.1	0,842/0,764/0,078	0,718/0,704/0,140	13,494/13,463/0,031
1.2	0,811/0,733/0,078	0,592/0,546/0,046	12,309/12,278/0,031
1.3	0,718/0,656/0,062	0,640/0,624/0,016	12,246/12,23/0,016
2.1	0,530/0,281/0,249	2,238/2,191/0,047	16,052/16,021/0,031
2.2	0,422/0,234/0,188	2,871/2,839/0,032	12,168/12,137/0,031
2.3	0,390/0,171/0,219	3,120/3,089/0,031	8,674/8,658/0,016
3.1	0,624/0,374/0,250	2,334/2,287/0,047	15,865/15,834/0,031
3.2	0,499/0,187/0,312	2,434/2,387/0,047	10,125/10,093/0,032
3.3	0,452/0,218/0,234	2,122/2,091/0,031	9,282/9,250/0,032
3.4	0,437/0,172/0,265	1,887/1,856/0,031	8,720/8,704/0,016
4.1	0,952/0,749/0,203	3,267/3,235/0,032	17,253/17,222/0,031
4.2	1,123/0,826/0,297	3,541/3,510/0,031	14,789/14,774/0,015
4.3	1,139/0,733/0,406	2,84/2,777/0,063	10,234/10,203/0,031
RF2	0,203	0,016	6,006

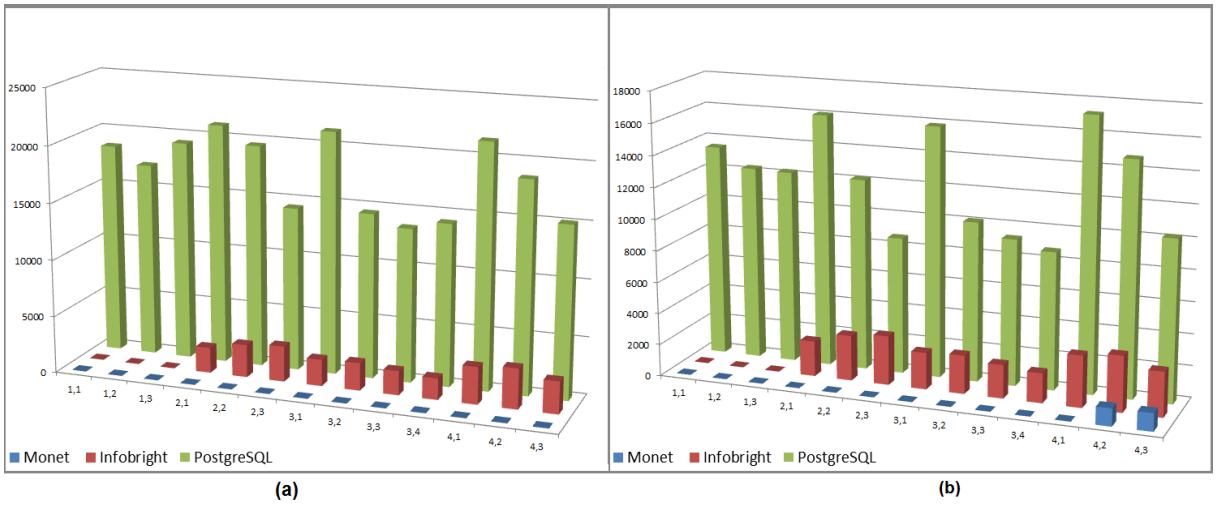

Figura 24: Gráfico representando (a) Tabela 27 e (b) Tabela 28

Tabela 29: Tempo consumido para cada consulta do Cenário 4, SF=10, (Total/CPU/Saída) com restrições de chaves, para o Teste 2

	Monet	**Infobright**	**PostgreSQL**
RF1	11185,560	39858,070	508,419
1.1	3,526/3,448/0,078	1,357/1,342/0,015	41,528/35,740/5,788
1.2	3,463/3,385/0,078	0,874/0,858/0,016	24,836/24,820/0,016
1.3	3,494/3,432/0,062	1,108/1,092/0,016	24,523/24,492/0,031
2.1	3,027/2,840/0,187	4,992/4,976/0,016	54,319/54,272/0,047
2.2	2,851/2,692/0,159	12,558/12,542/0,016	30,171/30,139/0,032
2.3	2,796/2,642/0,154	12,261/12,246/0,015	18,969/18,954/0,015
3.1	3,864/3,661/0,203	7,051/7,036/0,015	42,011/41,964/0,047
3.2	3,026/2,792/0,234	6,131/6,116/0,015	22,433/22,402/0,031
3.3	3,136/2,886/0,250	4,041/4,025/0,016	18,736/18,720/0,016
3.4	3,073/2,855/0,218	2,886/2,870/0,016	18,766/18,735/0,031
4.1	3,037/3,013/0,140	5,850/5,819/0,031	40,701/40,670/0,031
4.2	4,899/4,712/0,187	8,127/8,112/0,015	33,540/33,493/0,047
4.3	4,322/4,088/0,234	9,485/9,470/0,015	23,072/23,026/0,046
RF2	0,093	0,328	0,796

Tabela 30: Tempo consumido para cada consulta do Cenário 4, SF=10, (Total/CPU/Saída) sem restrições de chaves, para o Teste 2

	Monet	Infobright	PostgreSQL
RF1	5659,621	39858,070	87,173
1.1	2,168/1,560/0,608	1,357/1,342/0,015	24,975/24,944/0,031
1.2	2,044/1,451/0,593	0,874/0,858/0,016	21,279/21,248/0,031
1.3	1,420/1,248/0,172	1,108/1,092/0,016	21,294/21,263/0,031
2.1	1,045/0,421/0,624	4,992/4,976/0,016	37,783/37,736/0,047
2.2	0,718/0,312/0,406	12,558/12,542/0,016	23,650/23,619/0,031
2.3	1,185/0,343/0,842	12,261/12,246/0,015	15,756/15,725/0,031
3.1	1,966/0,796/1,170	7,051/7,036/0,015	36,161/36,114/0,047
3.2	1,217/0,327/0,890	6,131/6,116/0,015	18,096/18,049/0,047
3.3	1,030/0,484/0,546	4,041/4,025/0,016	15,787/15,756/0,031
3.4	1,217/0,296/0,921	2,886/2,870/0,016	15,554/15,523/0,031
4.1	1,996/1,497/0,499	5,850/5,819/0,031	36,395/36,363/0,032
4.2	2,231/1,670/0,561	8,127/8,112/0,015	27,066/27,034/0,032
4.3	2,059/1,279/0,780	9,485/9,470/0,015	18,096/18,064/0,030
RF2	0,375	0,328	11,732

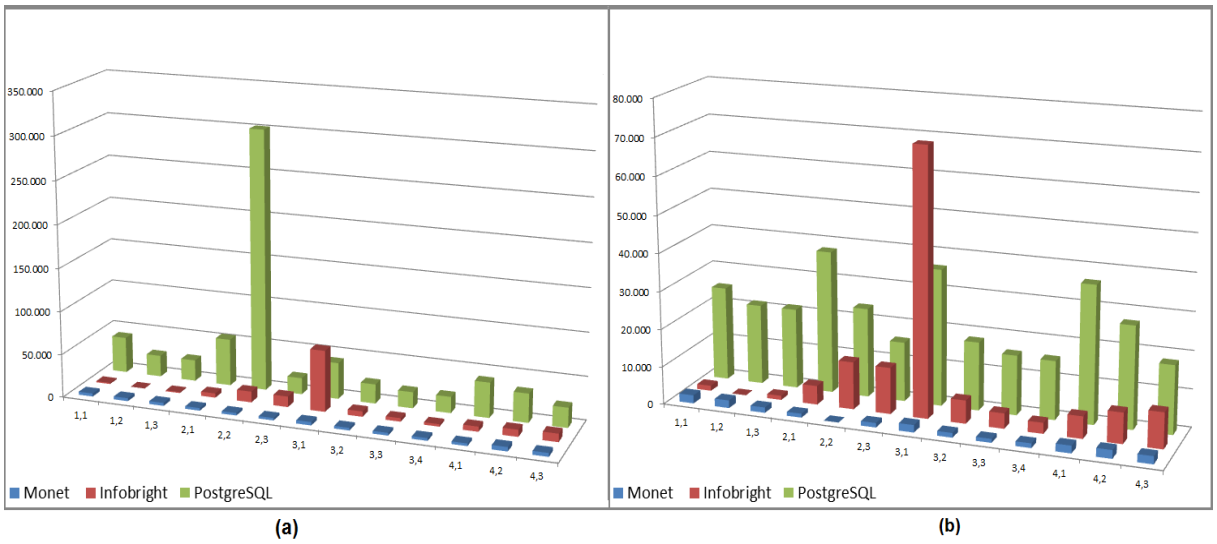

Figura 25: Gráfico representando (a) Tabela 29 e (b) Tabela 30

5.3 Análise

O principal objetivo dos testes aplicados foi verificar a capacidade de cada SGBD de manter-se "estável" com o aumento do número de informações, ou seja, manter seu tempo de resposta proporcional à escalabilidade. Para representar a estabilidade de cada SGBD,

verificou-se a percentagem, em relação ao primeiro cenário, para a realização das consultas e funções *refresh* nos três demais cenários. Dessa forma, é possível observar quanto tempo a mais em relação ao primeiro cenário cada SGBD levou para responder às consultas, comparando-os entre si. As Tabelas 31, 32, 33, 34, 35 e 36 e as Figuras 26, 27, 28, 29, 30 e 31 demonstram essas informações, para o primeiro caso de testes. As Tabelas 37, 38, 39, 40, 41 e 42 e as Figuras 32, 33, 34, 35, 36 e 37 demonstram essas informações, para o segundo caso de testes.

A Tabela 43 mostra o tamanho em disco necessário, em bytes, para armazenar os bancos de dados de cada cenário, para os três SGBD mencionados e a Tabela 44 a percentagem de crescimento em relação ao primeiro Cenário.

A partir dos testes realizados fica evidente que o SGBD que apresentou pior desempenho foi o MySQL, tanto pelo fato de executar apenas três das treze consultas em menos de 24 horas e ter consumido o maior tempo entre os demais para executá-las, quanto pela lentidão na inserção dos dados na tabela *lineorder*. Por tanto, a análise abaixo levará em conta apenas o PostgreSQL, MonetDB e Infobright.

Tabela 31: Percentagem do tempo consumido no segundo Cenário, SF=2, (utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 1)

	MonetDB	**PostgreSQL**	**Infobright**	**MySQL**
RF1	325,103%	213,477%	131,091%	623,223%
1.1	203,205%	281,744%	206,413%	8,624%
1.2	225,401%	198,202%	100,000%	62,394%
1.3	179,425%	187,500%	136,239%	282,974%
2.1	202,038%	215,563%	172,376%	
2.2	189,127%	238,699%	168,946%	
2.3	182,784%	230,635%	178,988%	
3.1	132,924%	201,628%	211,111%	
3.2	161,257%	216,021%	168,600%	
3.3	176,223%	196,569%	119,943%	
3.4	174,911%	200,794%	169,383%	
4.1	198,370%	214,591%	164,769%	
4.2	178,501%	202,321%	177,949%	
4.3	168,527%	194,536%	186,655%	
RF2	1362,500%	587,500%	48,387%	1432,931%

Tabela 32: Percentagem do tempo consumido no segundo Cenário, SF=2, (não utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 1)

	MonetDB	PostgreSQL	Infobright	MySQL
RF1	411,564%	1546,155%	131,091%	773,069%
1.1	168,400%	150,946%	206,413%	500,328%
1.2	118,400%	160,935%	100,000%	640,277%
1.3	137,200%	207,512%	136,239%	644,581%
2.1	123,032%	226,865%	172,376%	
2.2	138,790%	236,048%	168,946%	
2.3	119,872%	229,277%	178,988%	
3.1	138,765%	212,652%	211,111%	
3.2	138,997%	193,641%	168,600%	
3.3	133,232%	212,341%	119,943%	
3.4	158,108%	228,155%	169,383%	
4.1	166,400%	239,204%	164,769%	
4.2	136,538%	235,355%	177,949%	
4.3	140,855%	217,674%	186,655%	
RF2	165,957%	185,897%	48,387%	1408,418%

