UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

Implementação de comandos SQL/DDL no SimpleSQL

Aluno: Gabriel dos Santos Ferreira

Orientador: Ronaldo dos Santos Mello

Florianópolis, SC

Dezembro de 2013

SUMARIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SQL	*Structured Query Language*
DB	*Data Base*
DDL	*Data Definition Language*
DML	*Data Manipulation Language*
API	*Application Programming Interface*
REST	*Representational State Transfer*

RESUMO

O presente trabalho visa estender o SimpleSQL com suporte a comandos DDL da linguagem SQL. O SimpleSQL é uma camada de acesso relacional ao SimpleDB, um banco de dados NoSQL criado pela Amazon. O SimpleSQL dispõe de uma interface SQL simplificada, que abstrai do usuário qualquer conhecimento sobre os métodos de acesso ao SimpleDB, persistência de dados na nuvem e seu modelo de representação de dados. O SimpleSQL suporta atualmente os comandos básicos de manipulação de dados: SELECT, INSERT, DELETE e UPDATE. Neste trabalho são adicionados os comandos CREATE TABLE, DROP TABLE e ALTER TABLE e também é realizada a verificação dos comandos DML para fins de consistência das definições destes comandos frente ao esquema de definição de dados definido através desta extensão.

Palavras chave: Bancos de Dados nas Nuvens, SQL, NoSQL, SimpleDB, SimpleSQL.

ABSTRACT

This work aims to extend the SimpleSQL with support to SQL DDL commands. SimpleSQL is a layer of relational access to SimpleDB, a NoSQL database created by Amazon. SimpleSQL offers a simplified SQL interface that abstracts the user any knowledge about the SimpleDB access methods, data persistence in the cloud and its model of data representation. SimpleSQL currently supports the basic data manipulation commands: SELECT, INSERT, DELETE and UPDATE. This work added the commands CREATE TABLE, DROP TABLE e ALTER TABLE and a correctness checking of DML commands submitted to SimpleSQL against the schema defined through this extension.

1 INTRODUÇÃO

O conceito de software como serviço é um paradigma que cresceu muito e vem sendo cada vez mais utilizado nos últimos anos. Um dos principais motivos para tal crescimento é a redução de custos, uma vez que utilizando um software como serviço, paga-se de acordo como a demanda de uso, ou seja, paga-se somente pelos recursos utilizados ao invés de se gastar uma grande quantia para adquirir a licença e para cobrir os custos de manutenção (ARMBRUST et al., 2009). Softwares são oferecidos como serviço através da nuvem, conceito este que é detalhado em seções adiante.

Vistas as vantagens deste paradigma, vários tipos diferentes de softwares e plataformas passaram a ser oferecidas como serviço. As soluções para armazenamento e gerenciamento de dados seguiram esta tendência (SOUSA et al., 2010), começando assim a surgir inúmeros SGBDs oferecidos através da nuvem. Para se adaptar a esta realidade, estes sistemas começaram a ter sua arquitetura modificada, a fim de incorporar a eles características de sistemas distribuídos, como alta disponibilidade e tolerância a falhas, requisitos indispensáveis para softwares oferecidos como serviço. Ainda, por questões de eficiência e compatibilidade, os SGBDs oferecidos através da nuvem passaram a adotar modelos diferentes do tradicional modelo relacional (ABADI, 2009).

Sistemas que adotam esses novos modelos são conhecidos como *NoSQL* (FOWLER, 2012). Estes sistemas sacrificam consistência transacional forte e uma interface de alto nível para aumentar a escalabilidade e disponibilidade em sistemas largamente distribuídos, através de uma interface e de um modelo de dados mais

simples (CATTELL, 2010). Exemplos desses novos modelos são o chave-valor, baseado em coluna e de documento (SOUSA et al., 2010).

Como já mencionado, esses novos modelos de dados não seguem o modelo relacional. Dessa forma, não existe neles suporte para a linguagem SQL, que é utilizada pela grande maioria dos sistemas que usam bancos de dados atualmente. Devido a isso, a migração de um banco de dados para nuvem implicaria na reescrita do código das aplicações que manipulam dados nesse banco, pois as APIs para trabalhar com modelos NoSQL funcionam de maneira totalmente diferente daquelas que trabalham com modelo relacional.

Visando minimizar esta problemática, foi criado o SimpleSQL, uma camada relacional para acesso ao SimpleDB (CALIL; MELLO, 2012), um banco de dados NoSQL criado pela Amazon para armazenamento e gerenciamento de dados na nuvem. O SimpleSQL suporta uma versão restrita do SQL, isolando a aplicação cliente da interface de acesso ao SimpleDB, oferecendo desta forma uma interface relacional de alto nível para acesso a dados na nuvem.

Os comandos da linguagem SQL são divididos em dois grupos: o DML, que são comandos para manipulação de dados e o DDL, que são comandos para definição de dados. A *Data Definition Language (DDL)* é a linguagem adotada para a criação das estruturas de armazenamento de dados, sendo composta por comandos para a criação de entidades relacionais, como tabelas (relações), visões (*views*), regras de integridade referencial (*constraints*) e índices (JACOBS; CARVALHO, 2006). Essas estruturas são utilizadas para definir como os dados devem ser representados e são de fundamental importância para a organização e utilização eficiente do banco de dados. Devido a esta importância, este trabalho se propõe a adicionar o suporte a linguagem DDL ao SimpleSQL, a fim de oferecer ao usuário

final uma solução ainda mais próxima do modelo relacional. Esta era uma limitação a ser tratada no SimpleSQL, que suportava apenas o tratamento de comandos DML.

