Avaliando o Impacto da Modelagem de Dados em Aplicações OLAP utilizando SGBD Relacional e Colunar

Letícia Torres*, Gustavo Rezende Krüger†, Marcio Seiji Oyamada* e Clodis Boscarioli*

*Ciência da Computação
Universidade Estadual do Oeste do Paraná
Cascavel – PR, Brasil 85819–110
Email: {atorresleticia, msoyamada, clodisboscarioli}@gmail.com

†Orbit Sistemas
Cascavel – PR, Brasil 85819–720
Email: gustavorekr@gmail.com

***Abstract*—Data Warehouses has consolidate as the decision support technology used by Organizations that uses OLAP applications to access the stored data. As these data volume increases more efficient approaches to process them are needed. To do so, both traditional relational databases management systems and columnar ones can be used, each one with their advantages over the Data Warehouse modeling. More normalized models are traditional among tuple-oriented relational databases, whereas denormalized ones bring a better performance in columnar DBMS. A comparative study between MonetDB and PostgreSQL DBMS using TPC-H as a benchmark is presented here, to investigate which one is indicated to manage a Data Warehouse in information access. The results confirmed that, isolated, in denormalized environments MonetDB excels, while PostgreSQL is better for normalized modeling. In general, MonetDB stands out compared to PostgreSQL, with performance gains of almost 500% on normalized model, and over 1000% on the denormalized one.**

***Resumo*—*Data Warehouses* se consolidaram nas Organizações como tecnologia de apoio à tomada de decisão utilizando aplicações OLAP sobre os dados armazenados. Conforme o volume destes dados aumenta, tornam-se necessárias abordagens mais eficientes para seu processamento. Para tal tanto sistemas gerenciadores de banco de dados relacionais tradicionais quanto os colunares podem ser utilizados, e cada qual tem suas vantagens conforme a modelagem do *Data Warehouse*. Modelagens mais normalizadas são tradicionais entre os relacionais orientados a linha, enquanto que modelagens denormalizadas trazem desempenho superior em SGBD colunares. Um estudo comparativo entre os SGBD PostgreSQL e MonetDB utilizando o TPC-H como *benchmark* é aqui apresentado, investigando qual é o mais indicado para gerenciar um *Data Warehouse* na recuperação de informações. Os resultados experimentais confirmam que, isoladamente, em ambientes denormalizados o MonetDB se destaca, enquanto que o PostgreSQL é melhor em ambientes normalizados. Como um todo o MonetDB mostrou-se superior ao PostgreSQL obtendo ganhos de desempenho de quase 500% para o modelo normalizado e superior a 1000% para o denormalizado.**

I. Introdução

Organizações cada vez mais se apoiam no armazenamento de dados a fim de recuperar informações para tomadas de decisões de maneira ágil e eficaz. Estes dados são mantidos ao longo do tempo e por vezes, são armazenados em sistemas que diminuem a eficiência no acesso e processamento de dados por serem heterogêneos, autônomos e geograficamente distribuídos [1].

Como forma de tratar um grande volume de dados, persisti-los e tomar decisões de maneira inteligente e eficiente a tecnologia de *Data Warehouse* é aplicada, que consiste em um repositório de dados homogêneo, local e centralizado capaz de fornecer informações de diversas fontes de dados [1]; e seu conceito está atrelado à criação de um ambiente com acesso a dados para tomar decisões de forma rápida e efetiva.

Para uma boa decisão de negócio além de um repositório é necessário uma aplicação que o acesse e traga informações. Aplicações OLAP (*Online Analytical Processing*) estão fortemente associadas a *Data Warehouses* por tratarem de uma análise multidimensional de dados e consultas analíticas. À combinação de um *Data Warehouse* e aplicações OLAP dá-se o nome de ambiente OLAP.

Existe uma série de princípios que devem ser seguidos ao projetar e implantar um ambiente OLAP, destacando-se a rapidez com que os dados devem ser recuperados [1]–[3]. Assim, um dos aspectos

técnicos a ser considerado são os Sistemas Gerenciadores de Bancos de Dados (SGBD) usados como gerenciadores de um *Data Warehouse*, e duas classes de SGBD podem ser utilizadas para implementar este ambiente: SGBD relacionais orientados à linha e à coluna. Dada a importância na escolha de um SGBD e as diferentes soluções apresentadas por cada um é viável o uso de um *benchmark* para a análise e conclusão de qual SGBD escolher. Tendo em vista o escopo da decisão de negócio, busca-se um *benchmark* que englobe e trabalhe com foco nisto e que traga uma configuração que se aplique tanto em SGBD relacionais quanto NoSQL. Com esse objetivo, a organização TPC [4] desenvolveu um *benchmark* para avaliar o desempenho de SGBD atuando em ambiente OLAP, o TPC-H [5].