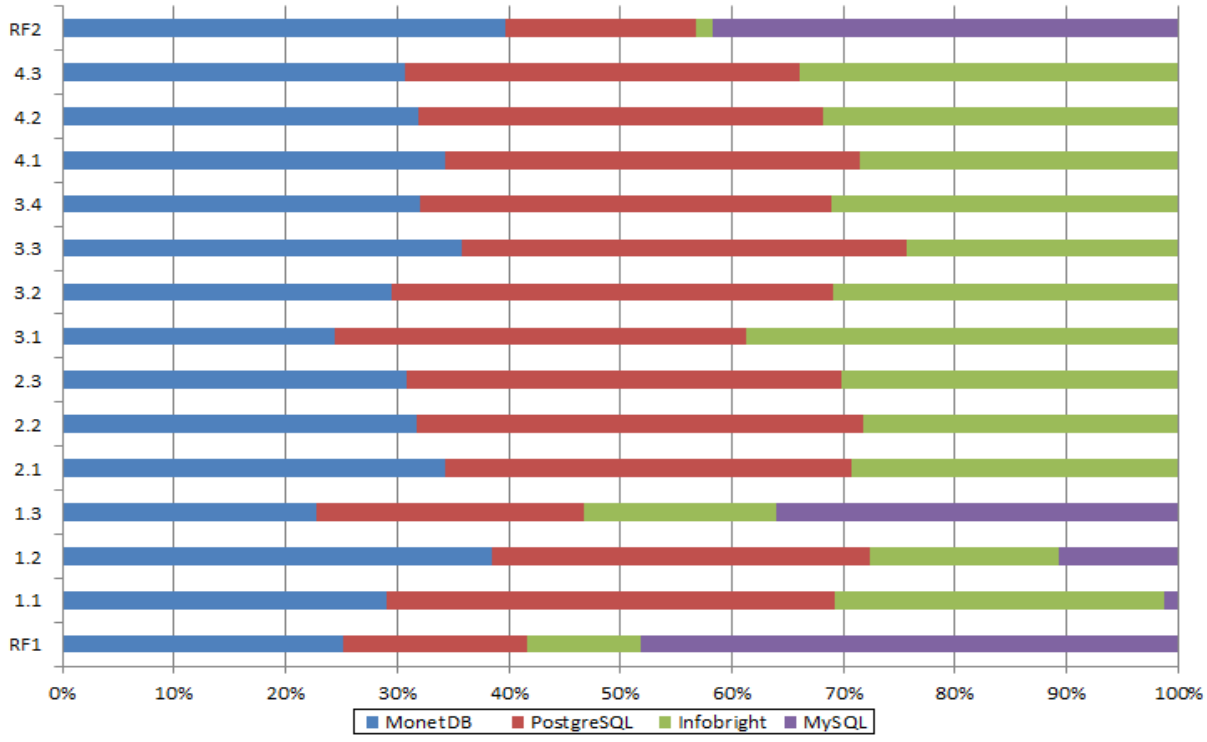

Figura 26: Gráfico referente à Tabela 31

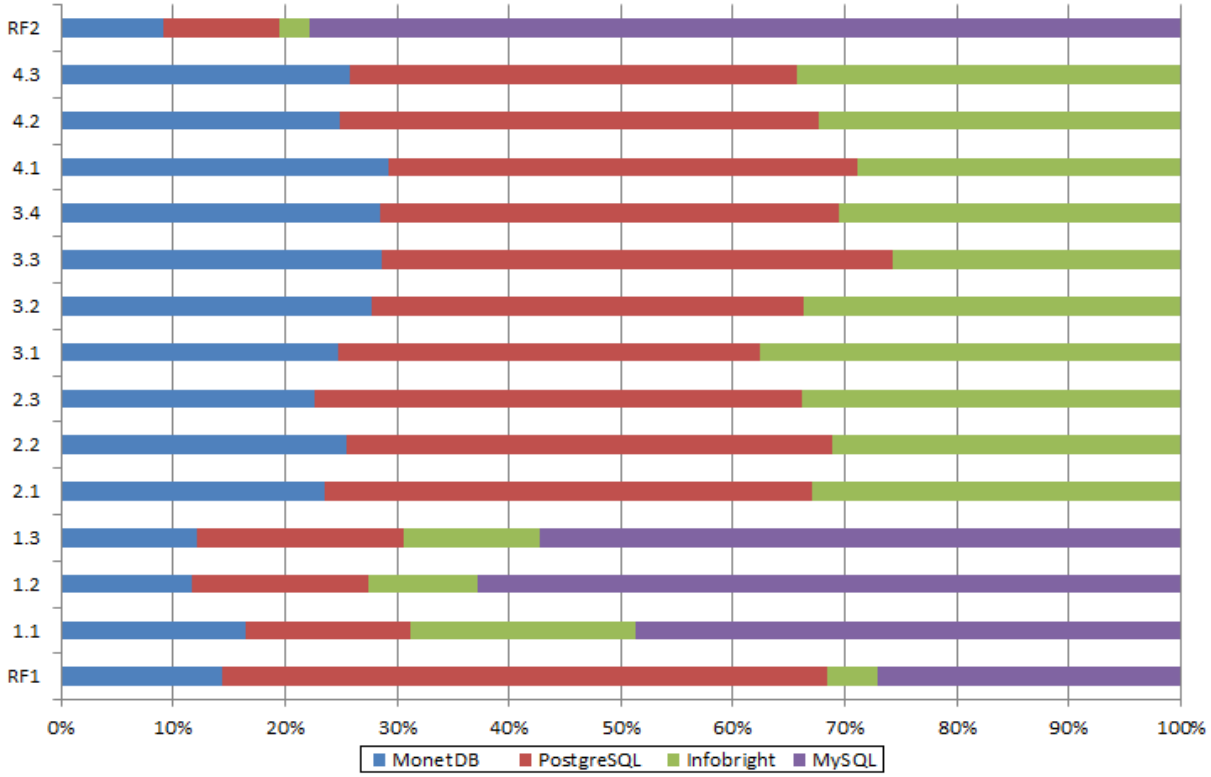

Figura 27: Gráfico referente à Tabela 32

Tabela 33: Percentagem do tempo consumido no terceiro Cenário, SF=5, (utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 1)

	MonetDB	**PostgreSQL**	**Infobright**
RF1	578,275%	951,647%	391,722%
1.1	266,667%	715,364%	490,982%
1.2	229,412%	512,275%	220,085%
1.3	210,398%	501,242%	264,679%
2.1	229,403%	541,667%	530,939%
2.2	191,800%	570,145%	968,946%
2.3	191,392%	522,653%	1065,175%
3.1	138,367%	561,262%	584,473%
3.2	167,408%	524,031%	493,000%
3.3	207,757%	519,404%	482,194%
3.4	194,306%	427,172%	381,235%
4.1	200,109%	562,037%	438,671%
4.2	193,886%	533,419%	370,000%
4.3	150,000%	516,095%	851,646%
RF2	781,250%	831,250%	112,903%

Tabela 34: Percentagem do tempo consumido no terceiro Cenário, SF=5, (não utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 1)

	MonetDB	PostgreSQL	Infobright
RF1	2005,529%	392,619%	391,722%
1.1	324,800%	488,093%	490,982%
1.2	268,000%	372,922%	220,085%
1.3	306,000%	498,612%	264,679%
2.1	231,778%	596,940%	530,939%
2.2	255,516%	539,807%	968,946%
2.3	230,128%	506,290%	1065,175%
3.1	215,556%	551,586%	584,473%
3.2	200,000%	476,000%	493,000%
3.3	256,707%	520,932%	482,194%
3.4	263,514%	514,951%	381,235%
4.1	299,467%	629,664%	438,671%
4.2	316,667%	507,217%	370,000%
4.3	311,164%	482,119%	851,646%
RF2	265,957%	436,501%	112,903%