A adição da DDL oferece ainda grandes vantagens, tais como validação de dados e validação sintática de comandos SQL dentro da própria camada, tirando essas responsabilidades da aplicação cliente. O DDL também garante a consistência de integridade de chaves, uma vez que é oferecido suporte a definição de chaves primárias e estrangeiras.

1.1 Objetivo Geral

O objetivo deste trabalho é estender a camada relacional SimpleSQL, de forma que ela suporte comandos para definição de dados. Este trabalho adiciona suporte aos comandos CREATE TABLE, ALTER TABLE e DROP TABLE.

1.2 Objetivos Específicos

- Adicionar ao SimpleSQL suporte a definição de dados;
- Realizar testes que comprovem a viabilidade da utilização destes comandos na camada relacional;
- Verificar a corretude dos comandos DML submetidos ao SimpleSQL contra o esquema de dados gerado pelos comandos DDL definidos neste trabalho.

2 BANCO DE DADOS NAS NUVENS

As seções a seguir apresentam os conceitos de computação nas nuvens, bancos de dados NoSQL e, por fim, o SimpleDB, que é o banco de dados na nuvem sobre o qual a camada SimpleSQL foi desenvolvida.

2.1 Computação nas nuvens

O conceito de computação nas nuvens diz respeito à utilização da capacidade de processamento e armazenamento de grandes *data centers* por meio da Internet (AMOROSO, 2012). O armazenamento de dados é feito através de serviços que podem ser acessados em qualquer lugar do mundo, a qualquer hora, sem a necessidade de instalar programas gerenciadores de dados ou armazenar dados localmente, ou seja, todo acesso é feito remotamente.

As principais características da computação nas nuvens são a escalabilidade, o provisionamento dinâmico de recursos sobre demanda, cobrança baseada no uso do recurso ao invés de uma taxa fixa (*pay as you go*) e a distribuição geográfica dos recursos de forma transparente aos usuários.

Uma das características de grande parte das aplicações atualmente é a necessidade de lidar com um grande volume de dados e transações. Muitas também têm uma grande preocupação com questões de privacidade dos dados. Os provedores de serviços oferecem garantias de disponibilidade, privacidade e instalação, tornando assim a computação nas nuvens um grande atrativo que passou a ser amplamente utilizado nos dias de hoje.

Os três principais tipos de serviços oferecidos na nuvem são:

- *Software as a Service* (*SaaS*): softwares para finalidades especificas oferecidos ao usuário final, geralmente através de uma interface web;
- *Plataform as a Sevice* (*PaaS*): plataformas, como sistemas operacionais;
- *Infrastructure as a Service* (*IaaS*): recursos computacionais em geral, como capacidade de armazenamento ou poder de processamento.

2.2 Sistemas NoSQL

Sistemas gerenciadores de bancos de dados possuem alta complexidade de instalação e manutenção. Por isso, muitas vezes é mais vantajoso serem utilizados também como um serviço. Para muitas empresas o sistema de pagamento baseado no uso e o suporte a manutenção representam uma grande economia de capital e esforço.

Por outro lado, para que os sistemas de banco de dados sejam oferecidos como serviço, é necessário que sejam embutidos neles características de sistemas distribuídos, como tolerância a falhas, escalabilidade e transparência ao usuário. Quando oferecidos como serviço, SGBDs precisam estar prontos para esse novo mundo, onde normalmente se lida com um volume muito grande de dados.

Os novos SGBDs nas nuvens vêm sendo desenvolvidos com enfoque nessas necessidades. Grande parte das soluções que vem sendo criadas não segue o modelo relacional. Estes sistemas são conhecidos como NoSQL e possuem todas as características de sistemas distribuídos citadas anteriormente, abrindo mão da

forte consistência de dados dos sistemas relacionais. Existem vários modelos que são adotados por esses sistemas, como o modelo chave-valor ou de super-coluna.

Bancos de dados NoSQL são fortemente relacionados com conceitos de computação nas nuvens. Algumas características de destaque deles são o suporte a escalabilidade horizontal, ou seja, aumento da capacidade de processamento do sistema através da adição de novas máquinas a rede e a replicação e particionamento de dados entre diferentes servidores. Além disso, possuem também um forte relacionamento com os paradigmas atuais de desenvolvimento. A modelagem da solução segundo um paradigma orientado a objetos é mais facilmente persistida em um modelo de dados NoSQL e sua interface de acesso simplificada, geralmente desenvolvida sobre HTTP (CALIL; MELLO, 2012).

Enquanto os sistemas relacionais possuem como premissa as características ACID (Atomicity, Consistency, Isolation, Durability), os sistemas NoSQL seguem o acrônimo BASE (Basically Available, Soft state, Eventually consistent). BASE é uma filosofia para modelagem de sistemas de dados que propõe que os sistemas devem se preocupar mais com a disponibilidade ao invés da consistência das operações (PRITCHETT, 2008). Os sistemas NoSQL se baseiam no teorema de CAP, que diz que sistemas distribuídos devem escolher apenas duas características entre tolerância a falhas, alta disponibilidade e consistência (CATTELL, 2010). No caso dos SGBDs NoSQL, abre-se mão de uma consistência mais forte para se ter alta disponibilidade e tolerância a falhas.