Além dos SGBD, deve-se considerar ainda a modelagem do ambiente para análise de dados. Segundo Bax e Souza [6], algumas empresas utilizam um ambiente com modelagem normalizada devido a facilidade de manutenção, fácil entendimento da relação entre as entidades e atributos e uma visão mais clara do sistema. Por outro lado, modelos denormalizados são bastante utilizados devido ao ganho de desempenho nas consultas, pela baixa complexidade destas e pela diminuição do número de entidades – por consequência das operações de junção entre as tabelas.

Dadas as variáveis SGBD relacional orientado à linha e à coluna, bem como ambientes com modelagem denormalizada e normalizada, o objetivo aqui é realizar um estudo comparativo entre SGBD como gerenciadores de um *Data Warehouse* utilizando-se de um *benchmark*. Os SGBD escolhidos foram o PostgreSQL como relacional tradicional, devido a sua ampla utilização no mercado, boa documentação e uso da linguagem SQL sem muitas modificações; e o MonetDB como colunar, por também utilizar-se da linguagem SQL, possuir interface simples através de linhas de comando, e ser o pioneiro entre os bancos colunares desenvolvidos.

II. Persistência de Dados e Recuperação de Informações

Como mencionado, empresas estão cada vez mais se apoiando na tomada de decisões a partir de dados persistidos em seus *Data Warehouses*. Inmon [7] define um *Data Warehouse* como base de todos os sistemas de suporte a decisão, atualmente definido como ambiente OLAP, e ainda como sendo uma coleção de dados não-volátil, orientado ao assunto principal da organização, integrado e variante no tempo. Estruturalmente, o repositório é definido como uma base de dados homogênea, local e centralizada [1].

Para analisar os dados armazenados no *Data Warehouse* além de implementá-lo é necessário que alguma aplicação leia seu conteúdo e apresente-o de forma gráfica e intuitiva ao analisador. Aplicações OLTP (*Online Transaction Processing*) são amplamente utilizadas por bancos operacionais através de transações que seguem o conceito de ACID – atomicidade, consistência, isolamento e durabilidade; entretanto, *Data Warehouses* trabalham com uma grande massa de dados, suporte a decisão e são intensivos para consultas analíticas, que acessam milhões de registros [8], combinando diversos dados a fim de retornar uma informação útil para o mercado. Sendo assim, dados históricos armazenados ao longo do tempo na empresa, taxa de vazão de consultas e, sobretudo, o tempo de resposta são mais importantes que pequenas transações.

Fugindo um pouco da ideia de ACID, aplicações OLAP se aplicam melhor no contexto apresentado por não focarem em pequenas transações. Elas têm por objetivo identificar tendências, padrões de comportamento e anomalias, bem como relações entre dados aparentemente não relacionados [2] sob um grande volume de dados. Geralmente as consultas em uma aplicação analítica são longas, levam tempo para executar e estão interessadas em atributos específicos. Ainda, o processo de inserção de dados em um ambiente OLAP é um pouco mais complexo que sistemas OLTP por seguir um processo fim-a-fim conhecido como ETL (extração, transformação e carga) [9]

O processo ETL faz parte da arquitetura de um *Data Warehouse*, e consiste em *extrair* dados de fontes heterogêneas para inseri-los em uma área de armazenamento temporária, denominada *staging area*, onde então será feita uma *transformação* de dados – limpeza, homogeneização, remoção de redundância e atribuição de chaves aos dados, de acordo com a modelagem do *Data Warehouse*; e por fim o *carregamento* dos registros na área de estruturação, que consiste no *Data Warehouse* em si. Esta área de estruturação trata de como os dados serão organizados, armazenados e disponibilizados para as aplicações analíticas, e é nela que a modelagem conceitual é definida.

Normalmente bancos de dados operacionais utilizam a tradicional modelagem relacional normalizada até a Terceira Forma Normal (3NF). Aplicações OLAP têm por base a modelagem dimensional, ou multidimensional, que faz uma analogia com um cubo para representar seus dados, uma vez que uma informação pode ser vista através de n dimensões.

A. Modelagem Dimensional

Enquanto que uma aplicação OLTP visa um alto nível de normalização, aplicações OLAP priorizam desempenho e portanto um nível maior de denormalização. Ainda não há, porém, consenso acerca da modelagem conceitual no campo do *Data Warehouse* [10]. Apesar disso, a modelagem Entidade-Relacional, e a Relacional por consequência, é voltada para consultas que realizam associações entre dados ao invés de consultas analíticas [3] que visam construir e trazer maior clareza sobre cenários de decisão.

Em sua concepção, a modelagem dimensional é composta por tabelas fato e tabelas dimensão.

