Uma Camada para o Mapeamento de Instruções SQL DML para o Banco de Dados NoSQL Chave-Valor Voldemort

Alternative Title: A Layer for the Mapping Management of SQL DML Instructions to the Key-Value NoSQL Database Voldemort

Augusto Verzbickas da Costa
Departamento de Informática e Estatística, UFSC
Florianópolis – SC, Brasil
augustoverzbickas@gmail.com

Patricia Vilain
Departamento de Informática e Estatística, UFSC
Florianópolis – SC, Brasil
patricia.vilain@ufsc.br

Ronaldo dos Santos Mello
Departamento de Informática e Estatística, UFSC
Florianópolis – SC, Brasil
r.mello@ufsc.br

RESUMO

Sistemas de informação que utilizam bancos de dados relacionais para a manutenção dos seus dados estão cada vez mais migrando para bancos de dados na nuvem a fim de minimizar custos com o gerenciamento de grandes volumes de dados. Os bancos de dados NoSQL são boas opções para esta finalidade, em particular, os bancos de dados chave-valor, que apresentam um modelo de dados simples e produtos escaláveis. Mesmo assim, os custos envolvidos com a migração para um gerenciador de dados NoSQL podem ser altos, principalmente em termos de alteração do código do sistema para adequação de interfaces de acesso. Este trabalho apresenta VoldemortSQL, uma camada de mapeamento de instruções SQL DML para o sistema gerenciador de banco de dados NoSQL chave-valor Voldemort, visando evitar o esforço de migração de dados e de interfaces de acesso de um banco de dados relacional para o banco de dados Voldemort. O uso desta camada permite, ao sistema de informação, a manipulação de dados na nuvem através da continuidade da utilização do padrão de acesso SQL. Experimentos demonstraram que o *overhead* de acesso introduzido com o VoldemortSQL não é proibitivo.

Palavras-Chave

NoSQL, modelo chave-valor, Project Voldemort, mapeamento de dados.

ABSTRACT

Information systems that uses relational databases for data maintenance are increasingly migrating to cloud databases in order to minimize costs related to the management of huge data volumes. In this context, NoSQL databases are good options, in particular, the key-value databases, which present a simple data model and scalable solutions. Even that, the costs regarding the migration to a NoSQL data management may be high, mainly in terms of source code updating of the information system for adapting database access interfaces. This paper presents VoldemortSQL, a layer that maps SQL DML instructions to the access methods of the NoSQL key-value database Voldemort with the purpose of avoiding the laborious work with data and access interface migrations from a relational database to Voldemort. The usage of this layer allows the manipulation of data in the cloud through the continuous usage of the SQL standard. An experimental evaluation had demonstrated that the overhead introduced with the layer is not prohibitive.

SBSI 2016, May 17–20, 2016, Florianópolis, Santa Catarina, Brazil.

Categories and Subject Descriptors

H.2 [**Database Management**]

General Terms

Management, Performance, Experimentation, Languages.

Keywords

NoSQL, key-value data model, Project Voldemort, data mapping.

1. INTRODUÇÃO

Uma grande variedade e volume de dados é produzida hoje por diversos dispositivos, pessoas e sistemas, muitas vezes em um curto intervalo de tempo, como por exemplo, dados gerados em redes sociais ou por sensores de propósitos diversos. A problemática do tratamento desta grande quantidade de dados é caracterizada na literatura como *Big Data* e motiva o desenvolvimento de sistemas de gerenciamento de banco de dados (SGBDs) diferentes dos tradicionais SGBDs relacionais [1], que requerem dados com estrutura rígida e um controle igualmente rígido da consistência desses dados. O paradigma de computação na nuvem é adequado ao gerenciamento de Big Data, pois fornece escalabilidade, disponibilidade e elasticidade de acesso. Dentre as soluções para gerenciamento de Big Data, as mais relevantes atualmente são os BDs NoSQL.

Diversos SGBDs NoSQL comerciais estão à disposição, sendo categorizados pelos seus modelos de dados, que são mais flexíveis em temos de representação de dados e, portanto, mais adequados à Big Data: *colunar*, *documento*, *chave-valor* e *grafo* [2]. Apesar dessa heterogeneidade de representação, todos têm em comum o fato de não seguirem o modelo relacional nem o padrão de acesso

SQL. Suas interfaces de acesso são, em geral, baseadas em primitivas simples de acesso, o que garante uma recuperação e armazenamento mais eficiente dos dados.

Sistemas de informação baseados em BDs relacionais podem ter interesse em migrar para um serviço de gerenciamento de dados na nuvem, em particular, para um SGBD NoSQL. Os motivos deste interesse são os mais diversos, dentre eles uma redução de custos com a administração de dados, bem como uma maior escalabilidade, elasticidade e disponibilidade do BD. Entretanto, não é simples alterar as interfaces de acesso a dados desses sistemas, baseadas no padrão SQL, uma vez que as primitivas de acesso NoSQL são completamente diferentes e o custo para tal alteração pode ser proibitivo.

Com a intenção de suprir esta demanda, este trabalho apresenta uma camada de mapeamento de dados relacionais para dados representados no modelo NoSQL chave-valor, bem como o mapeamento das mais importantes instruções de manipulação de dados SQL (SQL DML) para primitivas de acesso de um BD NoSQL chave-valor. A motivação para escolha de um BD chave-valor está na falta de trabalhos relacionados ao mapeamento relacional-chave-valor, assim como pela simplicidade e escalabilidade desta categoria de BD NoSQL. Esta solução é denominada *VoldemortSQL*, visto que o SGBD NoSQL acessado por ela é o *Voldemort*. A escolha pelo SGBD Voldemort foi motivada pela grande utilização do mesmo no mercado NoSQL, pela facilidade de configuração e uso e também por ser um produto *open source*. Uma avaliação experimental da utilização da camada *VoldemortSQL* demonstrou que o *overhead* introduzido por ela para algumas operações SQL DML não foi elevado, sendo a proposta promissora para uso por sistemas de informação baseados em BDs relacionais.