2.3 SimpleDB

O SimpleDB é a solução da Amazon para armazenamento e gerenciamento de dados na nuvem. Ele segue o modelo orientado a documentos para representar e acessar os dados. O SimpleDB faz parte do Amazon Web Services.

O modelo do SimpleDB possui três conceitos: *domínios, itens e atributos*. Um domínio é um contexto de itens, sendo responsável por armazenar um conjunto de itens. Um item modela uma entidade do mundo real, sendo eles compostos por uma coleção de atributos. Os atributos descrevem as características desta entidade, sendo que cada atributo é um par chave-valor. A chave é o nome do atributo e identifica o que ele representa, dando sentido ao seu valor. A Figura 1 a seguir mostra um exemplo de modelagem no SimpleDB, descrevendo um domínio com 3 itens descritos na figura.

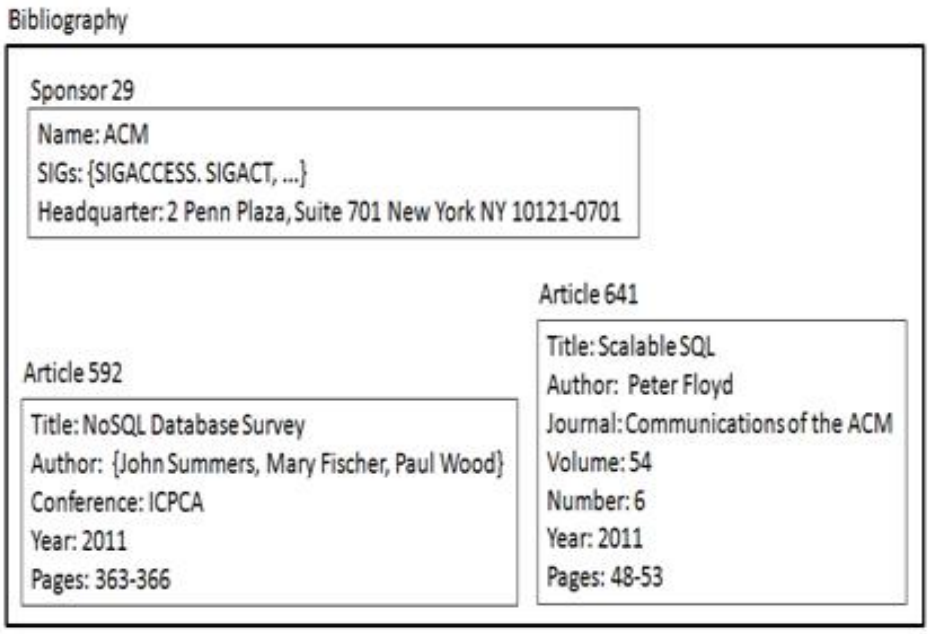

Figura 1: Exemplo de modelagem no SimpleDB.

O domínio é a entidade principal do modelo. Por questões de desempenho, é recomendado que haja uma distribuição de dados em diferentes domínios, mas nada impede que se utilize apenas um domínio para manter todos os dados. Um usuário pode possuir até 250 domínios, o que é suficiente para a maioria das aplicações.

No SimpleDB não existe o conceito de relação, e o sistema não suporta nativamente consultas envolvendo mais de um domínio. Esse tipo de consulta deve ser feito dentro da aplicação cliente, fazendo a junção dos resultados de múltiplas consultas. O SimpleDB também não apresenta o conceito de tipo de dados, tudo nele é representado como texto.

Como já mencionado, os domínios são compostos por uma coleção de itens. Cada item possui um nome, que atua como um identificador único. Os itens são compostos por uma coleção de atributos, que são pares chave-valor. Um atributo pode ter múltiplos valores para uma determinada chave.

A consistência no SimpleDB é implementada de forma que se garanta que qualquer operação de escrita vai atualizar todas as réplicas do item. Por outro lado, não se tem garantia de consistência na leitura, ou seja, uma leitura pode retornar o valor antigo do atributo. Como uma contramedida a esse problema, o SimpleDB oferece ao usuário a opção de leitura consistente, que aguarda o término de todas as operações de escrita para então retornar o valor da leitura, tendo um maior tempo de resposta que as operações de leitura convencionais.

O SimpleDB é acessado por meio de uma API REST. Sendo assim, todas as operações são realizadas através de requisições HTTP, por GET ou POST. A Figura 2 ilustra uma operação *PutAttributes*, que é utilizada para criação de itens e atualização de seus atributos, através do método POST.

```
POST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Host: sdb.amazonaws.com

Action=PutAttributes
&DomainName=MyDomain
&ItemName=Item123
&Attribute.1.Name=Color&Attribute.1.Value=Blue
&Attribute.2.Name=Size&Attribute.2.Value=Med
&Attribute.3.Name=Price&Attribute.3.Value=0014.99
&AWSAccessKeyId=your_access_key
&Version=2009-04-15
&Signature=valid_signature
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A01%3A28-07%3A00
```

Figura 2: Exemplo de comando para criação de item na API REST do SimpleDB através do método POST

No comando acima é criado um item chamado *Item123*, no domínio *MyDomain*, com os atributos *Color, Size e Price* que possuem os valores *Blue, Med e 0014.99*, respectivamente.

