Modelo de Banco de Dados Colunar: Características, Aplicações e Exemplos de Sistemas

Bruno Eduardo Soares, Clodis Boscarioli

Centro de Ciências Exatas e Tecnológicas – Universidade Estadual do Oeste do Paraná (UNIOESTE)
Av. Universitária, 2069 – Bairro Faculdade – 85819-110 – Cascavel – PR – Brasil
besoares90@gmail.com, clodis.boscarioli@unioeste.br

Abstract. *This paper describes some aspects of the NoSQL databases, in particular the columnar approach, presenting its main characteristics and advantages in relation to the linear storage method, analyzing its architecture and concepts applied on this storage model. The main advantages of columnar models are in compression, materialization and it helps the analysis of iteration blocks. This model is a tendency in different applications that require high availability.*

Resumo. *Este artigo discute alguns aspectos dos bancos de dados no modelo NoSQL, em específico na abordagem colunar, apresentando suas principais características e vantagens em relação ao método de armazenamento em linhas, analisando sua arquitetura e conceitos aplicados neste modelo de armazenamento. As principais vantagens do modelo colunar estão na compressão, na materialização, o que ajuda na análise de blocos de iteração. Este modelo é uma tendência em diferentes domínios de aplicação que exijam alta disponibilidade.*

1. Introdução

O armazenamento de informações sempre foi um fator fundamental ao se trabalhar com dados de forma digital. Os SGBDs relacionais vêm sendo amplamente utilizados desde sua concepção por serem de fácil manipulação e possuírem recursos que garantem a integridade dos dados, a exemplo das restrições de chave. No entanto, haja vista o crescimento e diversidade de informações geradas a todo tempo, o modelo de armazenamento em linhas nem sempre é o mais recomendado, principalmente, quando grandes quantidades de dados devem ser tratadas.

Os bancos de dados NoSQL podem ser vistos como alternativa para trabalhar com esses casos, por possuírem arquitetura diferenciada e seguirem conceitos diferentes, de forma a facilitar o tratamento com alta escalabilidade de dados. Grandes empresas, por exemplo, a Google, o Facebook e o Twitter adotaram a utilização de bancos de dados NoSQL, mais especificamente de modelo colunar, devido a grande demanda por consultas, vinculado ao elevado número de informações que manipulam.

Esse artigo traz uma discussão das principais características do modelo colunar e suas diferenças em relação ao modo de armazenamento do modelo relacional, apresentando sistemas que implementam esse modelo, e está estruturado da seguinte

forma: A Seção 2 introduz o conceito NoSQL, apresentando definições, características e modelos de bancos de dados que nele se enquadram. A Seção 3 descreve o modelo colunar de banco de dados, bem como sua arquitetura e as vantagens que podem ser obtidas sobre o modelo relacional. Por fim, a Seção 4 traz considerações e perspectivas do modelo colunar de banco de dados.

2. NoSQL

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.

A utilização de bancos de dados NoSQL se dá, principalmente, devido ao aumento gradativo de informações a serem armazenadas, no qual o desempenho é prejudicado e o tempo de resposta se torna 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).

NoSQL não é apenas mais um modelo de banco de dados, mas sim, um termo que define uma classe de modelos, sendo, conforme mostra literatura, os mais comuns e aplicados [Hecht e Jablonski, 2011]:

- 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). São exemplos de SGBD que suportam esse modelo: RIAK [Basho, 2012] e MemcacheDB [MemcacheDB, 2009].
- Orientado a colunas: Também chamado de modelo colunar, utiliza-se de tabelas para representação de entidades, e 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].
- Orientado a documentos: Similar ao modelo orientado a chaves, podendo gerar uma chave secundária para indexar seu valor. Exemplos de SGBD que suportam esse modelo o MongoDB [MongoDB, 2012] e o CouchDB [Apache, 2012b].
- Baseados em grafos: Os dados são armazenados em nós de um grafo cujas arestas representam o tipo de associação entre esses nós. São exemplos dessa abordagem o Ne04j [Ne04j, 2012], o GraphDB [GraphDB, 2012] e o InfoGrid [InfoGrid, 2012].

Uma das características chave de NoSQL é a habilidade de realizar escalonamento horizontal (particionamento do banco de dados) [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 da metodologia de armazenamento em linhas, 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.