Este artigo apresenta mais seis seções, além das referências bibliográficas. A seção 2 apresenta a fundamentação teórica sobre BDs NoSQL, o modelo chave-valor e uma visão geral do BD Voldemort. A seção 3 detalha a camada *VoldemortSQL*, que suporta a manipulação de dados relacionais no BD chave-valor Voldemort. A seção 4 apresenta um conjunto de experimentos realizado para avaliar a camada e discute os seus resultados. A seção 5 é dedicada aos trabalhos relacionados e a seção 6 é dedicada à conclusão.

2. NoSQL E O BD VOLDEMORT

O movimento NoSQL (*Not Only SQL*) surgiu, no contexto da computação em nuvem, em 2009 para denotar BDs com ênfase em escalabilidade e disponibilidade em contraponto à ênfase em consistência forte adotada pelos BDs relacionais tradicionais [2]. Esta nova categoria de BDs não adota o modelo relacional de dados e também não adota, necessariamente, a linguagem SQL para definição e acesso a dados. A disponibilidade de SGBDs NoSQL pagos ou gratuitos aumenta a cada dia em número e utilização devido a sua melhor capacidade de gerir *Big Data* [3].

Ao relaxar a consistência, o gerenciamento de dados provido pelos BDs NoSQL é menos complexo que nos BDs relacionais, uma vez que BDs NoSQL não suportam consultas com junções, consultas complexas e integridade referencial, para não incorrer em uma provável perda de desempenho [3]. A ênfase em escalabilidade e disponibilidade dos BDs NoSQL está também associada a modelos de dados mais simples em termos de representação e manipulação, modelos esses baseados em uma chave associada ao dado que é a principal forma de acesso a ele. Os modelos de dados NoSQL são o modelo *chave-valor*, *colunar*, *documento* e *grafo*, sendo o primeiro deles o foco deste artigo.

O modelo chave-valor é o mais simples dentre os modelos de dados NoSQL, sendo semelhante a um mapa ou dicionário, onde cada item de dado ou registro, seja ele simples ou complexo, é associado a uma chave que é a única forma de acesso a ele. Não existe a noção de referência ou chave estrangeira entre dados, sendo o controle de relacionamentos provido pela aplicação [4]. Em suma, o conteúdo do dado é visto como uma "caixa preta" pelo BD chave-valor, devendo este conteúdo ser tratado pela aplicação.

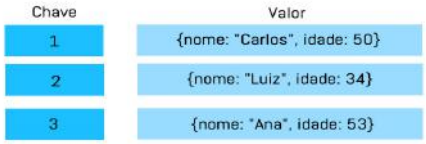

Figura 1. Estrutura de armazenamento do modelo chave-valor.

A Figura 1 mostra a estrutura de um BD NoSQL baseado no modelo chave-valor, que é bastante simples, ou seja, é composta pelos dois conceitos: a identificação do dado (*chave*) e o seu conteúdo (*valor*), definindo um conjunto de pares chave-valor.

As principais primitivas para manipulação de dados em um BD NoSQL chave-valor são GET, PUT e DELETE. A API que define essas primitivas segue a arquitetura REST [5]. Os comandos GET e DELETE possuem como parâmetro apenas a chave do registro a ser acessado ou removido, respectivamente. No caso da primitiva PUT, são informados a chave e o valor a serem inseridos no BD.

O *Voldemort* é um BD NoSQL que utiliza o modelo chave-valor para armazenamento de dados [6]. Ele é o BD alvo para fins de armazenamento de dados relacionais a serem manipulados através da camada de acesso proposta neste artigo. O Voldemort é considerado um BD de alto desempenho, visto que consegue executar uma grande quantidade de instruções de leitura e de escrita em um curto período de tempo. Entretanto, o Voldemort, assim como outros BDs chave-valor, não possui linguagem de consulta, ou seja, as instruções de acesso são aquelas oferecidas pela API REST. A Figura 2 mostra um exemplo de trecho de código que faz acesso ao Voldemort. Este exemplo ilustra as primitivas de acesso explicadas anteriormente.

```
value = store.get(key)
store.put(key, value)
store.delete(key)
```

Figura 2. Exemplos de primitivas suportadas pelo Voldemort.

A chave do objeto é uma *string*, sem limite de caracteres, e o seu valor pode ser configurado como *string*, um documento do tipo JSON [7] ou outros tipos de dados que não são detalhados neste trabalho [6].

O Voldemort também define esquemas para organizar suas informações chamados *stores*. Um BD pode ter quantos *stores* forem necessários, bastando especificá-los na configuração do Voldemort. Cada *store* possui um nome e os conjuntos de chave e valor que só existem dentro do *store* associado.

3. VOLDEMORTSQL

O *VoldemortSQL* é uma camada de mapeamento de instruções SQL DML para o BD chave-valor Voldemort. A finalidade da camada é prover o acesso transparente a dados relacionais mantidos na nuvem por sistemas de informação baseados em BDs relacionais. Desta forma, esses sistemas não necessitam alterar suas interfaces de acesso relacionais, continuando a manipular dados na nuvem através do padrão SQL.

A camada suporta uma versão simplificada das principais instruções SQL DML: INSERT, UPDATE, DELETE e SELECT. Essas instruções são convertidas em uma ou mais primitivas de acesso GET, PUT e DELETE do Voldemort, conforme o caso.