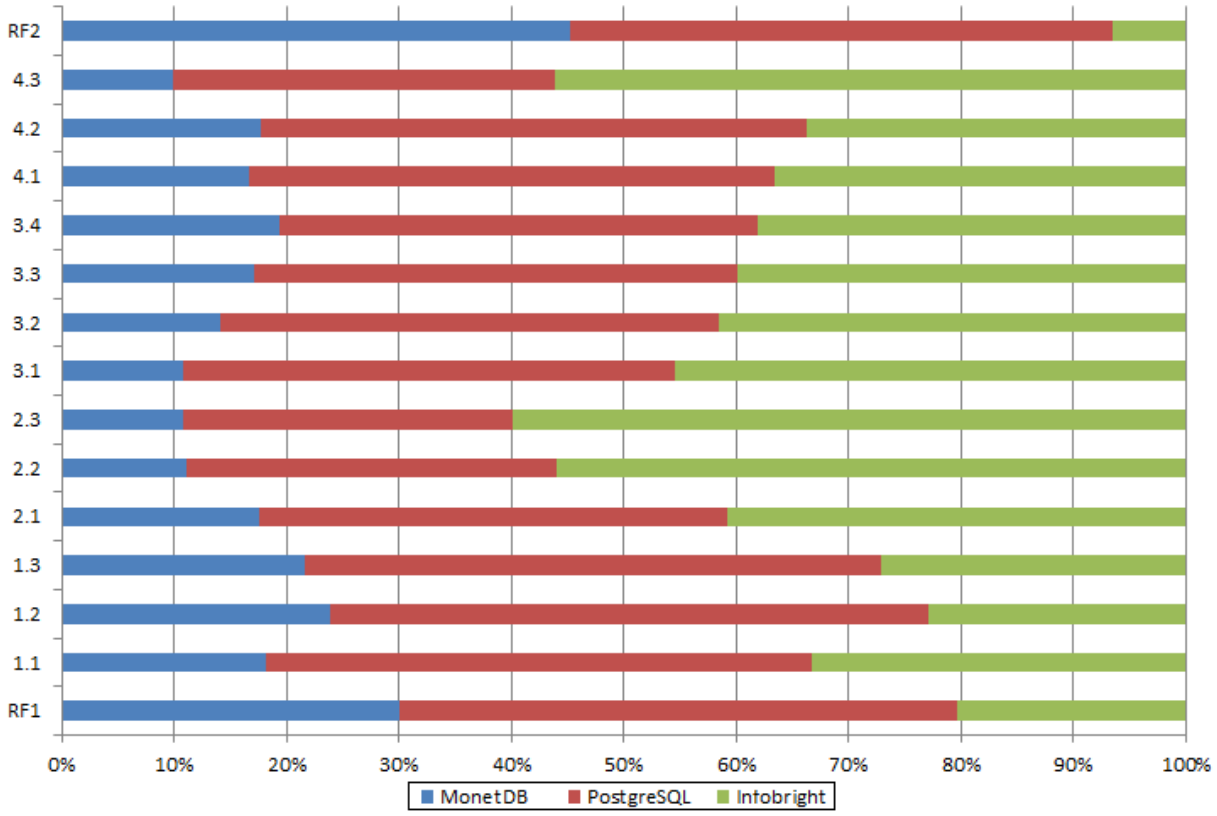

Figura 28: Gráfico referente à Tabela 33

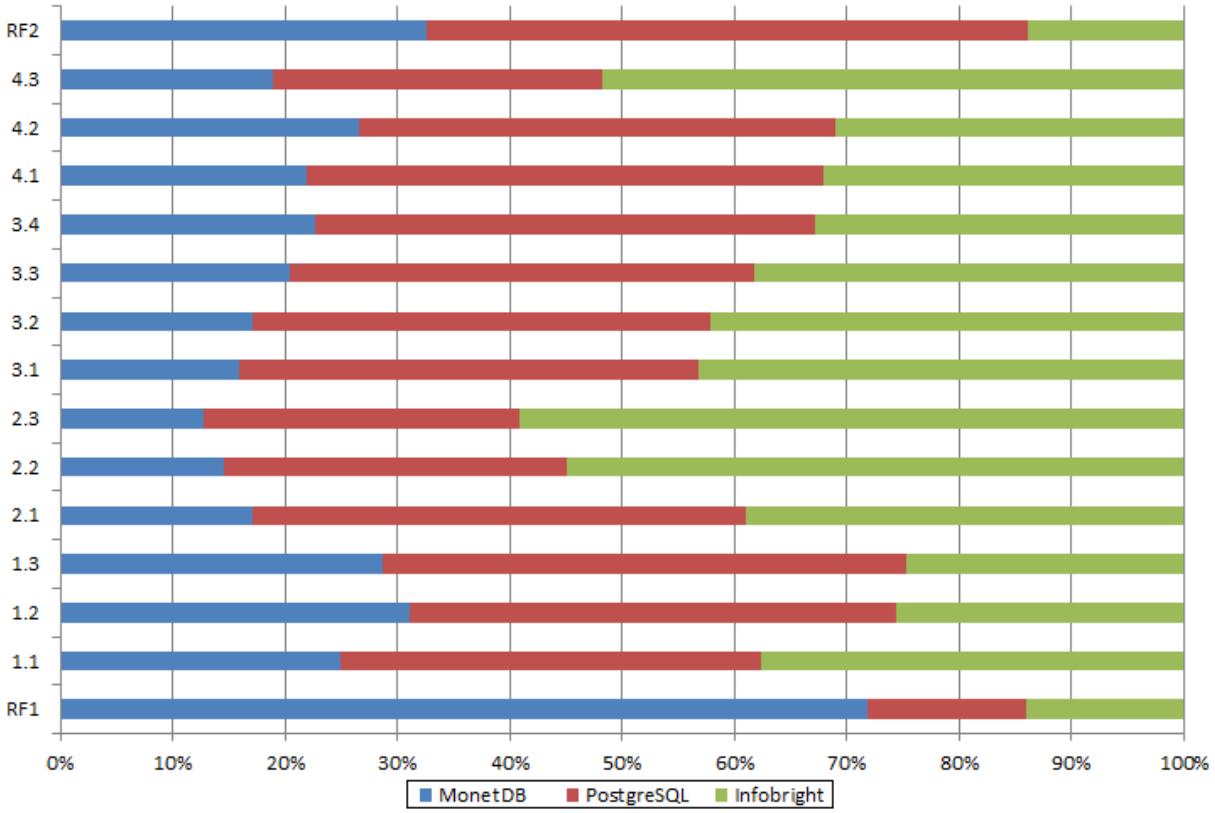

Figura 29: Gráfico referente à Tabela 34

Tabela 35: Percentagem do tempo consumido no quarto Cenário (utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 1)

	MonetDB	**PostgreSQL**	**Infobright**
RF1	396,753%	2661,163%	856,039%
1.1	726,709%	925,450%	834,870%
1.2	825,936%	823,925%	473,504%
1.3	714,381%	823,558%	522,477%
2.1	431,441%	1122,222%	1094,586%
2.2	497,861%	1307,394%	2522,222%
2.3	485,714%	949,691%	2755,837%
3.1	316,418%	1183,223%	1186,610%
3.2	385,864%	990,801%	1344,800%
3.3	473,524%	966,873%	1193,305%
3.4	482,918%	881,621%	820,494%
4.1	563,043%	1205,409%	838,316%
4.2	461,538%	987,878%	1317,949%
4.3	494,656%	931,644%	1841,248%
RF2	1562,500%	4162,500%	151,613%

Tabela 36: Percentagem do tempo consumido no quarto Cenário, SF=10, (não utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 1)

	MonetDB	PostgreSQL	Infobright
RF1	6128,468%	833,633%	856,039%
1.1	786,400%	762,357%	834,870%
1.2	611,600%	735,993%	473,504%
1.3	630,400%	1068,852%	522,477%
2.1	327,405%	1612,951%	1094,586%
2.2	377,580%	1196,145%	2522,222%
2.3	344,872%	958,377%	2755,837%
3.1	473,827%	1328,947%	1186,610%
3.2	395,543%	940,821%	1344,800%
3.3	389,939%	994,676%	1193,305%
3.4	479,392%	1070,356%	820,494%
4.1	607,467%	1482,851%	838,316%
4.2	593,376%	1110,288%	1317,949%
4.3	533,492%	1002,894%	1841,248%
RF2	165,957%	900,000%	151,613%

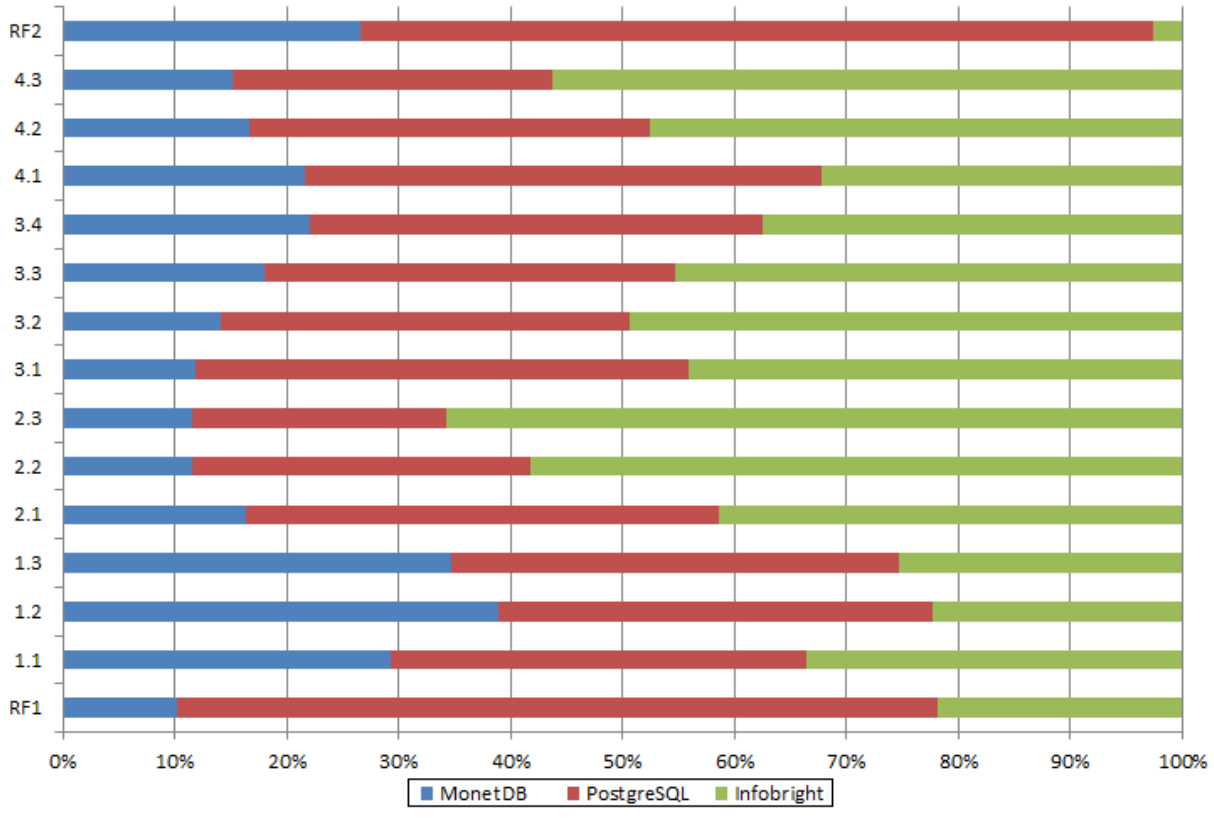

Figura 30: Gráfico referente à Tabela 35

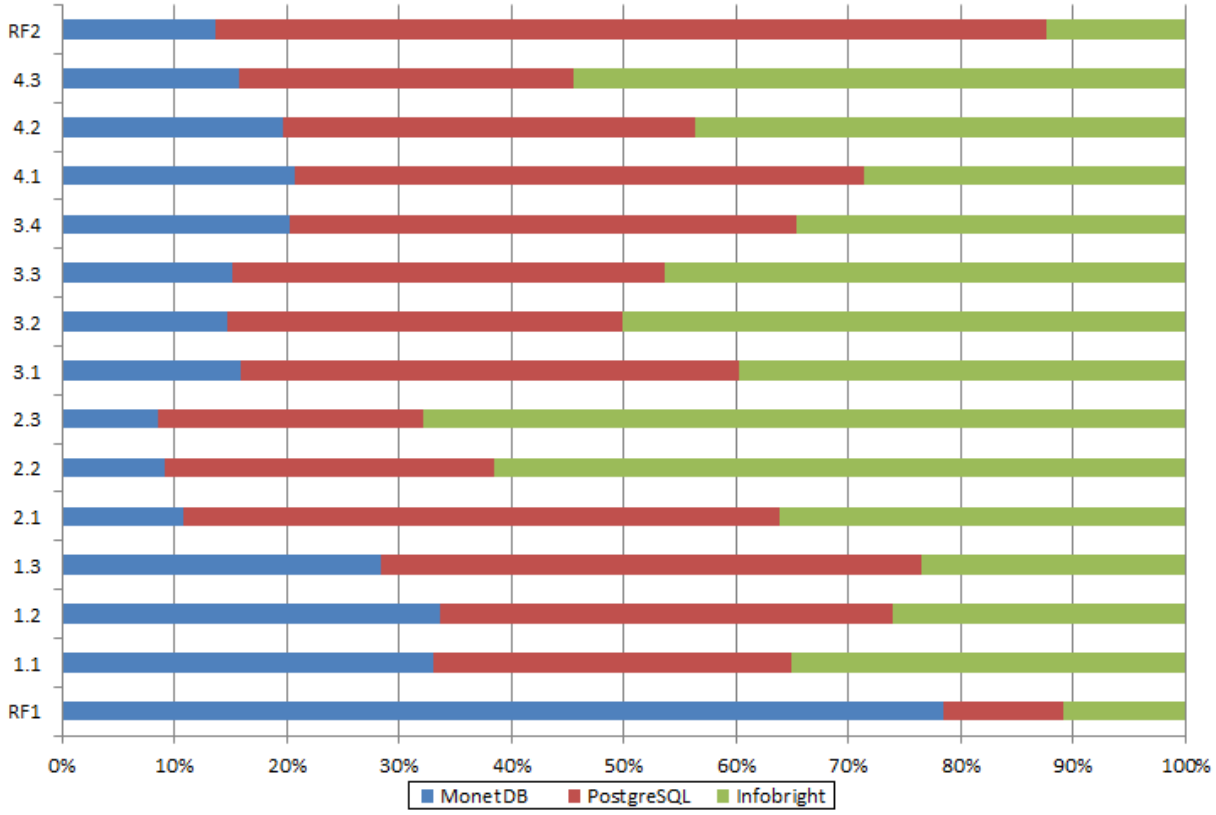

Figura 31: Gráfico referente à Tabela 36

Tabela 37: Percentagem do tempo consumido no segundo Cenário, SF=2, (utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 2)

	MonetDB	**PostgreSQL**	**Infobright**
RF1	210,543%	339,260%	257,820%
1.1	100,131%	197,175%	190,957%
1.2	151,446%	197,540%	133,690%
1.3	153,419%	186,473%	118,876%
2.1	147,170%	353,500%	17,508%
2.2	124,800%	190,625%	167,562%
2.3	128,056%	193,817%	262,834%
3.1	129,592%	199,001%	79,631%
3.2	129,005%	187,465%	160,583%
3.3	135,182%	186,972%	162,945%
3.4	120,911%	191,260%	189,074%
4.1	158,344%	217,771%	62,650%
4.2	147,931%	194,923%	190,611%
4.3	153,276%	212,454%	166,041%
RF2	420,635%	100,000%	17,714%

Tabela 38: Percentagem do tempo consumido no segundo Cenário, SF=2, (sem as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 2)

	MonetDB	PostgreSQL	Infobright
RF1	195,478%	192,902%	257,820%
1.1	164,624%	196,534%	190,957%
1.2	171,560%	181,475%	133,690%
1.3	160,724%	191,778%	118,876%
2.1	123,554%	227,997%	17,508%
2.2	74,950%	220,289%	167,562%
2.3	85,584%	198,226%	262,834%
3.1	366,385%	205,686%	79,631%
3.2	90,611%	178,535%	160,583%
3.3	76,847%	192,140%	162,945%
3.4	69,408%	187,304%	189,074%
4.1	391,459%	195,141%	62,650%
4.2	109,184%	205,918%	190,611%
4.3	99,844%	215,865%	166,041%
RF2	171,560%	187,881%	17,714%