A Amazon disponibiliza o SimpleDB apenas para ser contratado como serviço, não sendo possível fazer instalação local ou utilizar em nuvens privadas.

3 SIMPLESQL

O SimpleDB é um dos principais sistemas de banco de dados na nuvens atualmente. Ele se destaca principalmente pela sua simplicidade de uso e configuração. Por outro lado, por ele ser um sistema NoSQL, torna-se um grande problema para muitas aplicações que ja utilizam bancos de dados relacionais utilizá-lo, pois migrar uma aplicação desenvolvida para manipular dados de acordo com o modelo relacional pode significar um esforço muito grande, podendo até inviabilizar seu uso.

Com o objetivo de sanar ou minimizar essa dificuldade, foi desenvolvido o SimpleSQL, que é uma camada lógica relacional sobre o SimpleDB. O SimpleSQL realiza a tradução dos comandos SQL para a API do SimpleDB, permitindo a interoperabilidade de manipulação de dados relacionais na nuvem (CALIL; MELLO, 2012). Essa camada recebe as requisições no formato SQL, processa os dados na nuvem e devolve as resposta no formato relacional, no caso de uma consulta.

A grande vantagem de se utilizar essa camada é que ela abstrai da aplicação cliente qualquer conhecimento sobre a interface de acesso do SimpleDB e seu modelo de dados, oferecendo ao usuário uma interface relacional de alto nível para acesso aos dados na nuvem.

O SimpleSQL foi desenvolvido utilizando a plataforma *Microsoft .NET Framework 3.5* e a linguagem de programação *C# 3.0*. Pela falta de um padrão de interface de acesso entre os sistemas NoSQL, o SimpleSQL, na sua versão atual, suporta unicamente o Amazon SimpleDB. Outra limitação do SimpleSQL é que ele manipula apenas dados que tenham sido inseridos através do próprio SimpleSQL, não tratando a manipulação de dados colocados diretamente da nuvem. Isso se

deve ao fato de que o conceito de tabela do modelo relacional não foi mapeado para o SimpleDB, pois não existe nenhum conceito similar a este no mesmo. Para tratar esse problema, em cada item criado pelo SimpleSQL é adicionado um atributo especial, chamado de *SimpleSQL_TableName,* que guarda o nome tabela a qual o item pertence, sendo o SimpleSQL capaz de manipular apenas itens que possuam esse atributo. A Figura 3 mostra o relacionamento entre conceitos do modelo relacional e do modelo do SimpleDB, utilizado para fazer a tradução entre os dois modelos.

Relational	SimpleDB
Schema	Domain
Table	-
Tuple	Item
Column	Attribute name
Value	Attribute value

Figura 3: Equivalência entre conceitos relacionais e o modelo do SimpleDB.

3.1 Arquitetura do SimpleSQL

A arquitetura do SimpleSQL pode ser dividida em 3 componentes principais: A interface de acesso, a decomposição dos comandos e o processamento e retorno de respostas.

Para que seja capaz de conectar-se ao SimpleDB e identificar os domínios, o SimpleSQL deve receber, em sua instanciação, os seguintes dados:

- *Access Key*: chave de acesso do usuário ao SimpleDB. Esta informação pode ser encontrada no portal da Amazon, após autenticação;
- *Secret Access Key*: chave de acesso secreta. Junto da *access key*, formam o par de autenticação e autorização do usuário. Também é encontrada no portal da Amazon;
- Distribuição de domínios: caso o usuário tenha mais de um domínio, ele deverá prover ao SimpleSQL um dicionário que tenha como chave o nome do domínio e como valor a lista de tabelas daquele domínio. Caso seja apenas um domínio, é informado apenas o seu nome, simplificando o processo.

É necessário também que o ambiente de execução do SimpleSQL tenha acesso ao *website* da Amazon.

A Figura 4 a seguir ilustra a arquitetura do SimpleSQL. Cada componente desta arquitetura é explicado nas próximas seções.

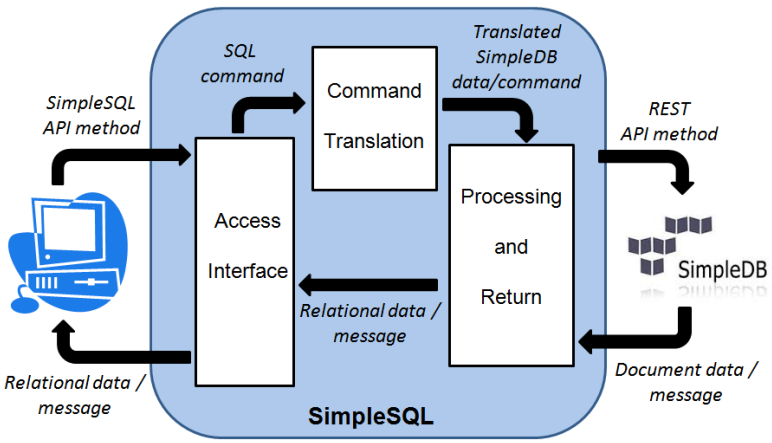

Figura 4: Arquitetura do SimpleSQL

3.2 Interface de Acesso

A interface de acesso é composta pelo conjunto de métodos que ficam visíveis para a aplicação cliente. A interface de acesso possui atualmente métodos para versões restritas dos comandos SQL/DML SELECT, UPDATE, INSERT e DELETE, que foram desenvolvidas em um trabalho de conclusão de curso anterior. Este trabalho estende esse componente com comandos SQL/DDL. Os comandos suportados e suas restrições são os seguintes:

CREATE TABLE: permite a criação de tabelas com atributos *not null* e *unique*. Permite ainda a definição de chaves primária e chaves estrangeiras. Em relação aos tipos de dados, são suportados os tipos INT, DOUBLE, CHAR e VARCHAR;

DROP TABLE: exclusão de tabelas com verificação de referências de chaves estrangeiras, ou seja, é verificado nas outras tabelas do domínio se existe alguma chave estrangeira fazendo referência a esta tabela, impedindo que ela seja removida caso a referência exista;

ALTER TABLE: suporte à adição de colunas (ADD COLUMN) e exclusão de colunas (DROP COLUMN), com verificação de referências para exclusão de colunas, ou seja, é verificado nas outras tabelas do domínio se existe alguma chave estrangeira fazendo referência a esta coluna, impedindo que ela seja removida caso a referência exista.

Para receber estes comandos, a interface de acesso do SimpleSQL possui dois métodos de entrada. O primeiro deles é o método *ExecuteQuery,* responsável pelo processamento de consultas. Este método recebe como entrada um comando do tipo SELECT, em forma de texto, e retorna os resultados dessa consulta em uma estrutura tabular. O outro método é o *ExecuteNonQuery,* responsável pelo

processamento dos demais comandos, incluindo os comandos DDL. Ele recebe como entrada um comando que não seja do tipo SELECT, também em forma de texto, e retorna uma *string*, mostrando alguma informação sobre a operação feita.

3.3 Decomposição de comandos

A decomposição dos comandos é feita através de expressões regulares, que são aplicadas sobre o texto do comando. Através da expressão é reconhecido qual comando vai ser processado e seus parâmetros. A partir da identificação do comando e seus parâmetros, é instanciado um objeto do tipo correspondente ao comando. A Figura 5 apresenta as expressões regulares utilizadas para decompor os comandos CREATE TABLE, DROP TABLE e ALTER TABLE.

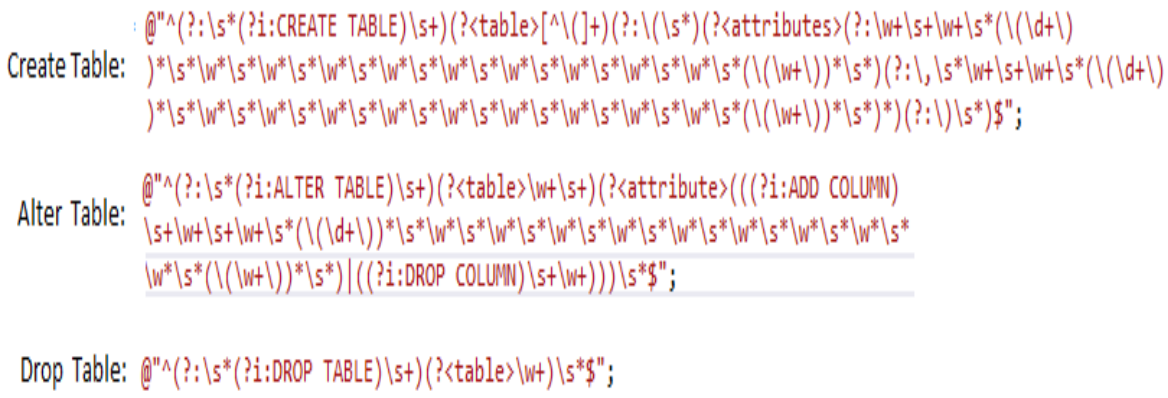

Figura 5: Expressões regulares utilizadas para decompor os comandos CREATE TABLE, DROP TABLE e ALTER TABLE.

O SimpleSQL possui uma classe genérica *Command* e todos os comandos implementados, incluindo os comandos DDL, herdam desta classe.

Depois de reconhecidos e separados seus parâmetros, os comandos estão prontos para ser processados e então submetidos ao SimpleDB. A Figura 6 a seguir ilustra o diagrama de classes do domínio do SimpleSQL.