O *VoldemortSQL* encontra-se implementado na linguagem de programação Java e utiliza a plataforma *Heroku* [8], que disponibiliza uma forma simples de criar servidores na nuvem através da definição de serviços Web que permitem manipular os dados e receber resultados de consultas. Esta plataforma facilita o trabalho do desenvolvedor, pois oferece *pacotes* de ambientes com servidores configurados e prontos para serem utilizados. Para o controle das dependências do projeto é utilizado o *Apache Maven* [9], uma ferramenta de automação de compilação utilizada para desenvolvimento Java. Ele pode ser utilizado também para construir ou gerenciar objetos escritos em outras linguagens e empacotamento de arquivos.

3.1 Mapeamento entre os Modelos Relacional e Chave-Valor

O modelo relacional é mais rico em termos de conceitos (tabelas, atributos, colunas, dentre outros) que o modelo chave-valor, que possui apenas esses dois conceitos. Desta forma, os conceitos do modelo relacional tiveram que ser adaptados para um desses dois conceitos. A Tabela 1 mostra a estratégia de mapeamento adotada pelo *VoldemortSQL*.

Tabela 1. Mapeamento relacional-chave-valor

Relacional	Chave-Valor
BD	Componente da Chave
Tabela	Componente da Chave
Tupla	Valor
Atributo	Componente do Valor
Chave Primária	Componente da Chave
Chave Estrangeira	Componente do Valor

Conforme ilustra a Tabela 1, informações a respeito do BD, tabela e chave primária compõem a chave de acesso a tupla relacional mapeada para o BD chave-valor. Já a tupla em si (conjunto de pares atributo-valor) define um valor no BD chave-valor. Como BDs chave-valor não possuem controle de integridade referencial, chaves estrangeiras são tratadas da mesma forma que outros atributos da tabela, ou seja, constituem parte do valor da tupla.

Considere, por exemplo, um BD relacional denominado EMPRESA, que possui uma tabela FUNCIONARIO e uma tupla com os seguintes atributos: id (chave primária), nome, salário e departamento (esse último uma chave estrangeira). Considere ainda que os valores dos atributos são, respectivamente, 31, "João da Silva", 3000.00 e "DRH". Desta forma, a chave de acesso desta tupla no BD chave-valor é "EMPRESA.FUNCIONARIO.31" e o valor associado a esta chave é {nome:"João da Silva", salário:3000.00, departamento:"DRH"}.

Como não existe linguagem de manipulação de dados similar à SQL em BDs NoSQL chave-valor, um esquema de metadados foi definido para permitir o mapeamento de uma consulta SQL. Esse esquema é mantido no Voldemort e exemplificado na parte superior da Figura 3. Cada BD relacional é representado por uma *store* no Voldemort e mantém 2 conjuntos de pares chave-valor, denominados *Tabela de Metadados* e *Tabela de Registros*. A primeira delas é a que define o esquema de metadados.

O exemplo da Figura 3 mostra os metadados da tabela "FUNCIONARIO" pertencente ao BD relacional exemplo "EMPRESA". Neste esquema, cada chave é o nome da tabela e seu valor é o conjunto de metadados necessário para prover o mapeamento. O valor é um conteúdo complexo que indica qual atributo é a chave primária da tabela (*pks*) e a lista de chaves de acesso no BD chave-valor para todas as tuplas desta tabela (ids). Esta chave de acesso, ao ser concatenada com o nome do BD relacional, permite o acesso às tuplas da tabela no Voldemort.

A Tabela de Registros é a que mantém os dados propriamente ditos das tabelas relacionais no Voldemort, conforme já explicado anteriormente. O nome do BD relacional é omitido na chave, pois ele já identifica a *store* na qual o conjunto chave-valor se encontra, sendo o par *(nome da tabela, valor da chave primária)* suficiente para identificar e acessar, em uma *store* específica, uma tupla relacional.

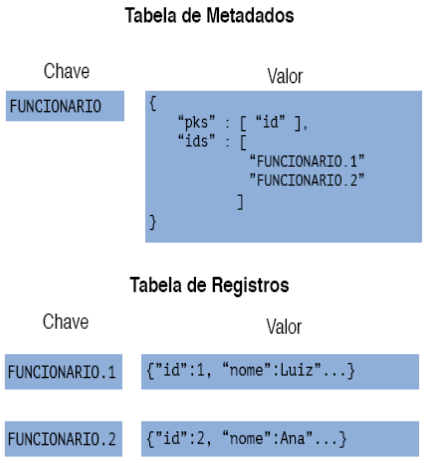

Figura 3. Exemplo do esquema de metadados de mapeamento utilizado pelo VoldemortSQL

3.2 Arquitetura

A arquitetura da camada *VoldemortSQL* é composta de três módulos: *interface de acesso*, *tradutor de instruções* e o *processador de instruções*. Cada um destes módulos é responsável por uma tarefa no processo de manipulação e acesso de/a dados. A interface de acesso serve como o receptor das instruções SQL DML consideradas, recebendo-as e as enviando para o tradutor. Uma vez analisadas e traduzidas para primitivas da API REST, elas são encaminhados ao processador de comandos, que acessa o BD Voldemort para executá-las. A Figura 4 mostra o funcionamento da arquitetura do *VoldemortSQL*, desde a submissão da instrução SQL até o retorno de uma mensagem ou dos dados ao usuário ou sistema.

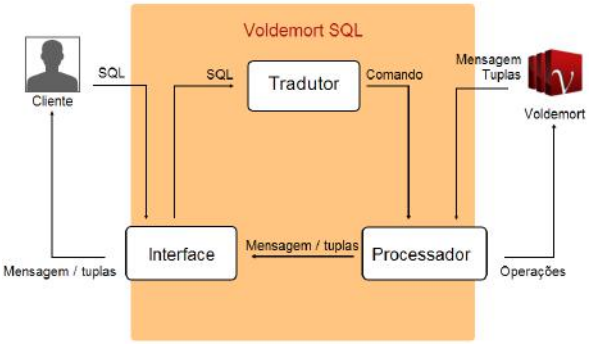

Figura 4. Arquitetura da camada VoldemortSQL.