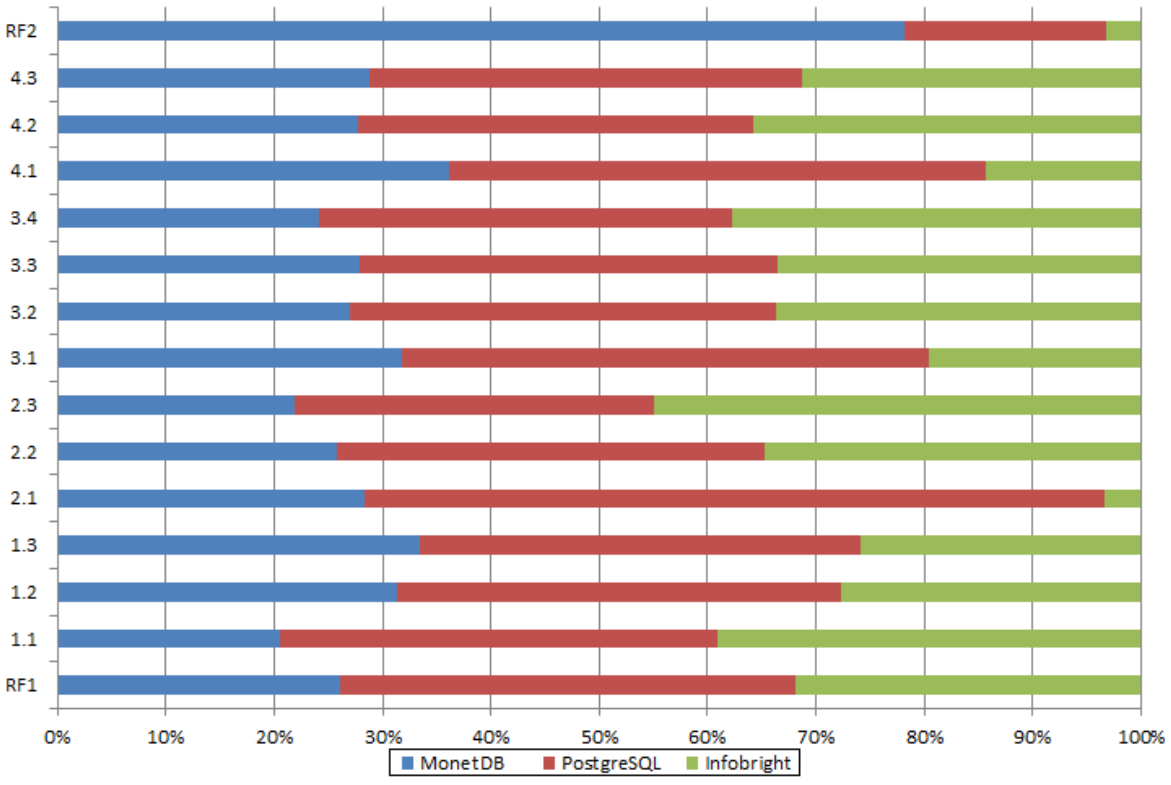

Figura 32: Gráfico referente à Tabela 37

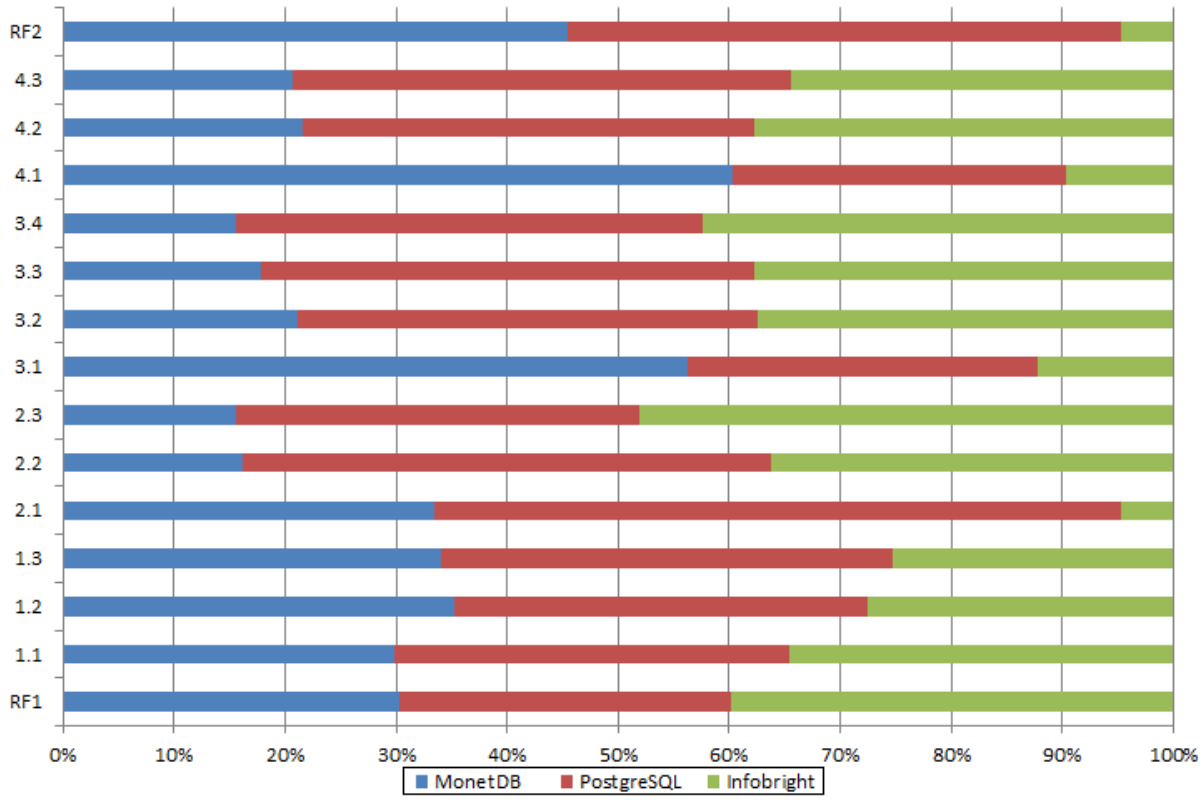

Figura 33: Gráfico referente à Tabela 38

Tabela 39: Percentagem do tempo consumido no terceiro Cenário, SF=5, (utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 2)

	MonetDB	**PostgreSQL**	**Infobright**
RF1	539,098%	463,853%	563,769%
1.1	122,513%	671,242%	381,915%
1.2	161,157%	688,145%	316,578%
1.3	160,043%	758,828%	257,028%
2.1	94,151%	777,708%	44,832%
2.2	68,600%	784,375%	593,182%
2.3	46,894%	815,882%	834,225%
3.1	97,813%	686,501%	126,779%
3.2	57,841%	683,695%	472,621%
3.3	48,700%	704,794%	504,038%
3.4	65,767%	809,251%	448,219%
4.1	129,105%	764,993%	139,615%
4.2	104,139%	669,607%	527,720%
4.3	71,083%	800,265%	444,444%
RF2	147,619%	569,841%	9,143%

Tabela 40: Percentagem do tempo consumido no terceiro Cenário, SF=5, (sem as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 2)

	MonetDB	PostgreSQL	Infobright
RF1	478,188%	442,709%	563,769%
1.1	234,540%	502,944%	381,915%
1.2	248,012%	456,058%	316,578%
1.3	200,000%	493,591%	257,028%
2.1	109,504%	639,267%	44,832%
2.2	84,569%	586,410%	593,182%
2.3	89,245%	496,508%	834,225%
3.1	105,405%	574,611%	126,779%
3.2	74,367%	515,005%	472,621%
3.3	74,220%	517,391%	504,038%
3.4	71,875%	548,083%	448,219%
4.1	169,395%	594,521%	139,615%
4.2	163,703%	561,039%	527,720%
4.3	177,969%	546,688%	444,444%
RF2	186,239%	469,586%	9,143%