1) Tabelas Fato: De acordo com Kimball [3], uma tabela fato, ou apenas fato, traz a lista de atributos que representam as regras, ou métricas, de negócio. Fatos geralmente são descritos como atributos numéricos, relacionados com quantidades; e aditivos, pois uma aplicação OLAP nunca irá retornar somente uma tupla fato, mas sim várias tuplas onde a operação mais recorrente é a soma de todas elas.

Atributos textuais não são definidos como um fato. Na maioria das vezes descrevem algo e devem estar inseridos em tabelas dimensão pois há maior chance de estarem relacionados com os atributos dimensão e consomem menos espaço em disco. Outro ponto a se ater é a importância de se evitar incluir dados com zeros que não representam dados úteis, ou representando nada, pois a inclusão de dados sem significado sobrecarregariam o *Data Warehouse* sem motivo.

2) Tabelas Dimensão: Como dito, tabelas fato não trazem consigo atributos textuais de negócio, apenas métricas. As entidades responsáveis por isto são as entidades dimensão, que contém descrições que tornam os fatos mais claros. Não é estranho encontrar tabelas com mais de 100 atributos e poucas tuplas, pois a intenção das dimensões é descrever as regras de negócio.

É nas tabelas dimensão que são realizados os filtros de consulta. Por filtros, entende-se agrupamentos, padrões e ordenações por exemplo. De forma a exemplificar, se fosse desejado buscar as vendas por nome de fornecedor em um determinado intervalo de data, os atributos "nome de fornecedor"e "data"deveriam estar armazenados em tabelas dimensão. Segundo Kimball [3], quanto mais bem descritos os atributos dimensão, melhor o *Data Warehouse* é, tornando a qualidade do Warehouse dependente das entidades de dimensão.

Os modelos dimensionais compostos por tabelas fato e dimensão aplicados no escopo do *warehouse* são comumente conhecidos como modelos estrela (modelos *star*, ou *star join*) [1], [3], [7], [10], pois é organizado de forma a ter um único fato com várias dimensões. Este modelo torna a modelagem simples, trazendo benefícios ao desempenho na recuperação de dados devido a redução no número de junções; ademais, é flexível à alterações e expansões.

Com a afirmação de que modelos normalizados são mais fáceis para se dar manutenção não sendo necessário, por exemplo, alterar exaustivamente vários registros de um atributo em uma relação pois ele não será repetido, alguns modelos *star* podem ser trabalhados para oferecer suporte à hierarquia de atributos das tabelas dimensões, criando novas tabelas denominadas subdimensão. Estes modelos *star-normalizados* são conhecidos como *snowflake* e caracterizam bem modelos relacionais de dados.

Para comparar qual modelagem pode trazer mais benefícios na recuperação de informações, é crucial o uso de uma ferramenta que consiga fazer o gerenciamento do *Data Warehouse*, desde a modelagem até o acesso aos dados, de maneira inteligente e que traga alguma vantagem para o administrador do banco. Elmasri e Navathe [11] citam os sistemas gerenciadores de banco de dados como solução para isto. Dentre as vantagens que os tornam úteis como gerenciadores de *Data Warehouses*:

3) Controle de redundância: garante que dados não serão gravados mais de uma vez no modelo.

4) Restrição de acesso: usuários não autorizados não terão acesso ao banco.

5) Índices: tornam os acessos a disco mais eficientes.

6) Restrição de integridade: dados não serão inseridos de maneira inconsistente.

7) Persistência de dados: uma vez escritos, os dados prevalecem no modelo.

8) Backup e restauração: é possível salvar e recuperar posteriormente informações do banco.

Uma classe bem conhecida de SGBD é o modelo relacional [12] orientado à linhas, ou tuplas. Sua criação se deu por Edgar Codd em 1970 [13] e é estruturado com base em um conjunto de entidades que tratam objetos reais, representadas por tabelas; contendo atributos e cada qual um valor. Os dados inseridos neste banco serão os valores destes atributos e ficam organizados sob a forma de tuplas, que corresponde a uma linha da tabela. A relação entre tabelas é feita através de chaves, primária e estrangeira, onde a chave primária garante a unicidade de um registro na tabela e a chave estrangeira a relação de integridade entre tabelas. Vale apontar que o modelo relacional segue a ideia ACID e estão bastante atrelados à modelagens mais normalizadas, se aproximando do modelo *snowflake* de *Data Warehouse*. Exemplos

de SGBD relacionais são: MySQL, PostgreSQL, Oracle e SQLServer.