Em suma, o *software* cliente que acessa o *VoldemortSQL*, seja pelo serviço Web ou pela biblioteca, envia uma instrução SQL ao *VoldemortSQL* e recebe tuplas, caso essa instrução seja um SELECT, ou uma mensagem de sucesso ou falha, caso esta instrução seja um INSERT, DELETE ou UPDATE. Mais detalhes sobre essas formas de acesso são fornecidas a seguir.

3.2.1 Módulo Interface

O VoldemortSQL pode ser acessado de duas formas: através da sua API, que aceita requisições REST, ou utilizando a sua biblioteca Java. Quando a biblioteca Java for adicionada ao projeto de implementação do sistema como uma dependência, ela requer que a biblioteca Java de acesso ao *VoldemortSQL* também seja adicionada. Para fins de simplificação, tanto a API quanto a biblioteca possuem os mesmos métodos de análise de instruções SELECT, INSERT, UPDATE e DELETE, que estão sumarizados na Figura 5 e explicados a seguir.

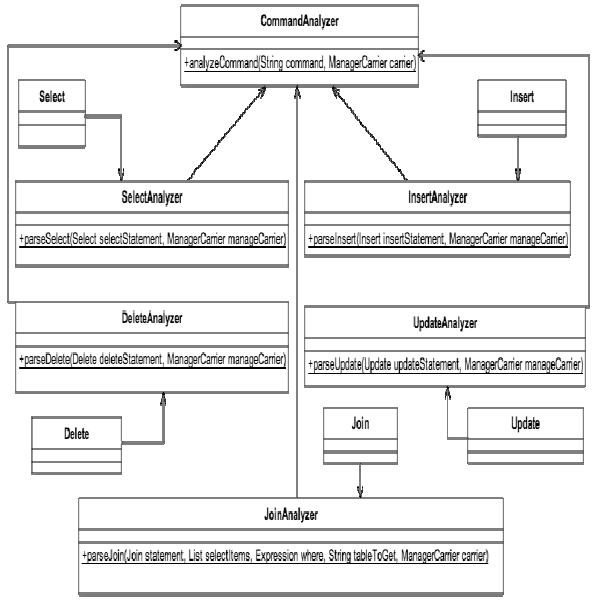

Figura 5. Diagrama com as principais classes de análise de instruções SQL da camada VoldemortSQL.

Para analisar uma instrução SQL através da biblioteca utiliza-se a classe *CommandAnalyzer* e o método estático *analyzeSqlCommand*. Este método recebe como parâmetro uma *string* contendo a instrução SQL que deve ser analisada e um objeto do tipo *ManagerCarrier*. Este objeto contém o cliente de acesso ao Voldemort e, para criá-lo, basta utilizar um construtor que recebe três parâmetros: o endereço do seu BD no Voldemort, o nome do *store* que armazena os metadados da aplicação e o nome do *store* que possui os dados do sistema de informação.

Se o *VoldemortSQL* estiver hospedado em um servidor *Heroku*, o acesso a ele pode ser feito através de uma chamada REST ao seu serviço Web. Para isso, realiza-se uma chamada do tipo GET passando, como único parâmetro, a instrução SQL, já que os *stores* a serem acessados no BD Voldemort devem ser configurados previamente.

3.2.2 Módulo Tradutor

O modulo Tradutor utiliza a biblioteca *JSQLParser* para a decomposição de instruções SQL em objetos Java. Esta é uma biblioteca Java que foi adicionada como depedência ao projeto do VoldemortSQL e serve apenas para identificar o tipo de instrução SQL. Esses objetos facilitam a manipulação dos dados e permitem maior controle sobre possíveis exceções que os comandos digitados pelo usuário possam gerar. Quando o objeto referente a uma instrução SQL é identificado, na classe *CommandAnalyzer* (Figura 5), ele é designado ao seu analisador correspondente para que seja decomposto e tenha sua execução realizada.

Ao final da análise de instruções SQL, cada um dos analisadores retorna o resultado da sua operação respectiva ao *CommandAnalyzer*. Cada instrução é traduzida em uma série de primitivas da API REST, conforme mostra a Tabela 2.

Tabela 2. Tradução de instruções SQL para as primitivas de acesso da API REST do Voldemort

Instrução SQL	Primitiva da API REST
SELECT	GET no esquema de metadados GET no BD chave-valor
INSERT	PUT no esquema de metadados PUT no BD chave-valor
UPDATE	GET no esquema de metadados GET no BD chave-valor PUT no BD chave-valor
DELETE	GET no esquema de metadados PUT no esquema de metadados DELETE no BD chave-valor

O processamento dessas instruções traduzidas é realizado pelo módulo Processador, detalhado a seguir.

3.2.3 Módulo Processador

Quando uma instrução SQL é traduzida, o modulo Processador executa as operações necessárias no BD Voldemort, conforme descrito na Tabela 2. Essas execuções são detalhadas a seguir.

Uma instrução INSERT gera a inclusão de dados no Voldemort através da execução da primitiva PUT. Os metadados também são atualizados através da adição da chave primária da nova tupla no esquema de metadados (ver seção 3.1). Caso a tabela ainda não exista no esquema de metadados do *VoldemortSQL*, um novo registro chave-valor é criado na *Tabela de Metadados*.

O processamento de uma instrução SELECT inicia com um comando GET no esquema de metadados para obter o identificador de todos os registros da respectiva tabela no BD

Voldemort. Depois disso, os identificadores obtidos são utilizados para buscar os registros referentes a eles. Caso a instrução SELECT possua uma cláusula WHERE (filtro sobre os dados), os registros obtidos são analisados para verificar se atendem à condição da cláusula.