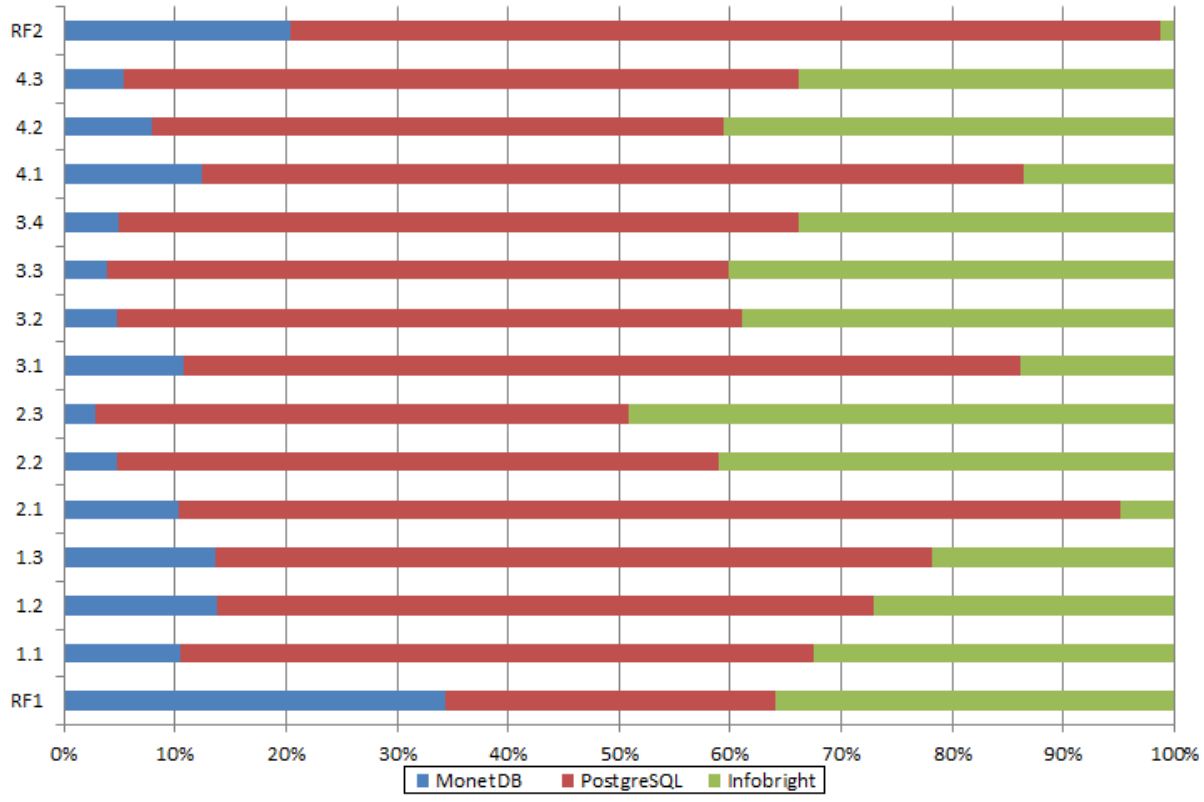

Figura 34: Gráfico referente à Tabela 39

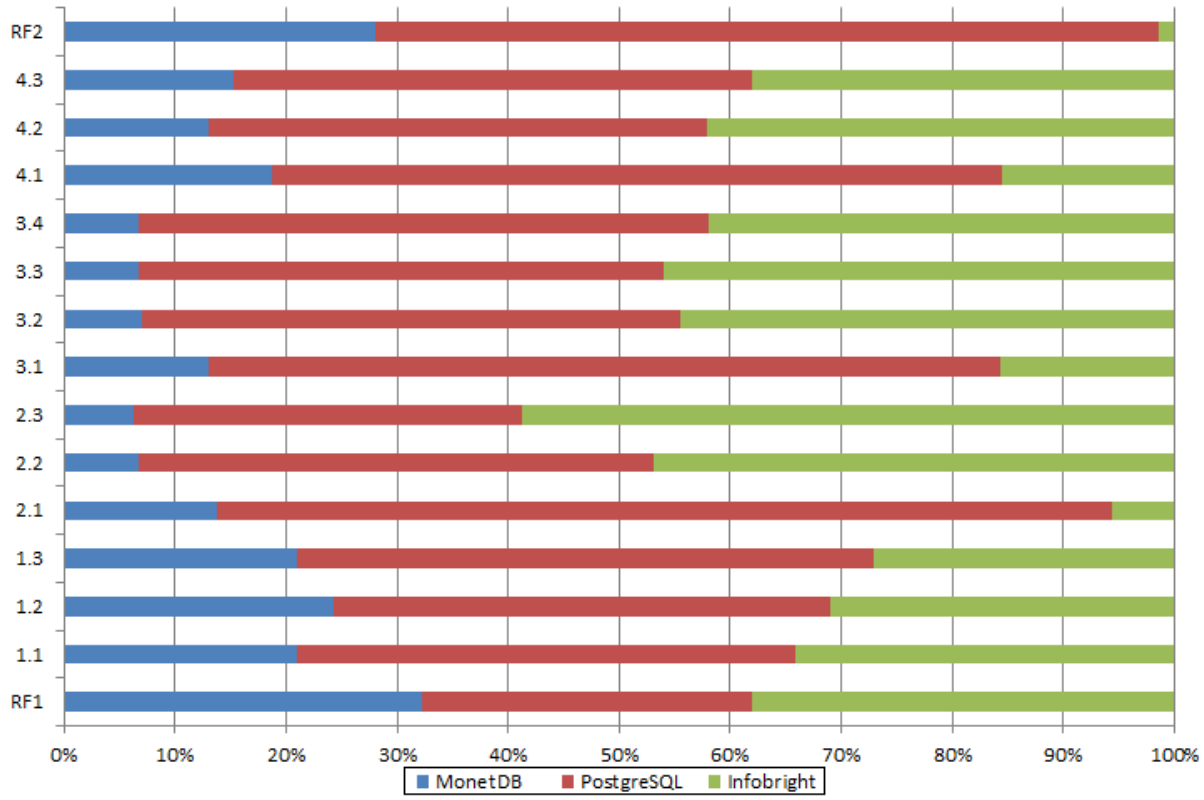

Figura 35: Gráfico referente à Tabela 40

Tabela 41: Percentagem do tempo consumido no quarto Cenário, SF=10, (utilizando as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 2)

	MonetDB	**PostgreSQL**	**Infobright**
RF1	2152,228%	3741,328%	2639,384%
1.1	461,518%	1504,093%	721,809%
1.2	715,496%	1001,452%	467,380%
1.3	746,581%	964,333%	444,980%
2.1	571,132%	2001,437%	100,000%
2.2	570,200%	1208,774%	2594,628%
2.3	560,321%	1075,950%	3278,342%
3.1	563,265%	1353,447%	382,998%
3.2	510,287%	1057,163%	1190,485%
3.3	543,501%	976,342%	959,857%
3.4	518,212%	1065,040%	685,511%
4.1	405,474%	1449,466%	250,000%
4.2	654,072%	1207,778%	1211,177%
4.3	615,670%	1222,682%	1484,351%
RF2	147,619%	1263,492%	187,429%

Tabela 42: Percentagem do tempo consumido no quarto Cenário, SF=10, (sem as restrições de chave), em relação ao primeiro, para realização das consultas (Teste 2)

	MonetDB	PostgreSQL	Infobright
RF1	1687,349%	1054,087%	2639,384%
1.1	603,900%	930,861%	721,809%
1.2	625,076%	788,403%	467,380%
1.3	395,543%	858,283%	444,980%
2.1	215,909%	1504,699%	100,000%
2.2	143,888%	1139,759%	2594,628%
2.3	271,167%	901,889%	3278,342%
3.1	332,095%	1309,707%	382,998%
3.2	181,371%	920,448%	1190,485%
3.3	169,130%	879,989%	959,857%
3.4	200,164%	977,624%	685,511%
4.1	355,160%	1254,135%	250,000%
4.2	325,219%	1026,783%	1211,177%
4.3	321,719%	966,667%	1484,351%
RF2	344,037%	917,279%	187,429%

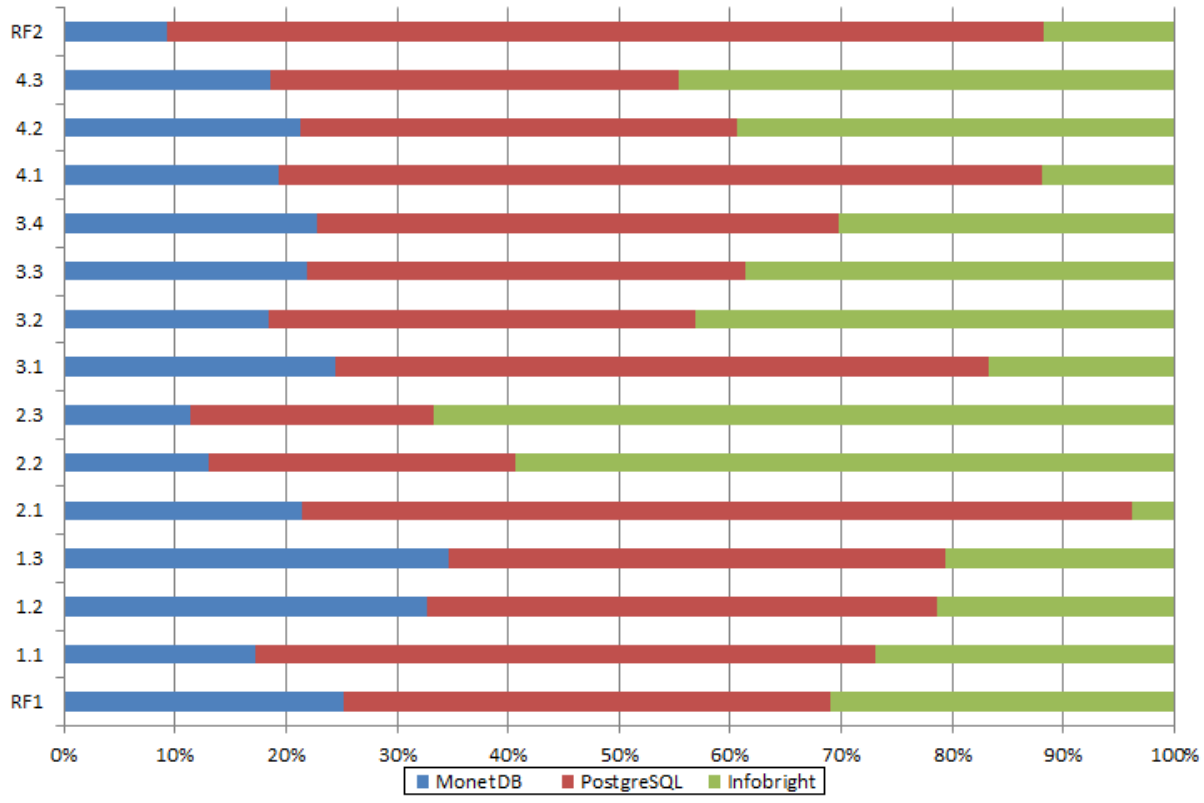

Figura 36: Gráfico referente à Tabela 41

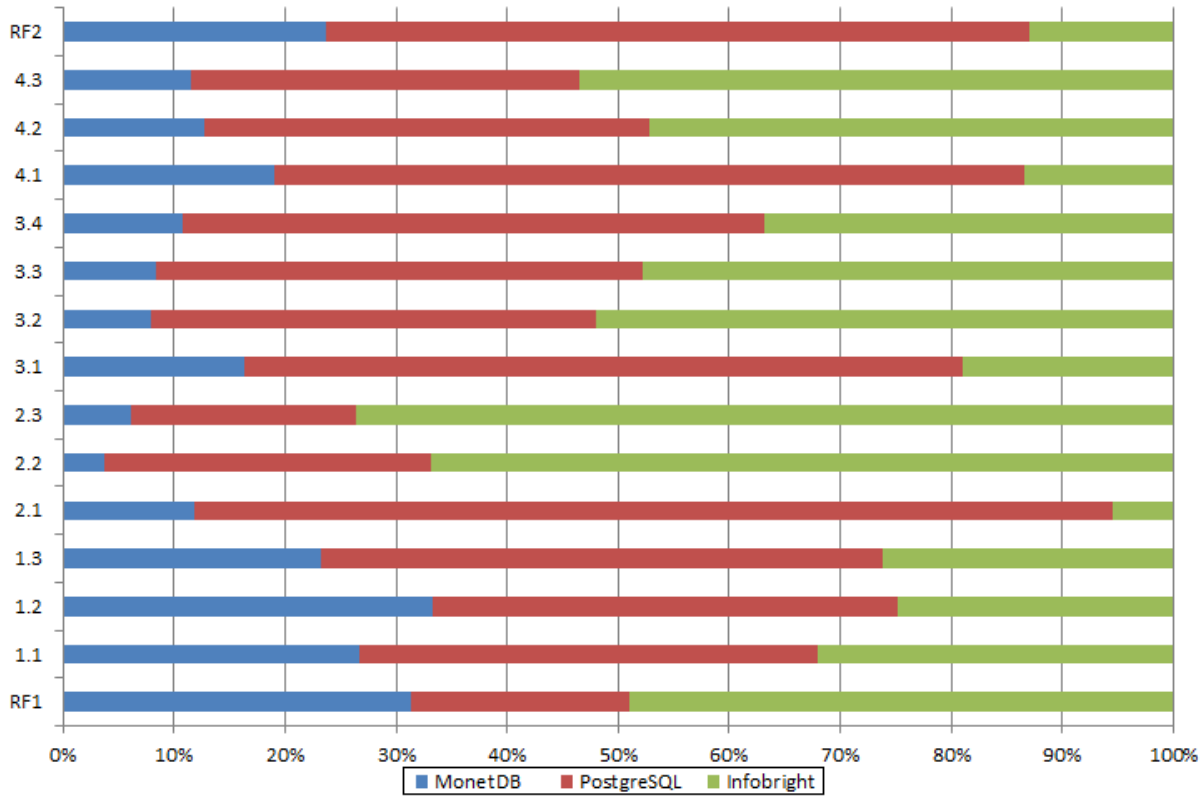

Figura 37: Gráfico referente à Tabela 42

Tabela 43: Tamanho em disco requerido pelos bancos de dados de cada SGBD (utilizando restrições de chaves/não utilizando restrições de chaves)

	MonetDB	PostgreSQL	Infobright
SF1	563.924.992/437.166.080	809.963.520/670.035.968	110.292.992
SF2	1.124.548.608/872.062.976	1.631.178.752/1.332.637.696	219.979.776
SF5	2.794.143.744/1.833.615.360	3.966.365.696/3.281.637.376	522.448.896
SF10	6.561.468.416/5.352.869.888	7.896.420.352/6.533.804.032	1.051.013.120

Tabela 44: Percentagem do espaço em disco requerido para armazenar os bancos de dados (utilizando restrições de chaves/não utilizando restrições de chaves)

	MonetDB	PostgreSQL	Infobright
SF2	199,415%/199,481%	201,389%/198,890%	199,450%
SF5	495,481%/419,432%	489,697%/489,770%	473,692%
SF10	1163,536%/1224,448%	974,911%/975,142%	952,928%

No primeiro caso de teste, em todos os cenários o PostgreSQL foi o que menos levou tempo para executar a função de *refresh* SF1. Por outro lado, foi o que mais demorou, no geral, para executar as consultas e a função de *refresh* SF2. Em todos os testes aplicados sem as restrições de chave o MonetDB se destacou, executando 11 das 13 consultas com o menor

tempo, em todos os cenários. Nos testes com as restrições de chaves, o Infobright se destacou para o primeiro e segundo cenários, executando 9 e 10 consultas, respectivamente, em menos tempo que os outros. Nos demais testes com restrições de chave, o MonetDB foi o que levou menos tempo respondendo às consultas, executando 11 delas em menor tempo que os demais, para ambos últimos cenários.