Embora muito utilizados, modelos relacionais possuem algumas limitações como escalabilidade, complexidade, até mesmo a própria linguagem SQL [14] e o próprio fato de todas as informações de uma entidade serem mantidas juntas em linhas. Para um ambiente analítico limitações como estas podem acarretar em uma degradação no desempenho, pois além das já citadas, outra característica de consultas analíticas é que elas percorrem o banco processando somente os atributos necessários. Mesmo a adição de índices não ajudaria no desempenho, pois o elevado número de diferentes consultas faz com que seja necessário mais processamento para ler os índices [15]. Como forma de aprimorar a recuperação de dados uma diferente abordagem de SGBD pode ser aplicada no ambiente OLAP, os colunares.

Ao contrário do modelo relacional, este armazena cada *coluna* do banco separadamente, tendo todas as instâncias de um mesmo atributo armazenadas juntas. Armazenando dados desta forma reduz-se o tempo de acesso e escrita em disco [15], [16], pois em uma leitura são retornados apenas os atributos solicitados pela consulta, deixando de lado a leitura de uma tupla inteira [17]; e o SGBD torna-se mais eficiente em operações matemáticas de agregação [15] como encontrar o valor máximo, mínimo, média, etc.

Outro fator que torna vantajoso o uso de um SGBD colunar é a compressão de dados. De acordo com Abadi; Madden e Ferreira [18], a compressão é mais eficiente em um SGBD colunar, pois aumentam-se as chances de haver atributos iguais em linhas adjacentes na mesma coluna, visto que são do mesmo tipo. Em um SGBD orientado à linha para realizar compressão tuplas inteiras devem coincidir. Atributos com valor *null*, por exemplo, são tratados mais facilmente em um SGBD orientado à coluna, pois ele pode ser tratado como um valor a ser comprimido. Exemplos de SGBD colunar são o MonetDB, pioneiro em SGBD colunares; LucidDB e o C-Store.

III. TPC-H e o Benchmarking

Devido à variação de SGBD que podem ser utilizados em ambientação OLAP, para selecionar o mais adequado a cada modelo de dados a realização de um *benchmark* é o mais indicado, visto que sua função é identificar e definir os melhores padrões de excelência para produtos, serviços e processos, buscando melhorar o desempenho das organizações [19], [20].

De acordo com o manual de especificação fornecido pelo TPC [21], o TPC-H é um *benchmark* de suporte à decisão de negócios, constituído de uma série de consultas comerciais *ad-hoc* e modificações simultâneas de dados com finalidade de retratar a realidade das empresas. Ele representa sistemas de suporte a decisão que examinam grandes volumes de dados; executam consultas com um alto grau de complexidade; e respondem questões críticas de negócio. Como modelagem o TPC-H propõe um ambiente *snowflake*, descrito na Seção III-A, e os dados a serem inseridos nesse modelo são gerados através de um software desenvolvido pelo próprio TPC, o DBGen. A organização também propõe 22 consultas analíticas para recuperar dados do ambiente respondendo a alguma questão de negócio; e assim como a geração de dados o TPC também traz o QGen, software para geração das consultas.

O TPC-H não apresenta em seu manual de especificação uma modelagem mais denormalizada de dados que seja similar à modelagem *star*. Para tanto, é necessário construir um ambiente denormalizado para as devidas comparações. A descrição desta modelagem adaptada se encontra na Seção III-B.

A. Ambiente Snowflake

O modelo de ambiente proposto pelo TPC-H é um esquema normalizado baseado na modelagem *snowflake*. Ele é composto por oito entidades, sendo que destas, seis têm o tamanho multiplicado pelo fator de escala analisado; enquanto que as demais, *nation* e *region*, têm tamanho fixo. O relacionamento segue a mesma ideia do modelo dimensional entre fatos e dimensões, onde várias dimensões se relacionam com um fato através de chaves estrangeiras na entidade de fato. A diferença neste esquema é que não existe apenas um fato nas regras de negócio, mas dois, *orders* e *lineitem*. O relacionamento entre as tabelas do esquema *snowflake* pode ser obtido através do manual de especificação do TPC-H [21].

Os tipos de dados atrelados aos atributos de uma entidade podem ser identificadores, indicando um número inteiro que define a chave primária da entidade; inteiros e decimais; textos fixos e variáveis e datas.

B. Ambiente Star

As modificações no modelo do ambiente *snowflake* se fundamentaram na criação de um modelo como o apresentado pela modelagem *star*, com uma entidade fato central descrita por dimensões, eliminando-se tabelas subdimensão e unindo atributos fato de diferentes entidades. O modelo é ilustrado na Figura 1.