Cabe salientar que a camada *VoldemortSQL* trata junções do tipo INNER JOIN em instruções SELECT. Essa é uma capacidade de consulta inexistente em BDs NoSQL, sendo uma contribuição deste trabalho. O processamento de um INNER JOIN segue o processamento clássico de junções em BDs relacionais: obtém-se, para cada tupla da tabela de menor tamanho, as tuplas correspondentes, definidas pela condição de junção, na outra tabela. Quando se encontra todas as tuplas e suas correspondentes, coloca-se os seus atributos em um mesmo objeto, que ainda passa pelo filtro da cláusula WHERE, se existir, para verificar se todas as condições são atendidas. Em caso positivo, gera-se, por fim, a tupla resultante do casamento das duas tuplas das tabelas envolvidas na junção. Este não é ainda o algoritmo mais otimizado possível de junção, sendo esta otimização alvo de trabalhos futuros.

O processamento da instrução UPDATE primeiramente seleciona os dados que serão modificados, da mesma forma que no processamento de uma instrução SELECT. Após a busca dos registros no Voldemort, os dados são modificados e comandos iPUT são executados para armazenar os dados atualizados.

Por fim, o processamento do comando DELETE é realizado diretamente nos registros do Voldemort que devem ser removidos através da sua primitiva própria DELETE. Porém, primeiramente é executado um comando GET no esquema de metadados para resgatar os dados que devem ser excluídos. Quando os dados são excluídos do BD Voldemort, também são removidas as suas referências na *Tabela de Metadados*, conforme descrito na seção 3.1.

4. AVALIAÇÃO EXPERIMENTAL

Uma avaliação experimental preliminar, composta por um conjunto de testes para avaliar o desempenho do *VoldemortSQL*, foi realizada. A execução dos testes considerou um BD relacional exemplo fornecido pelo SGBD MySQL no seu próprio *Web site* [10]. Este BD possui dados referentes a países (tabela *Country*), suas principais cidades (tabela *City*) e suas línguas oficiais (tabela *CountryLanguage*). O esquema lógico relacional deste BD, denominado *World*, é mostrado na Figura 7.

O BD *World* original possui cerca de 5.000 tuplas. Entretanto, para realizar experimentos com volumes maiores de dados, o BD foi aumentado com dados artificiais. O primeiro teste foi realizado com o tamanho original do BD, o segundo teste foi realizado com 10 vezes mais tuplas que o tamanho original e o terceiro teste foi realizado com 100 vezes mais tuplas que o tamanho original. As tuplas foram geradas de maneira aleatória em cada uma das três tabelas, preservando-se a integridade referencial e a integridade das chaves primárias.

A fim de medir apenas o desempenho do Voldemort e da camada proposta, os testes realizados não executaram no servidor do *Heroku*, mas sim na própria máquina utilizada para o desenvolvimento. Desta forma, evitou-se computar o tempo gasto para transportar os dados de instruções SQL até o servidor, assim como o tempo gasto para transportar as tuplas resultantes até o cliente. O valor desses tempos depende de muitos fatores externos, como a largura de banda de Internet, distância até o servidor onde o Project Voldemort está instalado, configuração do servidor, entre outros, o que poderia distorcer muito os tempos finais coletados.

O computador onde os testes foram realizados apresenta a seguinte configuração: Macbook Pro – Mid 2012 com sistema operacional OS X Yosemite 10.10.3, processador 2.9 Ghz Intel Core i7 e memória de 8GB 1600 Mhz DDR3.

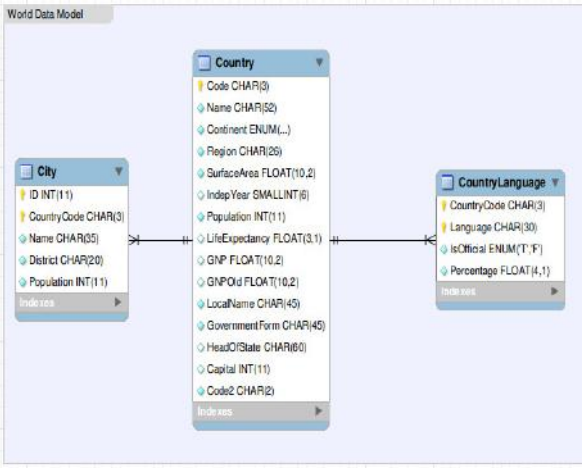

Figura 7. Esquema do BD relacional *World*.

Os primeiros dois tipos de testes tiveram como objetivo verificar o desempenho de operações de carga e de acesso utilizando a camada *VoldemortSQL*. O primeiro deles contabilizou os tempos gastos com a execução de instruções INSERT e o segundo os tempos gastos com instruções SELECT. Esses testes são detalhados a seguir.

4.1 Testes de Carga de Dados

O primeiro conjunto de testes avaliou o desempenho de instruções INSERT no BD relacional e seu mapeamento e armazenamento no Voldemort. As instruções de inserção foram realizadas nas três tabelas do BD *World* e os tempos de inserção para os três tamanhos considerados do BD são mostrados na Tabela 3. Os valores na Tabela 3 representam a media de três execuções da mesma instrução sobre o BD vazio para aumentar a confiabilidde do experimento. Esse procedimento foi realizado para todos os testes realizados neste trabalho.

Uma análise destes resultados mostra que, conforme o tamanho do BD vai aumentando, o tempo gasto para a inserção vai, obviamente, aumentando também. Entretanto, este aumento vai sendo proporcionalmente cada vez menor à medida que o BD vai crescendo, conforme mostra a Figura 8. Isso é positivo, visto que mostra a boa escalabilidade do *VoldemortSQL*, mesmo para uma operação cara de BD, como é a carga de dados. Na Figura 8, o eixo X representa os três tamanhos considerados do BD e o eixo Y representa a quantidade de registros inserida por minuto em cada tabela. O tempo gasto com a inserção de tuplas nas tabelas *City* e *CountryLanguage* foi maior devido a verificações de integridade referencial realizadas pelo *VoldemortSQL*.