Quanto à estabilidade dos SGBD no primeiro caso de Teste, nota-se que no Cenário 2, com a utilização das restrições de chave para os demais SGBD, o Infobright foi o que menos reduziu o desempenho com a duplicação do número de informações. No entanto, no último cenário, este foi o que mais reduziu o desempenho. O MonetDB mostrou-se o SGBD mais estável entre os três, mantendo um bom desempenho em todos os demais cenários. O PostgreSQL, com exceção do último cenário, foi o que mais sofreu redução no desempenho com o aumento da escalabilidade, sendo o Cenário 2 o que mais o prejudicou.

No segundo caso de teste O PostgreSQL manteve o menor tempo na execução da função de *refresh* SF1 e o maior tempo na execução das consultas e da função de *refresh* SF2, em todos os casos. Nos testes executados sem as restrições de chave o MonetDB levou menos tempo, no geral, para executar as consultas referentes aos três últimos Cenários (SF2, SF5 e SF10), executando 8, 10 e 9 consultas, respectivamente, em menor tempo que os demais SGBD. No primeiro Cenário o Infobright executou 10 consultas em menor tempo. Para os testes sem utilização de restrição de chaves o MonetDB manteve o mesmo desempenho, executando 10 consultas de forma mais rápida, para os três últimos Cenários, sendo que o Infobright executou 10 consultas em menor tempo no primeiro Cenário.

Quanto à estabilidade dos SGBD no segundo caso de teste, o MonetDB se destacou em todos os Cenários, novamente se mostrando o SGBD mais estável. Em nenhum dos casos o PostgreSQL apresentou a menor variação no tempo de consulta, porém apresentou melhor desempenho que o Infobright em alguns casos, principalmente a partir do Cenário 3. No entanto, na maior parte dos casos o Infobright obteve melhor desempenho que o PostgreSQL.

Em relação ao tamanho em disco necessário para armazenar os bancos de dados o Infobright se destacou. No pior caso, ele necessitou de 3.5 vezes menos espaço em disco (Tabela 45).

O MonetDB também utilizou menos espaço em disco que o PostgreSQL, em todos os casos. Nota-se também que o Infobright, foi o que mais se manteve estável na utilização de espaço em disco com o aumento da escalabilidade.

Tabela 45: Proporção do tamanho dos bancos de dados em disco em relação ao Infobright (Utilizando chaves/Não utilizando)

	MonetDB	PostgreSQL
SF1	5.112/3.963	7.343/6.075
SF2	5.112/3.964	7.415/6.058
SF5	5.348/3.509	7.591/6.281
SF10	6.242/5.093	7.513/6.216

As Tabelas 46, 47, 48 e 49 apresentam a taxa de consultas por hora para cada teste realizado. Por estes resultados nota-se, no geral, que o MonetDB foi o SGBD com capacidade de executar o maior número de consultas por hora, sendo que seu desempenho cresceu consideravelmente em relação aos demais, aumentando-se a escalabilidade. O PostgreSQL em nenhum caso superou outro SGBD.

Tabela 46: Taxa de consultas por hora para o Teste 1, utilizando as restrições de chaves

	Monet	Infobright	PostgreSQL
SF1	4001.773	4682.682	1928.497
SF2	3714.325	6316.417	1695.404
SF5	8699.828	5277.577	1682.267
SF10	7397.003	4819.984	1646.156

Tabela 47: Taxa de consultas por hora para o Teste 1, sem as restrições de chaves

	Monet	Infobright	PostgreSQL
SF1	7963.957	4682.682	1518.355
SF2	10528.953	6316.417	1277.939
SF5	13141.631	5277.577	1528.861
SF10	14835.708	4819.984	1458.474

Tabela 48: Taxa de consultas por hora para o Teste 2, utilizando as restrições de chaves

	Monet	Infobright	PostgreSQL
SF1	4453.817	3859.183	1732.086
SF2	5906.311	6700.169	1712.925
SF5	21861.821	7452.633	1231.956
SF10	7888.250	5302.414	1322.973

Tabela 49: Taxa de consultas por hora para o Teste 2, sem as restrições de chaves

	Monet	Infobright	PostgreSQL
SF1	5068.785	3859.183	1527.674
SF2	7503.742	6700.169	1542.704
SF5	17949.691	7452.633	1449.597
SF10	15442.141	5302.414	1511.011

Capítulo 6

Conclusões

A busca por alternativas para persistência de dados cresce a cada dia, devido ao surgimento de necessidades que os bancos de dados relacionais nem sempre podem suprir, principalmente em questão de desempenho *versus* escalabilidade, disponibilidade, tolerância à partição e tipos de dados a serem manipulados. Os bancos de dados NoSQL surgiram para suprir tais necessidades, oferecendo diferentes maneiras de organização e manipulação de dados.

O modelo colunar da família NoSQL possui destaque, principalmente, em aplicações de DW sob o formato estrela, devido a seu bom desempenho na realização de consultas. Esse desempenho é obtido devido à forma de manipulação e organização dos dados por esse modelo, que favorece o tipo "padronizado" de consultas oferecidas por este modelo de DW.

Por meio da teoria ora pesquisada, notou-se que o modelo colunar possui vantagem em três pontos principais: (i) na compressão, por ter a capacidade de organizar informações semelhantes de forma contígua; (ii) na materialização, por não precisar processar *tuplas* desnecessárias; (iii) no bloco de iteração, por ser capaz de analisar todos os valores de uma coluna com um número menor de instruções de CPU.

Na avaliação experimental realizada com os quatro SGBD ficou evidente que, nas condições de organização de dados apresentadas, os SGBD de modelo colunar, em especial o MonetDB, responderam às consultas de forma mais rápida, conseguindo manter um bom desempenho à medida que o número de informações aumentou. As Tabelas 50 e 51 mostram quantas vezes o MonetDB e o Infobright foram mais rápidos que o PostgreSQL na execução das consultas para o Teste 1 e o Teste 2, respectivamente. Em contraposição, notou-se que o PostgreSQL foi capaz de inserir as informações, através da função *refresh* SF1, de forma muito mais rápida que os SGBD de modelo colunar, sendo essa diferença de 8 a 457 vezes.

As Tabelas 52 e 53 apresentam as informações de quantas consultas cada SGBD executou em menor tempo, em cada cenário, para o Teste 1 e Teste 2, respectivamente. Das 208

consultas executadas no total, o MonetDB obteve o menor tempo de execução em 133 e o Infobright em 75.

Tabela 50: Número de vezes que cada SGBD foi mais rápido que o PostgreSQL ao executar as consultas do Teste 1 (Utilizando chaves/Não utilizando)

	Monet	**Infobright**
SF1	3,437/6,849	4,046/3,919
SF2	4,130/10,199	5,071/4,785
SF5	9,801/13,134	3,820/3,495
SF10	7,053/15,045	3,031/3,168

Tabela 51: Número de vezes que cada SGBD foi mais rápido que o PostgreSQL ao executar as consultas do Teste 2 (Utilizando chaves/Não utilizando)

	Monet	**Infobright**
SF1	3,944/4,349	2,337/2,231
SF2	6,076/5,754	5,873/5,329
SF5	30,613/18,035	5,997/5,636
SF10	8,842/15,367	5,130/4,065

No armazenamento em disco, observou-se que o Infobright foi o SGBD que mais apresentou vantagens, tanto no espaço em disco necessário para armazenar os dados, quanto na estabilidade com o aumento do número de informações. Já o MonetDB, embora utilizando menos espaço em disco que o PostgreSQL em todos os casos, se mostrou o menos estável nos dois últimos Cenários.

Tabela 52: Número de consultas que cada SGBD executou em menor tempo para o Teste 1 (Utilizando chaves/Não utilizando)

	Monet	**Infobright**
SF1	4/11	9/2
SF2	3/11	10/2
SF5	10/11	3/2
SF10	11/11	2/2

Tabela 53: Número de consultas que cada SGBD executou em menor tempo para o Teste 2 (Utilizando chaves/Não utilizando)

	Monet	**Infobright**
SF1	3/3	10/10
SF2	8/8	5/5
SF5	10/10	3/3
SF10	9/10	4/3

Pelo levantamento bibliográfico e pela avaliação experimental conclui-se que, no tipo de estruturação de dados em estrela, os SGBD de modelo colunar são mais rápidos na realização de consultas e mais estáveis em manter esse tempo de resposta para diferentes situações de escalabilidade. Comparando o MonetDB, o Infobright e o PostgreSQL, confirma-se a teoria de que a inserção de dados no modelo colunar é mais lenta que no relacional. Pelos resultados obtidos nos testes, a forma de estruturação do modelo colunar deixou claro que os SGBD que seguem este modelo possuem maior aproveitamento na compressão de dados, realização de consultas e menor aproveitamento na inserção de dados, utilizando ou não restrição de chaves. Não se sabe dizer se o MySQL apresentou tais estatísticas precárias em relação aos demais SGBD por alguma incompatibilidade ou por não ter sido feito nenhum ajuste sobre estes SGBD.

Em trabalhos futuros espera-se:

- Incluir novos SGBD nos testes, tanto de modelo colunar quanto de modelo relacional, a exemplo do Oracle [Oracle, 2012a], Oracle Database 11*g* [Oracle, 2012c] e do SQL Server [Microsoft, 2012].
- Realizar duas novas baterias de testes, uma utilizando índice nos SGBD de modelo relacional, visando constatar o quanto de vantagem a utilização deste recurso proporciona aos SGBD que seguem este modelo e outra testando os SGBD com múltiplos acessos ao servidor (*throughput test*, do TPC-H).

Apêndice A

Detalhes do SSB

O SSB é um padrão de benchmark baseado no TPC-H (Figura 28), adaptado deste para seguir o modelo em estrela de estruturação de dados.

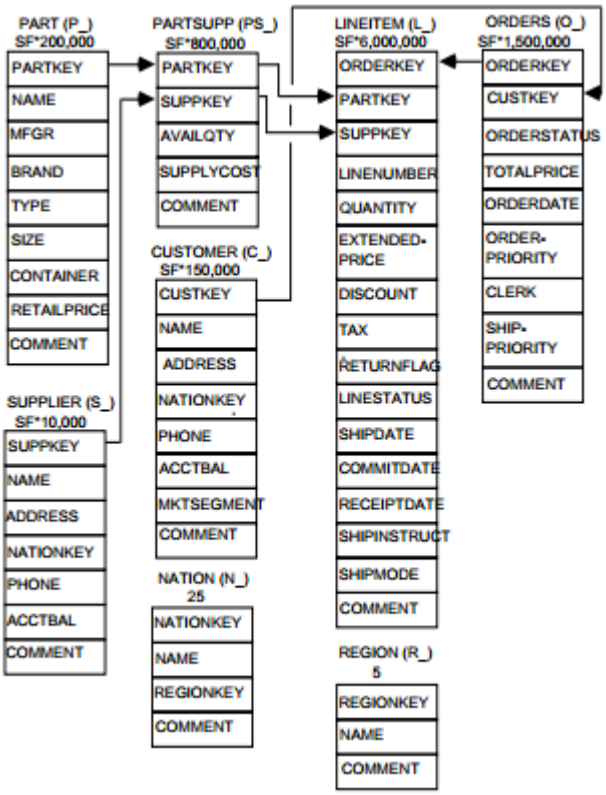

Figura 28: Modelo TPC-H

As principais alterações do SSB em relação ao TPC-H são:

- Exclusão da tabela PARTSUPP;
- Combinação das tabelas LINEITEM (L) e ORDERS (O), formando a tabela LINEORDER (LO), deletando os seguintes atributos: L_RETURNFLAG, L_LINESTATUS, L_SHIPDATE, L_RECEIPTDATE, L_COMMENT, L_SHIPINSTRUCT, L_CLERK, O_ORDERSTATUS, O_COMMENT, LO_CLERK e

adicionando os seguintes atributos à nova tabela: LO_SUPPLYCOST, LO_ORDSUPPLYCOST e LO_ORDTOTALPRICE;

- Mudança na cardinalidade da tabela PART (P) de SF*200.000 registros para 200.000*x(1+$\log_2$SF). Redução do atributo P_NAME de 55 para 22 bytes. Adição das colunas P_COLOR, P_CATEGORY. Mudança da coluna P_MFGR, de texto fixo de tamanho 25, para seis caracteres. Exclusão das colunas P_RETAILPRICE, P_COMMENT e substituição da coluna P_BRAND pela coluna P_BRAND1, que é uma divisão de P_CATEGORY.
- Redução de SF*10.000 para SF*2.000 na cardinalidade da tabela SUPPLIER (S). Adição da coluna S_CITY, utilizando os nove primeiros caracteres da coluna S_NATION.
- Redução de SF*150.000 para SF*30.000 na cardinalidade da tabela CUSTOMER (C) e exclusão da coluna C_ACCTBAL.
- Criação da tabela DATE (D).