A primeira modificação no modelo original foi a exclusão das entidades subdimensão *nation* e

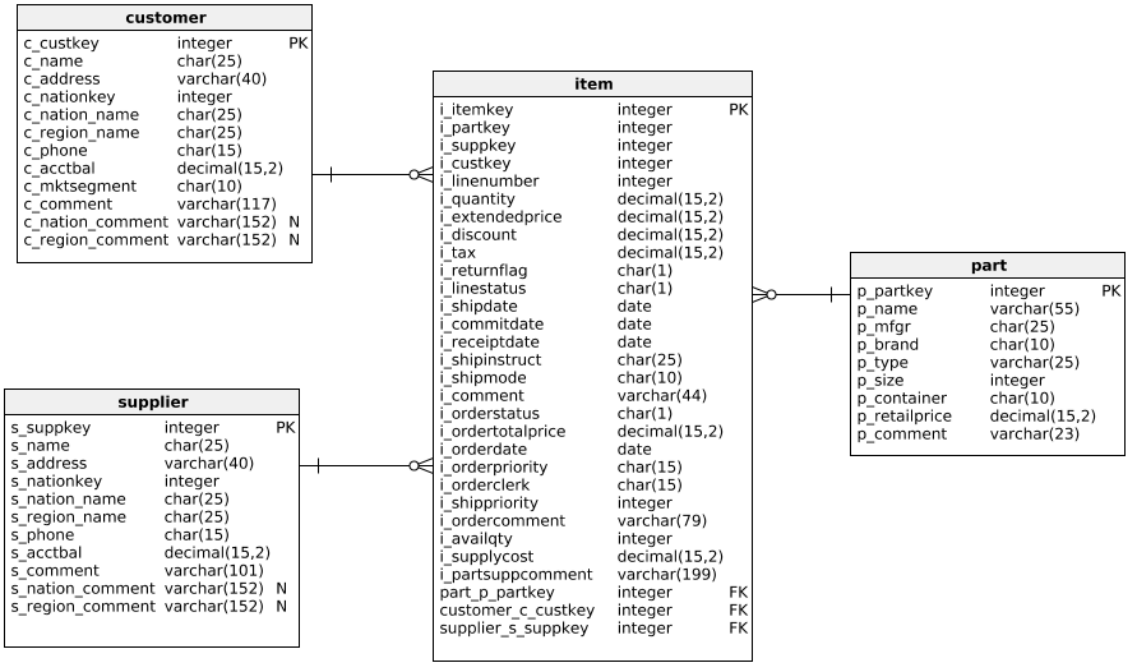

Figura 1. Esquema do ambiente *star*

region. Incluiu-se portanto os atributos `n_name`, `n_comment`, `r_name` e `r_comment` nas entidades *supplier* e *customer*.

Com a intenção de criar um único fato no esquema denormalizado, observou-se (i) a possibilidade de unir as entidades *lineitem* e *orders* em uma única entidade renomeada *item*. Isto também eliminaria algumas junções realizadas entre as duas entidades nas consultas do TPC-H. Também se nota que (ii) as entidades *part* e *supplier* se relacionam com *lineitem* através da entidade intermediária *partsupp*. Optou-se assim por incluir os atributos de *partsupp* na nova entidade *item*, excluindo a primeira do esquema e mantendo o relacionamento de *supplier* e *part* diretamente com *item*.

O esquema final agora possui apenas uma entidade fato: *item*; e três entidades dimensão: *part*, *customer* e *supplier*. Os dados presentes nas entidades são do mesmo tipo descritos na seção anterior. Tanto as consultas quanto os dados foram modificados de forma a se adequar ao novo esquema.

Como o ambiente modificado não faz parte do TPC-H, o DBGen não tem suporte à geração automática de dados segundo a nova modelagem. Com isso, para que seja feita a nova geração de dados optou-se por uma abordagem que conta com o auxilio dos dados já gerados pelo DBGen. Após populados os registros no banco original são executadas junções a fim de projetar os atributos requeridos por cada entidade do modelo *star*. O resultado desta projeção é inserido nas tabelas da modelagem adaptada. O mesmo se aplica às consultas, como o QGen não têm suporte à diferentes modelagens, adaptou-se manualmente as 22 consultas.