4.2 Testes de Acesso a Dados de uma Tabela

Na sequência, um segundo conjunto de testes avaliou o desempenho de instruções SELECT sobre os três tamanhos

considerados do BD *World* já populados. O primeiro teste avaliou uma busca simples pelo ID de uma tupla em cada uma das três tabelas. O tamanho do BD foi indiferente neste experimento, já que o código do *VoldemortSQL* foi desenvolvido de forma que, quando as condições da cláusula WHERE são apenas chaves primárias, busca-se os dados no Voldemort de forma direta, sem ler o registro para verificar o ID, visto que o ID já se encontra na chave do registro. Desta forma, são apresentados, na Tabela 4, os tempos obtidos para o BD com tamanho 10 vezes maior do que o original.

Tabela 3. Tempo médio de execução das instruções INSERT para os três tamanhos considerados do BD *World*

BD com Tamanho Original		
Tabela	**Tempo (ms)**	**Tuplas Inseridas**
City	12032	4079
Country	2350	239
CoutryLanguage	2601	983
BD com Tamanho 10x Maior		
City	107712	40790
Country	21402	2390
CoutryLanguage	23829	9830
BD com Tamanho 100x Maior		
City	1063487	407900
Country	203111	23900
CoutryLanguage	235732	98300

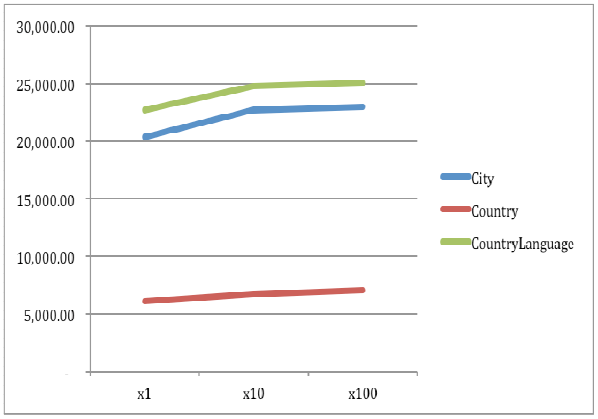

Figura 8. Tempo médio gasto com a inserção de tuplas por minuto para os três tamanhos considerados do BD *World*.

Tabela 4. Tempo médio de execução das instruções SELECT para buscas diretas pela chave de acesso

Tabela	Tempo (ms)
City	463
Country	452
CountryLanguage	481

Em suma, o tempo de busca, quando a condição envolve apenas chaves primárias, é bastante satisfatório, visto que este é o principal tipo de busca no modelo chave-valor, justificando o bom desempenho.

Outro teste considerou buscas por atributos que não fazem parte da chave primária sobre a tabela de maior tamanho (*City*). Ele foi executado para os três tamanhos do BD e as instruções SELECT definidas tiveram variações no número de filtros. As instruções SELECT são apresentadas na Tabela 5 e os resultados são apresentados na Tabela 6. Cabe ressaltar que os tempos de execução de cada consulta não são diretamente proporcionais ao número de filtros, sendo mais dependentes do número de tuplas retornadas por cada uma delas.

Tabela 5. Instruções SELECT consideradas para buscas por atributos não-chave

ID	Consulta
C1	SELECT * FROM CITY WHERE CITY.POPULATION > 500000;
C2	SELECT * FROM CITY WHERE CITY.POPULATION > 500000 AND CITY.COUNTRYCODE = 'BRA';
C3	SELECT * FROM CITY WHERE CITY.POPULATION > 500000 AND CITY.COUNTRYCODE = 'BRA' AND CITY.DISTRICT = 'RIO DE JANEIRO';

Tabela 6. Tempo médio de execução das instruções SELECT da Tabela 5 para os três tamanhos considerados do BD *World*

BD com Tamanho Original	
Consulta	**Tempo (ms)**
C1	481
C2	474
C3	463
BD com Tamanho 10x Maior	
C1	4423
C2	4072
C3	4231
BD com Tamanho 100x Maior	
C1	42329
C2	38221
C3	40388

Uma análise dos resultados demonstra que, conforme o tamanho do BD vai crescendo, o tempo gasto para trazer os dados também aumenta. Porém, o crescimento do tempo vai se desacelerando em relação ao crescimento dos registros, o que é positivo. Pode-se observar esta relação de crescimento entre registros e o tempo gasto no gráfico da Figura 9, para a consulta C3. Nesta figura, o eixo X representa os três tamanhos de BD considerados e o eixo Y representa a quantidade de registros por segundo analisados pelo *VoldemortSQL*.

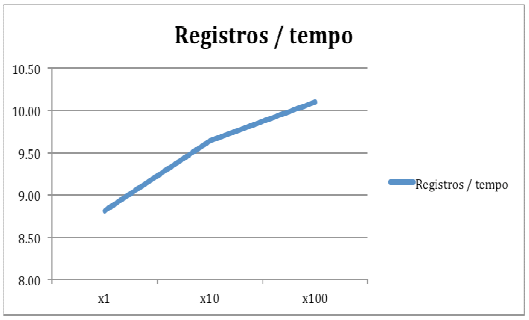

Figura 9. Tempo médio gasto com a análise de registros por segundo na consulta C3 para os três tamanhos considerados do BD *World*.

4.3 Testes de Acesso a Dados de mais de uma Tabela

Outro teste de acesso a dados avaliou o desempenho do *VoldemortSQL* ao realizar uma seleção envolvendo mais de uma tabela através do uso de junções na instrução SELECT. A Tabela 7 mostra os resultados obtidos para cada uma das instruções de seleção realizadas envolvendo conjuntos de tabelas sendo unificadas através de cláusulas INNER JOIN. Nestas consultas, os únicos filtros definidos foram as condições de junção.