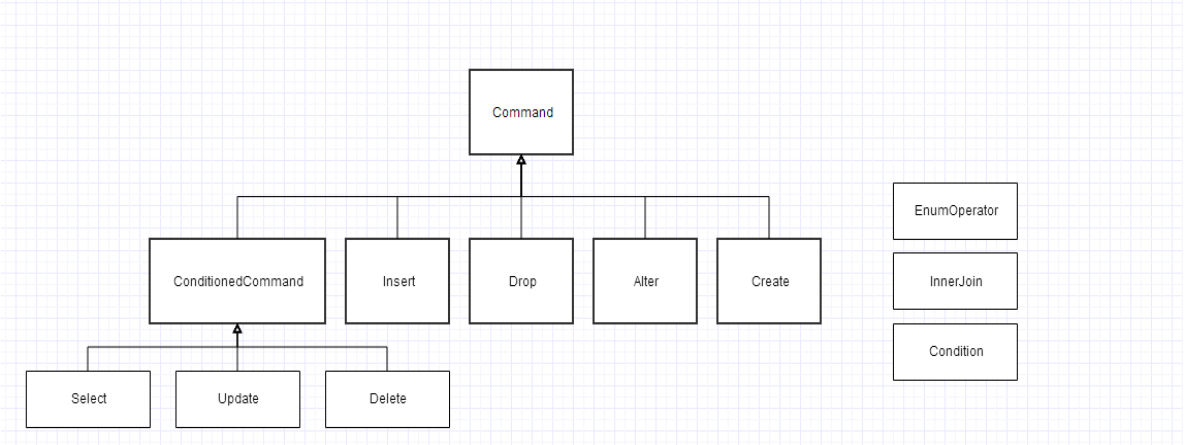

Figura 6: Diagrama de classes do domínio do SimpleSQL.

3.4 Processamento de comandos e retorno

Depois de instanciados os comandos, o SimpleSQL irá traduzí-los para a API do SimpleDB. Os comandos são então submetidos ao SimpleDB e as respostas transformadas para apresentação no formato relacional, caso o comando seja uma consulta.

Para cada comando SQL, são executados uma série de métodos da API do SimpleDB, até que todas as operações necessárias para aquele comando sejam concluídas.

3.4.1 CREATE TABLE

Quando o comando processado é um CREATE TABLE, o SimpleSQL primeiramente verifica se existe uma tabela com mesmo nome já criada. Para tanto, é realizada uma consulta sobre o domínio *Schema,* e em seguida é verificado em cada item retornado o nome dele. Se nenhum dos itens tiver o nome igual ao nome da tabela passada no comando, então a tabela não existe e pode ser criada.

Após essa verificação, o SimpleSQL cria um item, que representará a tabela criada. Os atributos deste item criado e como o SimpleSQL o utiliza são explicados na seção 3.5.

Depois de criado o item, é chamado o método *PutAttributes* da API do SimpleDB, que recebe como parâmetros o nome do domínio (*Schema*) e o item criado. O método então cria a requisição HTTP e submete ela ao SimpleDB, armazenando o item criado no domínio *Schema*.

3.4.2 ALTER TABLE

Para um comando ALTER TABLE, primeiramente se verifica se a tabela passada no comando realmente existe, realizando uma consulta no domínio *Schema* e verificando nos itens retornados se algum deles tem o mesmo nome da tabela que foi passada no comando. Em caso positivo, o SimpleSQL irá verificar qual tipo de operação a ser feita.

Caso a operação seja ADD COLUMN, o SimpleSQL obtêm o item do *Schema* que corresponde a tabela passada no comando, através do método *GetAttributes* da

API do SimpleDB, e verifica nesse item se existe um atributo de mesmo nome que o da coluna a ser alterada. Em caso positivo, um erro é retornado, se não um novo atributo é adicionado nesse item, chamando o método *PutAttributes* da API do SimpleDB, que recebe como parâmetros o nome do domínio (*Schema*), o nome do item a ser alterado (nome da tabela) e o novo atributo que será adicionado.

Depois, o SimpleSQL localiza no domínio real todos os itens que pertencem aquela tabela e adiciona este novo atributo a todos os itens localizados. Para cada item localizado, é feita uma chamada ao método *PutAttributes* da API do SimpleDB, recebendo como parâmetros o nome do domínio, o nome do item a ser alterado e o novo atributo que será adicionado. Para cada chamada, o método cria uma requisição HTTP e submete ela ao SimpleDB, armazenando o novo atributo em cada item passado como parâmetro.

A operação DROP COLUMN funciona da mesma forma, primeiramente verificando se o atributo a ser deletado existe no *Schema*, e depois deletando o atributo passado tanto no *Schema* quanto nos itens do domínio real. Para deletar os atributos, o SimpleSQL chama o método *DeleteAttributes* da API do SimpleDB, passando como parâmetros o nome do domínio, o nome dos itens e o nome do atributo a ser deletado em cada item. Para cada chamada, o método cria uma requisição HTTP e submete ela ao SimpleDB, deletando o atributo desejado em cada item passado como parâmetro.

Para localizar os itens no domínio real, o SimpleSQL faz uma consulta sobre o atributo *SimpleSQL_TableName,* adicionado aos itens quando os mesmos são inseridos através do comando INSERT.

3.4.3 DROP TABLE

Para comandos DROP TABLE, o SimpleSQL primeiramente verifica no domínio *Schema* se a tabela a ser deletada existe. Para tanto, é realizada uma consulta sobre o domínio *Schema,* e em seguida é verificado em cada item retornado o nome dele. Se algum dos itens tiver o nome igual ao nome da tabela passada no comando, então a tabela existe e pode ser deletada.