O detalhamento do esquema SSB (Figura 13) é apresentado a seguir, junto com as consultas:

Q1.1

SELECT SUM(lo_extendedprice*lo_discount) **AS** revenue
FROM lineorder, date
WHERE lo_orderdate = d_datekey **AND** d_year = 1993 **AND** lo_discount **BETWEEN** 1 **AND** 3
AND lo_quantity < 25;

Q1.2

SELECT SUM(lo_extendedprice*lo_discount) **AS** revenue
FROM lineorder, date
WHERE lo_orderdate = d_datekey **AND** d_yearmonthnum = 199401 **AND** lo_discount **BETWEEN** 4 **AND** 6
AND lo_quantity **BETWEEN** 26 **AND** 35;

Q1.3

SELECT SUM(lo_extendedprice*lo_discount) **AS** revenue

FROM lineorder, date

WHERE lo_orderdate = d_datekey **AND** d_weeknuminyear = 6 **AND** d_year = 1994

AND lo_discount **BETWEEN** 5 **AND** 7 **AND** lo_quantity **BETWEEN** 26 **AND** 35;

Q2.1

SELECT SUM(lo_revenue), d_year, p_brand1

FROM lineorder, date, part, supplier

WHERE lo_orderdate = d_datekey **AND** lo_partkey = p_partkey **AND** lo_suppkey = s_suppkey

AND p_category = 'MFGR#12' **AND** s_region = 'AMERICA'

GROUP BY d_year, p_brand1 **ORDER BY** d_year, p_brand1;

Q2.2

SELECT SUM(lo_revenue), d_year, p_brand1

FROM lineorder, date, part, supplier

WHERE lo_orderdate = d_datekey **AND** lo_partkey = p_partkey **AND** lo_suppkey = s_suppkey

AND p_brand1 **BETWEEN** 'MFGR#2221' **AND** 'MFGR#2228' **AND** s_region = 'ASIA'

GROUP BY d_year, p_brand1 **ORDER BY** d_year, p_brand1;

Q2.3

SELECT SUM(lo_revenue), d_year, p_brand1

FROM lineorder, date, part, supplier

WHERE lo_orderdate = d_datekey **AND** lo_partkey = p_partkey **AND** lo_suppkey = s_suppkey

AND p_brand1 = 'MFGR#2221' **AND** s_region = 'EUROPE'

GROUP BY d_year, p_brand1 **ORDER BY** d_year, p_brand1;

Q3.1

SELECT c_nation, s_nation, d_year, **SUM**(lo_revenue) **AS** revenue

FROM customer, lineorder, supplier, date **WHERE** lo_custkey = c_custkey **AND** lo_suppkey = s_suppkey **AND** lo_orderdate = d_datekey

AND c_region = 'ASIA' **AND** s_region = 'ASIA' **AND** d_year >= 1992 **AND** d_year <= 1997

GROUP BY c_nation, s_nation, d_year **ORDER BY** d_year asc, revenue desc;

Q3.2

SELECT c_city, s_city, d_year, **SUM**(lo_revenue) **AS** revenue

FROM customer, lineorder, supplier, date

WHERE lo_custkey = c_custkey **AND** lo_suppkey = s_suppkey **AND** lo_orderdate = d_datekey

AND c_nation = 'UNITED STATES' **AND** s_nation = 'UNITED STATES' **AND** d_year >= 1992

AND d_year <= 1997

GROUP BY c_city, s_city, d_year **ORDER BY** d_year asc, revenue desc;

Q3.3

SELECT c_city, s_city, d_year, **SUM**(lo_revenue) **AS** revenue

FROM customer, lineorder, supplier, date

WHERE lo_custkey = c_custkey **AND** lo_suppkey = s_suppkey **AND** lo_orderdate = d_datekey

AND (c_city='UNITED KI1' **OR** c_city='UNITED KI5')

AND (s_city='UNITED KI1' **OR** s_city='UNITED KI5') **AND** d_year >= 1992 **AND** d_year <= 1997

GROUP BY c_city, s_city, d_year **ORDER BY** d_year asc, revenue desc;

Q3.4

SELECT c_city, s_city, d_year, **SUM**(lo_revenue) **AS** revenue

FROM customer, lineorder, supplier, date

WHERE lo_custkey = c_custkey and lo_suppkey = s_suppkey **AND** lo_orderdate = d_datekey
AND (c_city='UNITED KI1' **OR** c_city='UNITED KI5')
AND (s_city='UNITED KI1' **OR** s_city='UNITED KI5') **AND** d_yearmonth = 'Dec1997'
GROUP BY c_city, s_city, d_year **ORDER BY** d_year asc, revenue desc;

Q4.1

SELECT d_year, c_nation, **SUM**(lo_revenue - lo_supplycost) **AS** profit
FROM date, customer, supplier,
PART, lineorder **WHERE** lo_custkey = c_custkey **AND** lo_suppkey = s_suppkey
AND lo_partkey = p_partkey **AND** lo_orderdate = d_datekey **AND** c_region = 'AMERICA'
AND s_region = 'AMERICA' **AND** (p_mfgr = 'MFGR#1' **OR** p_mfgr = 'MFGR#2')
GROUP BY d_year, c_nation **ORDER BY** d_year, c_nation;

Q4.2

SELECT d_year, s_nation, p_category, **SUM**(lo_revenue - lo_supplycost) **AS** profit
FROM date, customer, supplier, part, lineorder
WHERE lo_custkey = c_custkey
AND lo_suppkey = s_suppkey **AND** lo_partkey = p_partkey **AND** lo_orderdate = d_datekey
AND c_region = 'AMERICA' **AND** s_region = 'AMERICA' **AND** (d_year = 1997 **OR** d_year = 1998)
AND (p_mfgr = 'MFGR#1' **OR** p_mfgr = 'MFGR#2')
GROUP BY d_year, s_nation, p_category **ORDER BY** d_year, s_nation, p_category;

Q4.3

SELECT d_year, s_city, p_brand1, **SUM**(lo_revenue - lo_supplycost) **AS** profit
FROM date, customer, supplier, part, lineorder
WHERE lo_custkey = c_custkey
AND lo_suppkey = s_suppkey **AND** lo_partkey = p_partkey **AND** lo_orderdate = d_datekey
AND c_region = 'AMERICA' **AND** s_nation = 'UNITED STATES' **AND** (d_year = 1997 **OR** d_year = 1998) **AND** p_category = 'MFGR#14'
GROUP BY d_year, s_city, p_brand1 **ORDER BY** d_year, s_city, p_brand1;

Tabela 54: Tabela LINEORDER (SF*6.000.000)

Nome do atributo	Tipo	Restrição/Tamanho
LO_ORDERKEY	Integer	Chave Primária
LO_LINENUMBER	Integer	Chave Primária
LO_CUSTKEY	Integer	Chave Estrangeira para C_CUSTKEY
LO_PARTKEY	Integer	Chave Estrangeira para P_PARTKEY
LO_SUPKEY	Integer	Chave Estrangeira para S_SUPPKEY
LO_ORDERDATE	Integer	Chave Estrangeira para D_DATEKEY
LO_ORDERPRIORITY	Character Varying	15
LO_SHIPPRIORITY	Character Varying	1
LO_QUANTITY	Integer	Valores entre 1-50
LO_EXTENDEDPRICE	Integer	<=55.450
LO_ORDETOTALPRICE	Integer	<=388
LO_DISCOUNT	Integer	Valores entre 0-10
LO_REVENUE	Integer	LO_EXTENDEDPRICE *(100-lo_discnt))/100
LO_SUPPLYCOST	Integer	
LO_TAX	Integer	Valores entre 0-8
LO_COMMITDATE	Integer	
LO_SHIPMODE	Character Varying	10

Tabela 55: Tabela PART (200,000*x(1+log2SF))

Nome do atributo	Tipo	Restrição/Tamanho
P_PARTKEY	Integer	Chave Primária
P_NAME	Character Varying	22
P_MFGR	Character Varying	6
P_CATEGORY	Character Varying	7
P_BRAND1	Character Varying	9
P_COLOR	Character Varying	11
P_TYPE	Character Varying	25
P_SIZE	Integer	Valores entre 1-50
P_CONTAINER	Character Varying	11

Tabela 56: Tabela SUPPLIER (SF*2,000)

Nome do atributo	Tipo	Restrição/Tamanho
S_SUPPKEY	Integer	Chave Primária
S_NAME	Character Varying	25
S_ADRESS	Character Varying	25
S_CITY	Character Varying	10
S_NATION	Character Varying	15
S_REGION	Character Varying	12
S_PHONE	Character Varying	17

Tabela 57: Tabela CUSTOMER (SF*30,000)

Nome do atributo	Tipo	Restrição/Tamanho
C_CUSTKEY	Integer	Chave Primária
C_NAME	Character Varying	25
C_ADRESS	Character Varying	25
C_CITY	Character Varying	10
C_NATION	Character Varying	15
C_REGION	Character Varying	12
C_PHONE	Character Varying	15
C_MKTSEGMENT	Character Varying	11

Tabela 58: Tabela DATE (dias referentes a 7 anos)

Nome do atributo	Tipo	Restrição/Tamanho
D_DATEKEY	Integer	Chave Primária
D_DATE	Character Varying	18
D_DAYOFWEEK	Character Varying	9
D_MONTH	Character Varying	9
D_YEAR	Integer	
D_YEARMONTHNUM	Integer	
D_YEARMONTH	Character Varying	7
D_DAYNUMINWEEK	Integer	

D_DAYNUMINMONTH	Integer	
D_DAYNUMINYEAR	Integer	
D_MONTHNUMINYEAR	Integer	
D_WEEKNUMINYEAR	Integer	
D_SELLINGSEASON	Character Varying	12
D_LASTDAYINWEEKFL	Character Varying	1
D_LASTDAYINMONTHFL	Character Varying	1
D_HOLIDAYFL	Character Varying	1
D_WEEKDAYFL	Character Varying	1

Referências Bibliográficas

[Abadi, 2008] ABADI, D. J. *Query Execution in Column-Oriented Database Systems*. Tese de Doutorado. Massachussets Institute of Technology, MA, Fevereiro, 2008.

[Abadi, Boncz e Harizopoulos, 2009] ABADI, D. J., BONCZ, P. A., HARIZOPOULOS, S. Column-Oriented Database Systems. In: *Proceedings of the VLB Endowment*, 2009. [s.l]: VLB Endowment, 2009, v. 2, n. 2, p.1664-1665.

[Abadi, Madden e Ferreira, 2006] ABADI, D. J., MADDEN, S., FERREIRA, M. Integrating Compression and Execution in Column-Oriented Database Systems. In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management of data*, 2006. Chicago, IL, USA: ACM, 2006, p. 671-682.

[Abadi, Madden e Hachem, 2008] ABADI, D. J., MADDEN, S. R., HACHEM, N. Column-Stores vs. Row-Stores: How Different Are They Really? In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of data*, 2008. New York, NY, USA: ACM, 2008, p. 967-980.

[Abadi et al., 2007] ABADI, D. J., MYERS, D. S., DEWITT, D. J., MADDEN, S., R. Materialization Strategies in a Column-Oriented DBMS. In: *IEEE 23rd International Conference on Data Engineering*, 2007. Istanbul: [s.n.], 2007, p. 466-475.

[Actian, 2012] ACTIAN CORPORATION. Vectorwise – *Actian Corporation*, 2012. http://www.actian.com/products/vectorwise/. Consultado na Internet em: 20/06/2012.

[Apache, 2012a] APACHE SOFTWARE FOUNDATION. *Cassandra*, 2012. http://cassandra.apache.org/. Consultado na Internet em: 10/03/2012.