Tabela 7. Tempo médio de execução das instruções SELECT com junções para o maior tamanho considerado do BD *World*

Tabelas Unificadas	Tempo (ms)
City e Country	55990
Country e CountryLanguage	15582
City, Country e CountryLanguage	198222

Este teste foi executado sobre o maior tamanho considerado do BD (530.100 tuplas). Como esperado, a operação de INNER JOIN no *VoldemortSQL* foi custosa, visto que o BD Voldemort não possui suporte para junções e, portanto, foi necessário acessar e cruzar todos os registros para verificar se as condições de junção eram satisfeitas. Entretanto, considerou-se que os resultados obtidos não foram proibitivos, uma vez que a consulta mais complexa, envolvendo as três tabelas e executando duas multiplicações de duas tabelas (407.900 x 23.900 e 98.300 x 23.900) consumiu em torno de 3 minutos.

4.4 Testes de *Overhead* com a Utilização da Camada

A avaliação experimental considerou também testes relacionados ao *overhead* introduzido com a utilização da camada *VoldemortSQL*. Esses testes foram realizados coletando o tempo total de execução de uma instrução SQL com o uso da camada e subtraindo o tempo de execução das primitivas de acesso no BD Voldemort. Com isso, obteve-se o valor de *overhead*.

A Tabela 8 apresenta os resultados obtidos com a execução de instruções INSERT e de alguns tipos de consultas, para alguns tamanhos do BD *World*. Em alguns casos, o tempo é medido em milissegundos e, em outros, em segundos. A coluna *Tempo VoldemortSQL* apresenta o tempo total de execução e a coluna *Tempo Voldemort* apresenta apenas o tempo de execução gasto pelo BD chave-valor.

Conforme ilustra a Tabela 8, obeserva-se que o percentual de *overhead* para as consultas por chave primária foi ligeiramente superior à 100%. Mesmo assim, considerou-se o resultado satisfatório, pois todos os tempos de *overhead* foram inferiores a um segundo. O mesmo ocorre para consultas que retornam todos os atributos das três tabelas.

O tempo total de processamento foi bem maior para as operações de carga no BD com maior tamanho, como já era esperado, por esta ser uma operação cara em geral, apresentando um percentual de *overhead* semelhante ao desempenho dos dois tipos de consulta avaliados anteriormente. Isso é justificado pela inserção de informações no esquema de metadados, além da inserção dos dados propriemente dita no BD chave-valor. Por outro lado, percebe-se que, na maioria dos casos, quase 50% do tempo total é gasto na gravação dos dados no BD Voldemort, evidenciando que um grande gargalo é o armazenamento no BD chave-valor e não necessariamente o processamento realizado pela camada.

Com relação ao desempenho das consultas com junções, já era esperado um *overhead* alto, uma vez que BDs chave-valor não suportam junções e, portanto, essa carga de processamento ficaria a cargo da camada. O tempo total de processamento foi bastante superior ao tempo das outras consultas, porém, verificou-se que o percentual de *overhead* foi similar ao processamento das demais instruções, o que é positivo. Para tentar diminuir esses tempos totais de processamento, pretende-se implementar, na próxima versão da camada, um algoritmo mais otimizado de junção.

Tabela 8. *Overhead* de processamento com a utilização da camada VoldemortSQL para a execução de algumas instruções SQL DML

Tabela(s)	Tempo VoldemortSQL	Tempo Voldemort	*Overhead*
Carga de Dados no BD com Tamanho 100x Maior (s)			
City	107.7	42.3	65.4
Country	21.4	9.8	11.6
Country Language	23.8	11.1	12.7
Consultas com Acesso Direto pela Chave Primária no BD com Tamanho 100x Maior (ms)			
City	475	179	296
Country	448	175	273
Country Language	415	178	237
Consultas com Retorno de Todos os Atributos das Tabelas no BD com Tamanho Original (ms)			
City	1874	932	942
Country	78	34	44
Country Language	317	147	170
Consultas com Junções no BD com Tamanho 100x Maior (s)			
City e Country	56	26.2	29.8
Country e Country Language	15.5	6.8	8.7
City, Country e Country Language	198.2	87.3	110.9

Uma análise geral dos resultados dos testes de *overhead* evidencia que, naturalmente, a camada proposta gera um aumento no tempo de processamento devido às tarefas adicionais de mapeamento relacional-chave-valor de dados e de instruções SQL. Entretanto, considera-se que este custo não é proibitivo principalmente para consultas de acesso direto e consultas que não envolvem junções, que ocorrem com bastante frequência em um BD. O *overhead*, em ambos os casos, foi inferior à um segundo.

5. TRABALHOS RELACIONADOS

Alguns trabalhos na literatura lidam com o problema do mapeamento de BDs relacionais para BDs na nuvem. O *SimpleSQL*, por exemplo, é uma proposta muito semelhente a essa, porém, o foco é outro modelo de dados NoSQL, ou seja, BDs de documentos, no caso, o SimpleDB, um BD mantido pela Amazon [11]. Mais recentemente, uma outra abordagem propõe a criação de um *middleware* que suporta o mapeamento de comandos SQL para alguns SGBDs NoSQL [12].

Propostas relacionadas, mas que caminham na direção de SGBDs relacionais projetados para a nuvem, são o *Relational Cloud* [13] e o *ConPaaS* [14]. Entretanto, são sistemas que incorrem, atualmente, em altos custos de aquisição e implantação.