Além de desempenho, há que considerar o Modelo CAP (*Consistency*, *Avaliability*, *Partition Tolerance* – Consistência, Disponibilidade e Tolerância à Partição) descrito em Gilbert e Lynch (2002), que versa que apenas dois dos três itens abaixo podem ser satisfeitos concorrentemente em um modelo de banco de dados:

- Consistência: percepção de 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 Modelo 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 (Atomicidade, Consistência, Isolamento e Durabilidade). No entanto, Pritchett (2008) questiona que se este garante consistência para bancos de dados particionados, então a disponibilidade é deixada em segundo plano (pelo Modelo CAP). Em contraposição a isso, o autor propôs o Modelo 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, ou seja, não ocorrerá no mesmo instante, gerando uma "fila" de consistência de posterior execução.

A disponibilidade do Modelo 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 Pritchett (2008), o Modelo 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. 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 escolha entre ACID ou BASE dependerá do tipo de aplicação com que se irá trabalhar. 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 é o caso do Cassandra [Apache, 2012a].

3. O Modelo Colunar de Banco de Dados

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 (Figura 1(b)), de forma densa e comprimida. Essa forma de armazenamento pode beneficiar a leitura dos dados, porém, comprometendo a escrita em disco.

Orientado a Linhas
Joao | 2432.00 | 1988 | Rio de Janeiro
Maria | 2511.00 | 1986 | São Paulo
Pedro | 3500.00 | 1976 | Mato Grosso
Jorge | 4200.00 | 1930 | Paraná

(a)

Orientado a Colunas
Joao | Maria | Pedro | Jorge
2432.00 | 2511.00 | 3500.00 | 4200.00
1988 | 1986 | 1976 | 1930
Rio de Janeiro | São Paulo | Mato Grosso | Paraná

(b)

Figura 1. Forma de armazenamento em (a) linhas e (b) colunas

As principais vantagens que se pode obter com a arquitetura de armazenamento dos bancos de dados de modelo colunar em relação aos de modelo relacional estão vinculadas à compressão, materialização e bloco de iteração, abaixo descritos.

3.1. 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, em particular 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.

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 SGBDs, devido às suas diferenças de arquiteturas. 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 um fator fundamental ao trabalhar com uma grande quantidade de dados devido à redução de espaço em disco e melhor desempenho. Vários SGBDs colunares aplicam métodos de compressão apresentando bons resultados em redução de espaço em disco. Esse fator se tornou um dos quesitos principais na escolha de um SGBD, principalmente para grandes empresas. Os SGBDs de modelo colunar podem comprimir informações com uma proporção maior que os de modelos relacionais, tornando-se esta uma importante vantagem sobre SGBDs desse modelo.

3.2. Materialização

Da mesma forma 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. Essa é a 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 a adotada no modelo colunar de banco de dados.

O modelo colunar de dados se beneficia na utilização da segunda abordagem, enquanto no relacional se obtém melhor desempenho com a primeira. Utilizando a *Late Materialization*, o modelo colunar é apto a reconstruir, em tempo hábil, um número

menor de *tuplas* para retorno do que na outra abordagem, pois os predicados da consulta são verificados antes do retorno da coluna, para que linhas desnecessárias não sejam analisadas.

Considere o exemplo onde se têm a tabela *Aluno* (Figura 2) e deseja-se recuperar o CPF e o RG dos alunos que possuem o número da matrícula maior que 2 e o nome começando com a letra 'A'. As Figuras 3 e 4 exemplificam os métodos de materialização *EM* e *LM*, respectivamente.