C. O Processo de Avaliação

Após a geração dos arquivos foi criado em cada SGBD um esquema correspondente às classes utilizadas, tanto para o ambiente *snowflake* quanto para o *star*. Cada SGBD foi então executado sobre os dois ambientes propostos. Em cada ambiente foi executado o teste de desempenho, que consiste de duas execuções: o teste de força e o teste de vazão. O tempo resultante de cada etapa foi utilizado para calcular o desempenho do SGBD, utilizando a unidade de medida *QphH@Size* – quantidade de consultas executadas por hora dada uma classe de tamanho de banco de dados, onde quanto maior o valor melhor é o desempenho do SGBD. As execuções em geral são compostas por:

- *SF*, que representa a classe do banco de dados.
- Q_i, que representa uma consulta, onde $1 \leq i \leq 22$.
- *S*, que define o número de sessões de consulta do Teste de Vazão.
- s, que representa uma dada sessão, onde $1 \leq s \leq S$.
- T_s, que representa o tempo, em segundos, resultante do processo inteiro.
- RF_j, que representa uma função de atualização (RF) onde:

– RF-1: inserção de novos registros. São gerados novos dados para as tabelas fato com quantidade proporcional ao fator de escala.
– RF-2: remoção de registros antigos. São removidos dados antigos das tabelas fato, e o número de registros deletados deve ser igual ao número de registros inseridos. Note que não necessariamente são removidos os dados inseridos na RF-1.

D. Teste de Força

O Teste de Força objetiva medir a taxa de consultas por hora do sistema com um único usuário ativo. Neste teste foi criada uma única sessão com o respectivo SGBD e as seguintes instruções foram executadas:

1) Execução da RF-1.
2) Execução de cada consulta proposta pelo TPC-H, ou adaptada, de forma sequencial uma única vez até a última consulta.
3) Execução da RF-2.

Foi armazenado ao fim do teste o tempo em segundos resultante de cada consulta, bem como o tempo de execução de cada função. Este resultado foi utilizado na Equação 1, que calcula a média geométrica entre o produto de todas as consultas e as duas funções de atualização.

$$Power = \frac{3600 * SF}{\sqrt[24]{\prod_{i=2}^{i=22} Q(i,0) * \prod_{j=1}^{j=2} RF(j,0)}} \quad (1)$$

E. Teste de Vazão

Este teste mede a capacidade do sistema de processar a maior quantidade possível de consultas no menor intervalo de tempo. Aqui o TPC-H exige um número mínimo de sessões de consulta de acordo com a classe de tamanho do banco de dados, como mostra a Tabela I.

Tabela I
NÚMERO MÍNIMO DE SESSÕES PARA UMA CLASSE DE BANCO DE DADOS

Classe do Banco de Dados	Número de Sessões
1 GB	2
10 GB	3
30 GB	4
100 GB	5
300 GB	6
1000 GB	7
3000 GB	8
10000 GB	9
30000 GB	10
100000 GB	11

Foi criada uma sessão para executar as funções de atualização e *N* sessões para executar as consultas, de acordo com a Tabela I. As instruções foram executadas em paralelo, sendo elas:

- Cada dupla de RF-1 e RF-2 deverão ser executadas sequencialmente *N* vezes, onde *N* é o número de sessões para a execução de consultas.
- Para cada sessão de consultas, cada consulta será executada sequencialmente até a última.

Ao fim do teste, foi armazenado o tempo em segundos da execução do processo inteiro. O processo inicia quando a primeira sessão, seja de consulta ou de função, executa sua instrução e finaliza quando a última sessão recebe uma resposta. O resultado foi utilizado para calcular o desempenho dos SGBD para o teste conforme a Equação 2.

$$Throughput = \frac{S * 22 * 3600}{T_s * SF} \quad (2)$$

O resultado dos cálculos de força e vazão foram ponderados de maneira uniforme para calcular o desempenho final do SGBD de acordo com a fórmula:

$$QphH@Size = \sqrt{Power * Throughput} \quad (3)$$

IV. RESULTADOS E DISCUSSÃO

O experimento para análise de desempenho foi feito sob uma base de dados de 1GB. Além dos SGBD PostgreSQL e MonetDB, considerou-se previamente a inclusão do MySQL nos testes de desempenho devido a sua popularidade. Seu uso foi logo descartado em virtude da notória discrepância de tempo para carga de dados se comparado aos outros dois. Essa diferença pode ser vista na Tabela II, que mostra o tempo da inserção de dados em segundos, e é ilustrada na Figura 2.

Tabela II
CARGA DE DADOS

	MySQL	PostgreSQL	MonetDB
Snowflake	205,49	84,688	33,54
Star	317	170	61

Para obter um melhor desempenho no carregamento de dados as restrições de chave foram realizadas após a inserção dos dados nos SGBD. Caso a restrição fosse feita na criação do banco a cada novo registro inserido a chave primária seria verificada. Esta carga de dados também foi realizada através dos SGBD de forma direta, sem o uso de *drivers* e outras ferramentas que possam atuar como cliente para conectar-se ao banco, assim

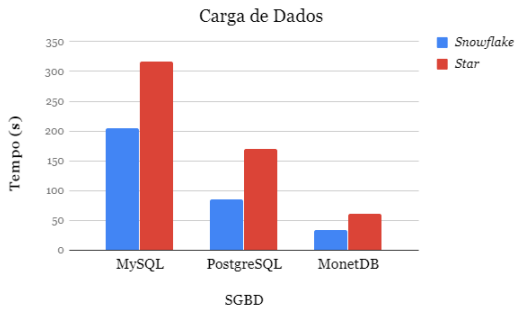

Figura 2. Gráfico da carga de dados

evitando incluir o tempo de latência destas no resultado.