Apesar dos trabalhos supracitados estarem também relacionados à problemática de gerenciamento de dados relacionais na nuvem, não foi encontrado outro trabalho que trate o mapeamento de esquemas e de instruções SQL DML para BDs NoSQL chave-valor. A busca por trabalhos relacionados foi realizada nos principais portais de pesquisa acadêmica (Google Scholar[1], DBLP[2], ACM DL[3] e IEEE Xplore[4]) e considerou uma janela de tempo de 10 anos até janeiro de 2016.

6. CONCLUSÃO

Este trabalho apresenta e avalia a *VoldemortSQL*, uma camada de *software* responsável pelo mapeamento de dados relacionais e instruções SQL DML para dados e operações correspondentes em um BD NoSQL chave-valor. A principal contribuição com esta camada é garantir transparência de acesso a sistemas de informação que manipulam dados relacionais e desejam manter esses dados em ambientes na nuvem mais adequados à gerência de *Big Data*. O foco deste trabalho foi o mapeamento para o BD NoSQL Voldemort, um BD de código aberto e de amplo uso no marcado. Mesmo assim, é possível estender o uso desta camada para outros BDs NoSQL chave-valor, uma vez que suas primitivas de acesso são idênticas ou bastante similares.

A utilização de uma camada de *software* por sistemas de informação baseados em BDs relacionais incorre em custos menores, em termos de modificação do sistema, se comparado com a troca de um SGBD relacional por um SGBD totalmente novo NoSQL. Neste segundo caso, seria necessário trocar toda a interface de acesso SQL pela nova interface de acesso do BD NoSQL, o que poderia ser proibitivo. Com a solução proposta neste trabalho, a interface SQL permanence inalterada.

Uma avaliação experimental preliminar serviu para validar o funcionamento do VoldemortSQL e medir o seu desempenho. Apesar das instruções SQL testadas incorrerem em um *overhead* (esperado) de processamento quando a camada é utilizada, avalia-se que o desempenho da camada não torna o seu uso proibitivo, conforme discutido na seção 4. Entende-se que este é um custo a ser pago para que sejam disponibilizadas capacidades de gerenciamento de dados relacionais em um BD na nuvem e, inclusive, oferecer novas funcionailidades de acesso, como consultas com junções e consultas com filtros sobre qualquer atributo de uma tabela, além da chave. Essas novas capacidades sem dúvida agregam valor em termos de acesso a dados em BDs chave-valor, cujas formas de acesso são bastante limitadas, se resumindo a buscas pela chave dos registros.

Outro fator positivo a salientar na avaliação experimental foi a escalabilidade da *VoldemortSQL*. Pode-se observar que, conforme o BD vai crescendo, o tempo gasto para executar consultas não aumenta proporcionalmente. Na verdade, este tempo vai diminuido. Ainda, quando uma consulta utiliza apenas filtros pela chave primária da tabela, o tempo de consulta permanece praticamente o mesmo, independentemente do tamanho do BD relacional.

Conforme já salientado, um trabalho futuro imediato é a melhoria do desempenho das consultas com junções através da busca de algoritmos de processamento de junções mais otimizados, como o *hash join* [15]. Outros possíveis trabalhos futuros são o suporte a instruções SQL DDL e o aprimoramento das capacidades das instruções SQL DML, experimentos envolvendo volumes maiores de dados, bem como o projeto e desenvolvimento de um modulo de otimização de consultas baseado em índices para acelerar buscas sobre os dados mantidos no BD chave-valor. Experimentos envolvendo a comparação de desempenho de acesso a dados em um BD relacional e no Voldemort, através do uso do VoldemortSQL, também estão previstos.

[1] https://scholar.google.com.br/

[2] http://dblp.uni-trier.de/

[3] http://dl.acm.org/

[4] http://ieeexplore.ieee.org/Xplore/

7. REFERÊNCIAS

[1] Tiwari, S. Professional NoSQL. Wrox, 2011.

[2] Sadalage, P. J. and Fowler, M. NoSQL Distilled. Addison-Wesley Professional, 2012.

[3] Planet Cassandra. What is NoSQL. Disponível em: http://www.planetcassandra.org/what-is-nosql/. Último acesso em 28/02/2015.

[4] Hecht, R. and Jablonski, S. NoSQL Evaluation: A Use Case-Oriented Survey. In: International Conference on Cloud and Service Computing (CSC), 2011. p.336-341.

[5] Fielding, R. T. Architectural styles and the design of network-based software architectures. PhD Thesis, University of California, Irvine, 2000.

[6] Project Voldemort. Disponível em: http://www.project-voldemort.com/voldemort/. Último acesso em 28/02/2015.

[7] JSON. Disponível em: http://www.json.org. Último acesso em 28/02/2015.

[8] Heroku. Disponível em: http://www.heroku.com. Último acesso em 28/02/2015.

[9] Maven. Disponível em: http://www.maven.apache.org. Último acesso em 28/02/2015.

[10] MySQL World Database. Disponível em: https://dev.mysql.com/doc/world-setup/en/. Último acesso em 28/02/2015.

[11] Calil, A. and Mello, R. S. SimpleSQL: A Relational Layer for SimpleDB. In: East European Conference on Advances in Databases and Information Systems (ADBIS), Springer-Verlag, 2012.

[12] Rith, J.; Lehmayr P. S.; Meyer-Wegener, K. Speaking in Tongues: SQL access to NoSQL systems. In: ACM Symposium on Applied Computing (SAC), 2014. p.855-857.

[13] Curino, C. et al. Relational Cloud: a Database Service for the Cloud. In: Biennial Conference on Innovative Data Systems Research (CIDR), 2011. p.235-240.

[14] Pierre, G.; Stratan, C. ConPaaS: A Platform for Hosting Elastic Cloud Applications. IEEE Internet Computing, v.16, n.5, p.88-92, 2012.

[15] Mishra, P.; Eich, M. H. Join Processing in Relational Databases. ACM Computing Surveys, v. 24, n. 1, p.63–113, 1992.