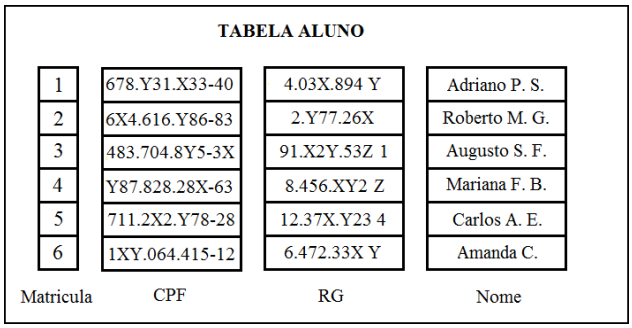

Figura 2. Exemplo de Instâncias de uma Tabela Aluno

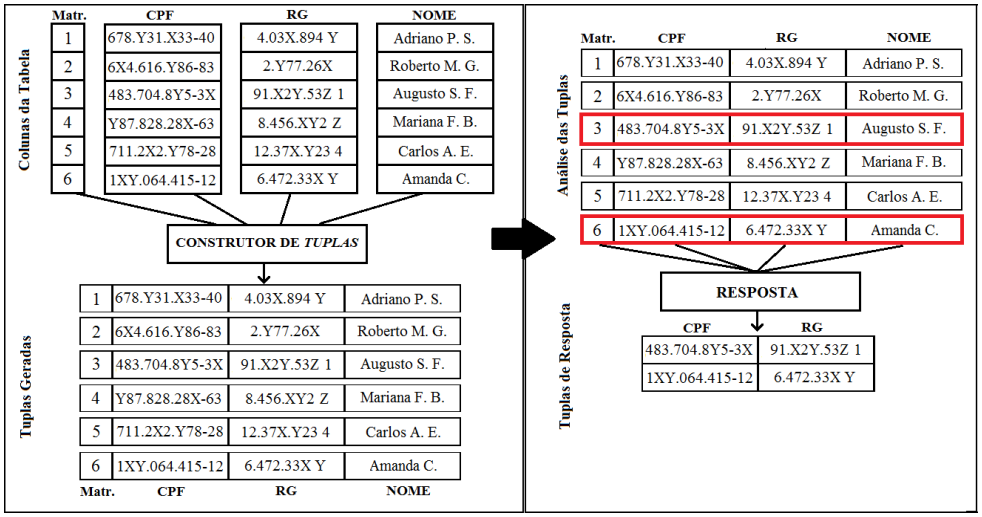

Figura 3. Tuplas geradas sobre o esquema da Figura 2 utilizando *EM*

Pela Figura 3 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 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 desnecessárias à consulta (que não atenderam ao predicado).

Na Figura 4 é mostrado que isso não ocorre utilizando a materialização *LM*. Primeiramente, as colunas referentes ao predicado são analisadas, verificando então quais são as posições que o atendem. A partir das posições obtidas é que as *tuplas* são construídas, não necessitando criar *tuplas* desnecessárias.

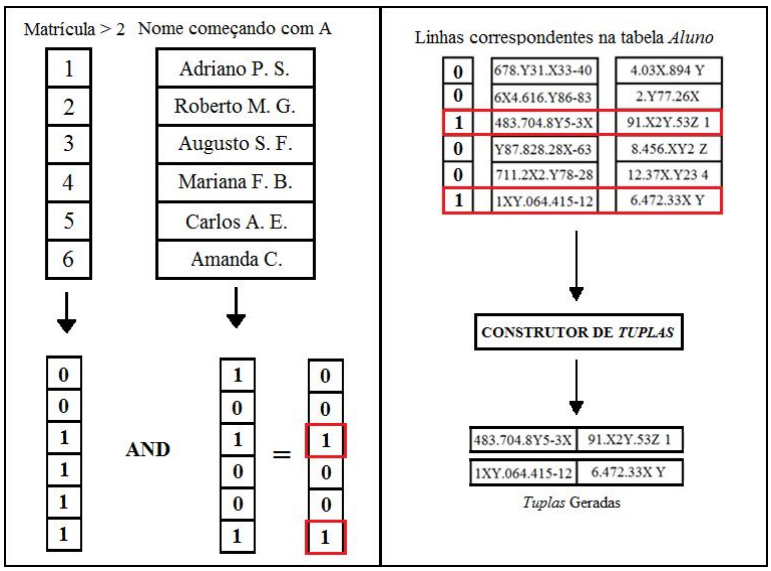

Figura 4. Tuplas geradas sobre o esquema da Figura 2 utilizando *LM*

3.3. Iteração de Bloco

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

Os SGBDs de modelo colunar possuem uma estrutura que minimiza este problema. Dado 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*.

A Tabela 1 traz os principais SGBDs que adotam o modelo colunar. Não há um padrão com relação à linguagem de consultas, interfaces, sistemas operacionais suportados, e outras características técnicas, além da dificuldade em minerar tais características técnicas, que diferem muito em formato e dimensão, nas documentações disponíveis sites dos fabricantes. Por exemplo, MonetDB descreve como características principais ACID, particionamento vertical, permite execução paralelizada, segue o padrão SQL 2003 e possui índice Hash. O Infobright garante ACID e alta escalabilidade, segue o ANSI SQL-92 com algumas extensões do SQL-99. O Cassandra tem controle de concorrência multiversão, opção de escolha entre consistência e disponibilidade e possui uma linguagem de consulta própria, a CQL (*Cassandra Query Language*). Vectorwise suporta ACID, processamento massivo paralelo e alta disponibilidade, com linguagem SQL (não informa o padrão). BigTable faz uso do sistema de arquivo distribuído *Google File System* para garantir confiabilidade e disponibilidade. HBase assegura consistência, persistência e é tolerante à partição. InfiniDB garante ACID, controle de concorrência multiversão e massivo processamento paralelo e SQL - não

informado o padrão. C-Store, tem alta disponibilidade é tolerante a Partição, usa SQL (não diz o padrão) e implementa Bit map como estrutura de índice. Vertica assegura alta disponibilidade e massivo processamento paralelo e SQL - não informa o padrão. Sybase IQ, tem as mesmas características do Vertica, além de estrutura de índice em bits.