[Apache, 2012b] APACHE SOFTWARE FOUNDATION. *CouchDB*, 2012. http://couchdb.apache.org/. Consultado na Internet em: 10/03/2012.

[Apache, 2012c] APACHE SOFTWARE FOUNDATION. *HBase*, 2012. http://hbase.apache.org/. Consultado na Internet em: 26/05/2012.

[Basho, 2012] BASHO. *Riak*, 2012. http://redis.io/. Consultado na Internet em: 10/03/2012.

[Boscarioli et al., 2006] BOSCARIOLI, C., BEZERRA, A., BENEDICTO, M., DELMIRO, G. *Uma reflexão sobre Banco de Dados Orientados a Objetos*. IV CONGED – Congresso de Tecnologia para Gestão de Dados e Metadados do Cone do Sul. Paraná, 2006.

[Brito, 2010] BRITO, R. W. Bancos de Dados NoSQL x SGBD Relacionais: Uma Análise Comparativa, 2010.

[Calpont, 2012] CALPONT CORPORATION. *InfiniDB*, 2012. http://infinidb.org/, Consultado na Internet em: 04/06/2012.

[Catel, 2010] CATEL, R. Scalable SQL and NoSQL Data Stores. *ACM SIGMOD Record*, New York, NY, EUA, v. 39, n. 4, p. 12-27, Dezembro, 2010.

[Citrusbyte, 2012] CITRUSBYTE. *Redis Database*, 2012. http://redis.io/. Consultado na Internet em: 10/03/2012.

[Codd, 1970] CODD, F. E. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, San Jose, California, vol. 13, no. 6, p. 377-387, Junho, 1970.

[Cormack, 1985] CORMACK, G. V. Data Compression in Database System. *Communications of the ACM*, New York, NY, USA, vol. 28, no. 12, p. 1336-1342, Dezembro, 1985.

[Date, 2003] DATE, J. C. Introdução a sistemas de bancos de dados. 8ª Edição. Local: Rio de Janeiro, Editora: Elsevier, 2003. 827p.

[Elmasri e Navathe, 2010] ELMASRI, R., NAVATHE, S. B. *Fundamentals of Database Systems*. 6ª Edição. [s.l]. Addison-Wesley, 2003. 1200 p.

[Equi4, 2009] EQUI4. Metakit embedded database library, 2009. http://equi4.com/metakit/. Consultado na Internet em: 26/05/2012.

[Firebird, 2012] FIREBIRD FOUNDATION INCORPORATED. *Firebird*, 2012. http://www.firebirdsql.org/. Consultado na Internet em: 10/03/2012.

[Gemnstone, 2012] GEMNSTONE. http://www.gemstone.com/products/gemstone. Consultado na Internet em: 10/03/2012.

[Gilbert e Lynch, 2002] GILBERT, S. LYNCH, N. Brewer's Conjecture and the Feasibility of Consistent, Avaliable, Partition-Tolerant Web-Services. *ACM SIGACT News*, New York, NY, USA, v.33, n. 2, p.51,59, Junho, 2002.

[Golfarelli, Maio e Rizzi, 1998] GOLFARELLI, M., MAIO, D., RIZZI, S. Conceptual Design of Data Wharehouses from E/R Schemes. In: Proceedings of the Thirty-First Hawaii International Conference on System Sciences, 1998. Washington, DC, USA: IEEE Computer Society, 1998, p.334-343.

[PostgreSQL, 2012] GRUPO DE DESENVOLVIMENTO GLOBAL DO POSTGRESQL. *PostgreSQL – The world's most advanced open source database*, 2012. http://www.postgresql.org/. Consultado na Internet em: 10/03/2012.

[Han et al., 2011] HAN, J., HAIHONG, E., GUAN LE; JIAN DU. Survey on NoSQL database. *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, 2011. p. 363-366, Dezembro, 2011.

[Harizopoulos et al., 2006] HARIZOPOULOS, S., LIANG V., ABADI D. J., MADDEN S. Performance Tradeoffs in Read-Optimized Databases. In: *Proceedings of the 32nd International Conference on Very Large Data Bases*, 2006. Seoul, Korea: VLDB Endowment, 2006, p. 487-498.

[Heuser, 2008] HEUSER, C. A.; Projeto de Banco de Dados. 6ª Edição. Instituto de Informática da UFRGS: BookMan, 2008. 282 p.

[HP, 2012] HP. *Vertica* © 2012. http://www.vertica.com/. Consultado na Internet em: 13/06/2012.

[IBM, 2012] IBM. *DB2 Database Software*, 2012. http://www-01.ibm.com/software/data/db2/. Consultado na Internet em: 10/03/2012.

[Infobrigth, 2012] INFOBRIGTH. *Big Data Analytics | Large Scale Data Analytics Tool © 2012 Infobrigth*. www.infobrigth.com, Consultado na Internet em: 20/06/2012.

[Informix, 2012] INFORMIX. *Informix*, 2012. http://www.informix.com.br/. Consultado na Internet em: 10/03/2012.

[Inmon, 2012] INMON, W. H. *What is Data Warehouse?*, 2012. https://www.business.auc.dk/oekostyr/file/What_is_a_Data_Warehouse.pdf. Consultado na Internet em: 09/07/2012.

[Khoshfian et al., 1987] KHOSHAFIAN, S., COPELAND G. P., JAGODIS T., BORAL H., VALDURIEZ P. A Query Processing Strategy for the Decomposed Storage Model. In: P*roceedings of the Third International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 1987, 636-643.

[Marcus, 2011] MARCUS A. The Architecture of Open Source Applications - Elegance, Evolution, and a Fearless Hacks, Capítulo 13: The NoSQL Ecosystem, Editora Kindle, EUA, 2011.

[Matei, 2010] MATEI, G. Column-Oriented Databases, an Alternative for Analytical Environment. *Database Systems Journal*, Romanian Comercial Bank, Bucharest, Romania, v. 1, n. 2, p. 3-16, Fevereiro, 2010.

[Mcknight, 2011] MCKNIGHT, W. *Best Pratices in the Use of Columnar Databases*, 2011. http://www.calpont.com/doc/Calpont_Whitepaper-Best-Practices-in_the_Use_of_Columnar_Databases.pdf. Consultado na Internet em: 16/03/2012.

[MemcacheDB, 2009] MEMCACHEDB. *MemcacheDB*, 2009. http://memcachedb.org/. Consultado na Internet em: 10/03/2012.

[Microsoft, 2012] MICROSOFT. *©2012 SQL Server Microsoft Corporation* . www.microsoft.com/sqlserver/en/us/default.aspx. Consultado na Internet em: 10/03/2012.

[Mishra e Eich 1992] MISHRA, P., EICH, M. H. Join Processing in relational databases. *ACM Computing Surveys (CSUR)*, New York, NY, USA, v. 24, n. 1, p. 63-113, Março, 1992.

[MonetDB, 2012] MONETDB. *MonetDB DV*, © 2012. http://www.monetdb.org/Home. Consultado na Internet em: 18/05/2012.

[MongoDB, 2012] MONGODB. *mongoDB*, 2012. http://www.mongodb.org/. Consultado na Internet em: 10/03/2012.

[Neil, Neil e Chen, 2009] O'NEIL P., O'NEIL B., CHEN, X. *The Star Schema Benchmark (SSB)*, 2009. http://www.cs.umb.edu/~poneil/StarSchemaB.PDF. Consultado na Internet em: 17/06/2012.

[Neil e Grafe, 1995] O'NEIL, P., GRAEFE, G. Multi-Table Joins Through Bitmapped Join Indices. *ACM SIGMOD Record*, New York, NY, USA, v. 24. n. 3, p. 8-11, Setembro, 1995.

[Objectivity, 2012] OBJECTIVITY/DB. *Objectivity/DB*, 2012. http://objectivity.com/. Consultado na Internet em: 10/03/2012.

[Oracle, 2012a] ORACLE. *Hardware and Software, Engineered to Work Together*, 2012. http://www.oracle.com/us/products/database/overview/index.html?origref=http://www.oracle.com/index.html. Consultado na Internet em: 10/03/2012.

[Oracle, 2012b] ORACLE. MySQL – *The World's most popular Open Source Database*, 2012. http://www.mysql.com/. Consultado na Internet em: 10/03/2012.

[Oracle, 2012c] ORACLE. Oracle Database 11*g*, 2012. http://www.oracle11g.com.br. Consultado na Internet em: 14/10/2012.

[Prichett, 2008] PRICHETT, D. Base: An Acid Alternative, *Queue – Object-Relacional Mapping*, Nova York, NY, EUA, v. 6, n. 3, p. 50-55, Maio/Junho, 2008.

[Red Brick Systems, 1995] RED BRICK SYSTEMS. *Star Schema Processing for Complex Queries*, 1995. http://www-306.ibm.com/software/data/informix/redbrick/. Consultado na Internet em: 26/06/2012.

[Silberchatz, Korth e Sudarshan, 2010] SILBERCHATZ, A., KORTH, F. H., SUDARSHAN S. Database System Concepts. 6ª Edição. [s.l]: McGraw-Hill Companies, Incorporated, 2010.

[Scalzo, 2012] SCALZO, B. *Data Warehouse Benchmark: Comparing Calpont InfiniDB® and a Row Based Database*, 2010. http://www.calpont.com/data-warehouse-benchmark. Consultado na Internet em: 16/03/2012.

[SquirreL, 2012] SQUIRREL SQL. *Universal SQL Client Version 3.3.0*, 2012. http://squirrel-sql.sourceforge.net/. Consultado na Internet em: 08/07/2012.

[Stonebreaker et al., 2005] STONEBREAKER, M., ABADI D. J., BATKIM A., CHEN, X., CHERNIACK, M., FERREIRA, M., LAU, E., LIN, A., MADDEN, S., O'NEIL, E., O'NEIL, P., RASIN, A., TRAN, N., ZDONIK, S. CStore: A Column-oriented DBMS. In: *Proceedings of the 31st international conference on Very large data bases*, 2005. Trondheim, Norway: VLDB Endowment, 2005, p. 553-564.

[Sybase, 2012] SYBASE. Sybase Inc. – v 7.6, 2012. http://www.sybase.com.br/products/datawarehousing/sybaseiq/. Consultado na Internet em: 13/06/2012.

[TPC-H, 2012] TPC-H. TCP – *Transaction Processing Performance Council*, 2012.

http://www.tpc.org/tpch/. Consultado na Internet em: 27/06/2012.

[Tsichirtzis e Lochovsky, 1976] TSICHIRTZIS, D. C., LOCHOVSKY, F. H. Hierchical Data-Base Management: A Survey. *Computing Surveis*, vol. 8, no. 1, Março, 1976.

[Versant, 2012] VERSANT CORP. db4ojects, 2012. http://www.db4o.com/. Consultado na Internet em: 10/03/2012.

[Weininger, 2002] WEININGER, A. Efficient Execution of Joins in a Star Schema. In: *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, 2002. New York, NY, USA: ACM, 2002, p. 542-545.

[Westmann, 2000] WESTMANN, T., KOSSMANN, D., HELMER, S., MOERKOTTE, G. The Implementation and Performance of Compressed Databases. *ACM Sigmod Record*, New York, NY, USA, vol. 29, no. 3, p. 55-67, Setembro, 2000.

[Widebase, 2012] WIDEBASE. *Widebase Projetct*, 2012. http://widebase.github.com/. Consultado na Internet em: 13/06/2012.

[Ziv e Lempel, 1977] ZIV, J., LEMPEL, A. A Universal Algorithm for Sequencial Data Compression. *IEEE Transactions on Information Theory - TIT*, [s.l], vol. 23, n. 3, p. 337-343, Maio, 1977.

[Zukowsky et al., 2005] ZUKOWSKY, M., BONCZ, P. NES, N., HÉMAN, S. MonetDB/X100 – A DBMS In the CPU Cache. *IEE Data Eng. Bull*, vol. 28, n. 2, p. 17-22, Agosto, 2005.

[Zukowsky et al., 2006] ZUKOWSKY, M., HÉMAN, S., NES, N., BONCZ, P. Super-Scalar RAM-CPU Cache Compression. In: *Proceedings of the 22nd International Conference on Data Engineering*, 2006. Washington, DC, USA: IEE Computer Society, 2006, p. 59.