Logo ao analisar os resultados de carregamento de dados notou-se que um banco de dados denormalizado apresentou certa demora na inserção de dados em relação a um normalizado. Isso se dá por causa da redundância de dados existente neste tipo de modelo, fazendo com que existam mais atributos duplicados em uma mesma entidade, o que aumenta o tamanho das entidades a serem inseridas. Toma-se como exemplo a entidade *nation*: no modelo *snowflake* ela é uma tabela subdimensão e possui 25 registros apenas, enquanto que no modelo denormalizado ela faz parte das tabelas dimensão *supplier* e *customer*, fazendo com que estas duas entidades tenham mais bytes armazenados. A Tabela III mostra o tamanho dos arquivos gerados para cada modelo. Note que *part* não tem seu valor alterado por não sofrer alteração entre as modelagens.

Tabela III
TAMANHO EM BYTES DAS ENTIDADES

	Snowflake	*Star*
Supplier	1.409.184	2.990.430
Customer	24.346.144	48.055.517
Part	24.135.125	24.135.125
Lineitem/Item	759.863.287	2.226.620.122

Após a inserção de dados, somente os SGBD PostgreSQL e MonetDB foram utilizados para o *benchmarking*. Optou-se por desenvolver o processo do TPC-H utilizando o *driver* JDBC[1] pela simplificação que seu uso traz ao desenvolvimento de aplicações e por dar suporte à vários SGBD. Porém, há que considerar a latência do *driver* no tempo de execução devido à ponte entre a chamada de um procedimento do JDBC até a conexão com o SGBD.

A execução começa com o teste de força seguido imediatamente do teste de vazão. Logo nos resultados apresentados por estas execuções notou-se a diferença de desempenho do PostgreSQL e do MonetDB – atentando-se para o fato de que quanto maior o valor, melhor. As Tabelas IV e V mostram os resultados obtidos em cada execução em consultas por hora (*qph*).

Tabela IV
TESTE DE FORÇA

	PostgreSQL	MonetDB
Snowflake	4,5711	46,5462
Star	2,2154	53,0620

Tabela V
TESTE DE VAZÃO

	PostgreSQL	MonetDB
Snowflake	1566,7346	5524,4257
Star	785,1225	7243,2810

Tendo os resultados do teste de força e vazão, calculou-se o resultado final de desempenho do *benchmark* para se obter a quantidade de consultas executadas por hora no sistema, conforme a Equação 3. A Tabela VI mostra o resultado final em *qph* e a Figura 3 ilustra graficamente os dados da tabela.

Tabela VI
RESULTADO DO *benchmark* – QPHH@1GB

	PostgreSQL	MonetDB
Snowflake	84,6271	507,0913
Star	41,7061	619,9545

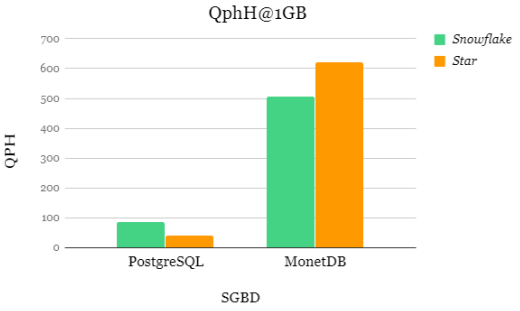

Figura 3. Gráfico do resultado final

Analisando de fato os resultados do *benchmark*, de maneira geral o SGBD que obteve melhores resultados foi o MonetDB, desde a execução isolada dos testes de força e vazão até o resultado final. Em relação aos ambientes o MonetDB denormalizado apresentou resultados melhores, o que reforça o uso de um SGBD colunar em ambientes analíticos

[1]http://www.oracle.com/technetwork/java/javase/jdbc/index.html

denormalizados sob um grande volume de dados. Se tomar apenas o SGBD, o desempenho baixo do PostgreSQL era esperado ao se comparar com um SGBD colunar devido ao armazenamento e leitura aprimorados do MonetDB, tanto para um ambiente normalizado quanto para um denormalizado.

Comparando os resultados do PostgreSQL de maneira isolada percebeu-se que a lógica nos resultados não é a mesma do MonetDB, e que outra teoria é reforçada: a de que SGBD relacionais orientados à linha estão fortemente atrelados a modelagens mais normalizadas e são otimizados para se adequar melhor a esta modelagem em ambientes transacionais.