Tabela 1. Exemplos de SGBD no modelo colunar

SGBD/Licença	Interfaces	Site do Projeto
MonetDB *Open Source*	SQL, JDBC, ODBC, PHP, Python, RoR, C/C++ e Pearl	http://www.monetdb.org
Infobright *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
Cassandra *Open Source*	Java, CQL, JDBC, DB-API 2.0, CLI, (Python), PDO (PHP) e DBI-compatible (Ruby)	http://cassandra.apache.org/
Vectorwise *Enterprise*	SQL, JDBC, ODBC e .NET	http://www.actian.com/products/vectorwise
BigTable Proprietária	C++, Python, Java	Não disponível
HBase *Open Source*	SQL, JDBC, possíveis por meio da API HBql disponível em: http://hbql.com	http://hbase.apache.org/
Metakit *Open Source*	C++, Mk4py (Python) e Mk4tcl (Tcl)	http://equi4.com/metakit/
InfiniDB *Community* e *Enterprise*.	JDBC, ODBC, ADO.NET, C/C++, PHP, Pearl, Python e Ruby	http://infinidb.org
C-Store *Open Source*	C/C++, SQL	http://db.csail.mit.edu/projects/cstore/
Widebase *Open Source*	C++, Erland, Go, Haskell, Java, PHP, Python, Ruby e Scala	http://widebase.github.com/
Vertica *Enterprise*	JDBC, ODBC e ADO.Net	http://www.vertica.com/

4. Conclusões

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.

Para a recuperação de dados, pode-se dizer que o modelo orientado por linhas é mais eficiente quando muitas colunas de uma tupla são requisitadas e quando a tupla é pequena, tal que o conteúdo possa ser todo recuperado em uma única operação de leitura de disco. O modelo orientado a colunas é mais eficiente quando apenas algumas

colunas de cada tupla precisam ser computadas e também quando se deseja substituir apenas instâncias de algumas colunas em todas as tuplas.

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 orientados a linhas 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, pois possuem imensa quantidade de informações e que são requeridas a todo tempo e para estes, o modelo colunar pode ser de grande valia.

A partir dessas considerações, e, considerando que diferentemente do modelo em linhas, os sistemas não seguem uma padronização base no modelo colunar, torna-se interessante avaliar em que pontos os bancos de dados de modelo colunar são melhor aplicáveis que os de modelo orientado a linhas. Diversos SGBD de modelo colunar possuem suporte a SQL (*Structured Query Language*) e gerenciam informações de forma muito semelhante a outros SGBD de modelo orientado a linhas, podendo assim, haver uma comparação adequada entre os modelos.

Referências

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

ABADI, D. J., BONCZ, P. A., HARIZOPOULOS, S. Column-Oriented Database Systems. In: *Proceedings of the VLB Endowment.* 2009, v. 2, n. 2, p.1664-1665.

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*. New York, NY, USA: ACM, 2008, p. 967-980.

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*. Istanbul, 2007, p. 466-475.

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

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

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

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

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.

GRAPHDB. http://www.sones.com. Consultado na Internet em: 20/09/2012.

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.

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, 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*. Seoul, Korea: VLDB Endowment, 2006, p. 487-498.

HECHT, R., JABLONSKI, S. NoSQL Evaluation: A Use Case Oriented Survey. In: *Proceedings of the 2011 International Conference on Cloud and Service Computing*, Hong Kong, China, 2011, p. 336-341.

INFOGRID. http://infogrid.org/blog/category/nosql/. Consultado na Internet em: 20/09/2012.

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

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

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

MONGODB. http://www.mongodb.org/. Consultado na Internet em: 10/03/2012.

NEO-LJ. http://neo-lj.org. Consultado na Internet em: 05/01/2013.

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

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

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.

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.

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

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

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: IEEE Computer Society, 2006, p. 59.