Depois, o SimpleSQL deleta no *Schema* o item que representa a tabela passada no comando, assim como deleta no domínio real todos os itens que correspondem a dados daquela tabela. Para deletar os itens, o SimpleSQL chama o método *DeleteAttributes* da API do SimpleDB, passando como parâmetros o nome do domínio e o nome dos itens, omitindo o nome de atributo para que seja deletado o item todo. Para cada chamada, o método cria uma requisição HTTP e submete ela ao SimpleDB, deletando os itens passados como parâmetro. A localização dos itens no domínio real também é feita através de uma consulta sobre o atributo *SimpleSQL_TableName.*

3.5 Esquema

Para manter a consistência com o que foi definido nos comandos DDL, foi adicionada uma verificação de consistência para os comandos DML. Por exemplo, se um atributo for definido como *unique*, essa condição é verificada para cada comando INSERT ou UPDATE. Além das verificações dos constraints e dos tipos para cada campo, o SimpleSQL também verifica nomes de tabelas e campos

repetidos (para comandos CREATE) ou que não existem (para os demais comandos).

Para possibilitar tais verificações, foi criado um domínio específico, denominado *Schema*. Esse domínio contém todas as definições das tabelas do banco de dados relacional criadas pelo SimpleSQL. Para cada tabela, é criado um item nesse domínio. Cada item corresponde a uma tabela e possui os seguintes atributos:

PRIMARY_KEY: atributo que possui como chave a palavra “PRIMARY_KEY” e como valor o nome da chave primaria definida.

UNIQUE: atributo que possui como chave a palavra “UNIQUE” e como valores o nome de todas as colunas definidas como únicas.

NOT_NULL: atributo que possui como chave a palavra “NOT_NULL” e como valores o nome de todas as colunas definidas como não nulas.

FOREIGN_KEY: atributo que possui como chave a palavra “FOREIGN_KEY” e como valores o nome da chave estrangeira, a tabela a qual ela se refere e a coluna a qual ela se refere, para todas as colunas definidas como chave estrangeira na tabela.

INT: atributo que possui como chave a palavra “INT” e como valores o nome de todas as colunas definidas como inteiros.

DOUBLE: atributo que possui como chave a palavra “DOUBLE” e como valores o de todas as colunas definidas como doubles.

VARCHAR: atributo que possui como chave a palavra “VARCHAR” e como valores o nome de todas as colunas definidas como varchar.

CHAR: atributo que possui como chave a palavra “CHAR” e como valores o de todas as colunas definidas como char.

Além destes atributos, é criado, para cada atributo da tabela, um atributo que tem como chave o nome do atributo da tabela e como valor a palavra “coluna”, a fim de indicar todas as colunas existentes da tabela. A Figura 7 demonstra dois exemplos de comandos CREATE TABLE e como estas tabelas ficam representadas no domínio *Schema*.

```
CREATE TABLE Articles
(ID INT PRIMARY KEY, title VARCHAR(100) NOT NULL,
 author_ID INT FOREIGN KEY REFERENCES Authors(ID) NOT NULL,
 journal VARCHAR(50) NOT NULL, volume INT NOT NULL,
 number INT, year INT NOT NULL, pages VARCHAR(15));

CREATE TABLE Authors
(ID INT PRIMARY KEY, SSN INT NOT NULL UNIQUE,
 name VARCHAR(50) NOT NULL, city VARCHAR(50));
```

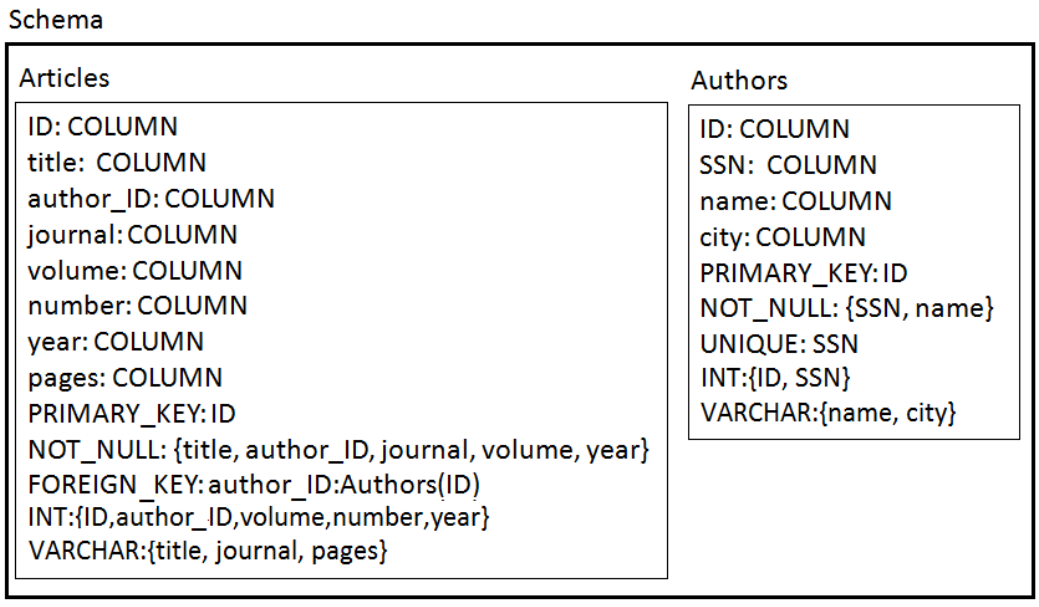

Figura 7: Exemplos de comandos CREATE TABLE e como essas tabelas ficam representadas no domínio *Schema.*

A cada comando submetido, o SimpleSQL acessa o domínio *Schema* para verficar se este comando respeita o que foi definido no esquema do banco de dados.