V. Conclusão e Trabalhos Futuros

O SGBD MonetDB foi superior com quase 500% de ganho no desempenho final em relação ao PostgreSQL para o ambiente *snowflake*, representando uma diferença de seis vezes no resultado final. Também foi melhor com mais de 1000% de ganho ao orientado à linha na modelagem *star*, tendo quase 15 vezes o valor do PostgreSQL. Estes resultados como um todo corroboram a ideia de que o uso de SGBD colunar traz vantagens a ambientes OLAP, por adaptar-se melhor à modelagem comumente utilizada por *Data Warehouses* e às suas consultas.

Sugere-se a inclusão de outros SGBD a fim de realizar um comparativo entre SGBD de diferentes classes, a recente abordagem NoSQL por exemplo, analisando seu comportamento sob um ambiente empresarial e também qual impacto a possível ausência de SQL pode trazer ao desempenho, visto que SGBD NoSQL como o MongoDB e o Neo4J utilizam uma linguagem orientada a documentos e a grafos, respectivamente. Também como trabalho futuro, será realizada a aplicação dos resultados do estudo em empresas que não possuam uma modelagem fortemente denormalizada e tampouco utilizem SGBD colunares, a fim de averiguar os resultados encontrados pelo *benchmark*.

Referências

[1] R. Wrembel and C. Koncilia, *Data warehouses and OLAP: concepts, architectures, and solutions*. Igi Global, 2007.

[2] E. Codd, S. Codd, and C. Salley, "Providing olap (on-line analytical processing) to user-analysis: An it mandate," *White paper*, 1993.

[3] R. Kimball and M. Ross, *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*, 2nd ed. Wiley Computer Publishing, 2002.

[4] "Tpc homepage," Transaction Processing Performance Council, 2017, acessado em: 12 de agosto de 2017. [Online]. Available: http://www.tpc.org

[5] "Tpc-h homepage," Transaction Processing Performance Council, 2017, acessado em: 19 de maio de 2017. [Online]. Available: http://www.tpc.org/tpch/

[6] M. P. Bax and R. Souza, "Modelagem estratégica de dados: Normalização versus "dimensionalização"," *KMBRASIL, Anais... São Paulo: SBGC*, 2003.

[7] W. H. Inmon, *Building the data warehouse*. John wiley & sons, 2005.

[8] S. Chaudhuri and U. Dayal, "An overview of data warehousing and olap technology," *ACM Sigmod record*, vol. 26, no. 1, pp. 65–74, 1997.

[9] A. Zelen, "Oltp vs. olap — what's the difference?" 2017, acessado em: 12 de abril de 2018. [Online]. Available: https://academy.vertabelo.com/blog/oltp-vs-olap-whats-difference/

[10] A. Sen and A. P. Sinha, "A comparison of data warehousing methodologies," *Communications of the ACM*, vol. 48, no. 3, pp. 79–84, 2005.

[11] R. Elmasri and S. B. Navathe, *Sistemas de Banco de Dados*, 6th ed. Pearson Education do Brasil, 2011.

[12] R. W. Brito, "Bancos de dados nosql x sgbds relacionais: análise comparativa," *Faculdade Farias Brito e Universidade de Fortaleza*, 2010.

[13] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[14] N. Leavitt, "Will nosql databases live up to their promise?" *Computer*, vol. 43, no. 2, 2010.

[15] G. Matei and R. C. Bank, "Column-oriented databases, an alternative for analytical environment," *Database Systems Journal*, vol. 1, no. 2, pp. 3–16, 2010.

[16] D. J. Abadi, "Query execution in column-oriented database systems," Ph.D. dissertation, Massachusetts Institute of Technology, 2008.

[17] S. Khoshafian, G. Copeland, T. Jagodits, H. Boral, and P. Valduriez, "A query processing strategy for the decomposed storage model," in *Data Engineering, 1987 IEEE Third International Conference on*. IEEE, 1987, pp. 636–643.

[18] D. Abadi, S. Madden, and M. Ferreira, "Integrating compression and execution in column-oriented database systems," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, 2006, pp. 671–682.

[19] P. Kyrö, "Revising the concept and forms of benchmarking," *Benchmarking: An International Journal*, vol. 10, no. 3, pp. 210–225, 2003.

[20] K. S. Bhutta and F. Huq, "Benchmarking–best practices: an integrated approach," *Benchmarking: An International Journal*, vol. 6, no. 3, pp. 254–268, 1999.

[21] *TPC BENCHMARK H Standard Specification. Revision 2.17.2*, Transaction Processing Performance Council, San Francisco, 2017, acessado em: 12 de maio de 2017. [Online]. Available: https://goo.gl/uaRRcv