Em alguns casos, além de acessar o *Schema*, é necessário também acessar o domínio real para fazer as verificações. Por exemplo, se um atributo é definido como *unique* ou como chave primária, a cada comando de inserção é preciso percorrer os domínio real, a fim de verificar se o valor que está sendo inserido já não existe naqueles dados. A figura 8 ilustra a sequência de passos executados para a verificação de um comando de inserção na tabela *Authors*, definida na figura 6.

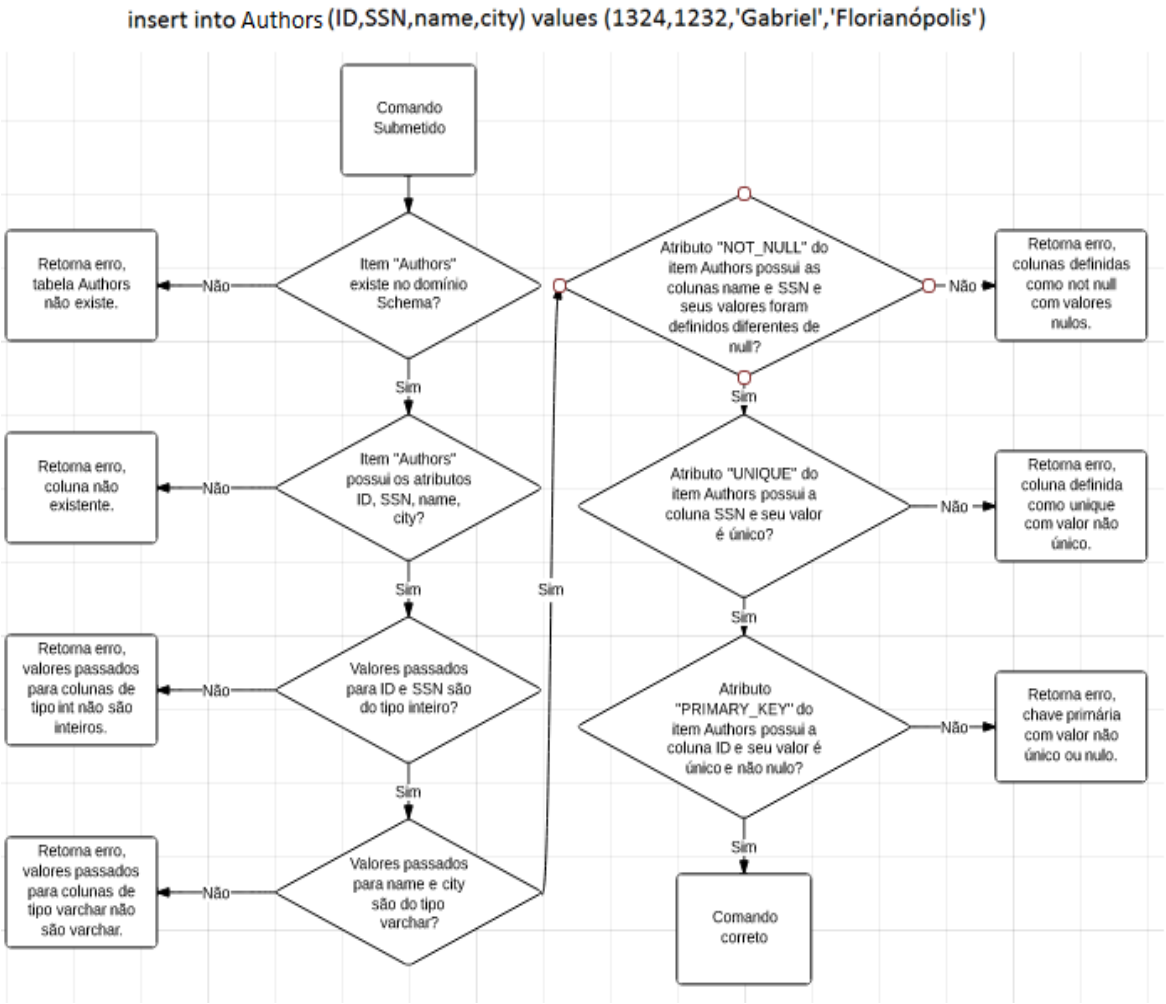

Figura 8: Sequência de passos executados para a verificação de um comando de inserção na tabela *Authors*.

3.6 Interface Cliente

Para realizar testes dos comandos adicionados ao SimpleSQL, foi desenvolvida uma interface cliente. A interface é simples, permitindo a entrada de comandos SQL em uma área de texto. Ela apresenta os resultados de consultas através de tabelas, assim como o tempo de início e fim para cada comando processado. A interface possui três botões:

Fetch: usado para realizar consultas e mostrar o resultado em forma de tabela;

Non-query: usado para realizar comandos que não são consultas.

Print: usado para imprimir no console os dados do domínio *Schema*.

Quando ocorre algum erro no reconhecimento do comando ou com relação à consistência com as definições, um alerta é mostrado, indicando qual erro ocorreu. A Figura 9 apresenta a tela da interface em funcionamento normal, enquanto a Figura 10 ilustra a tela na ocorrência de um erro de duplicação de chave primária em um comando INSERT.

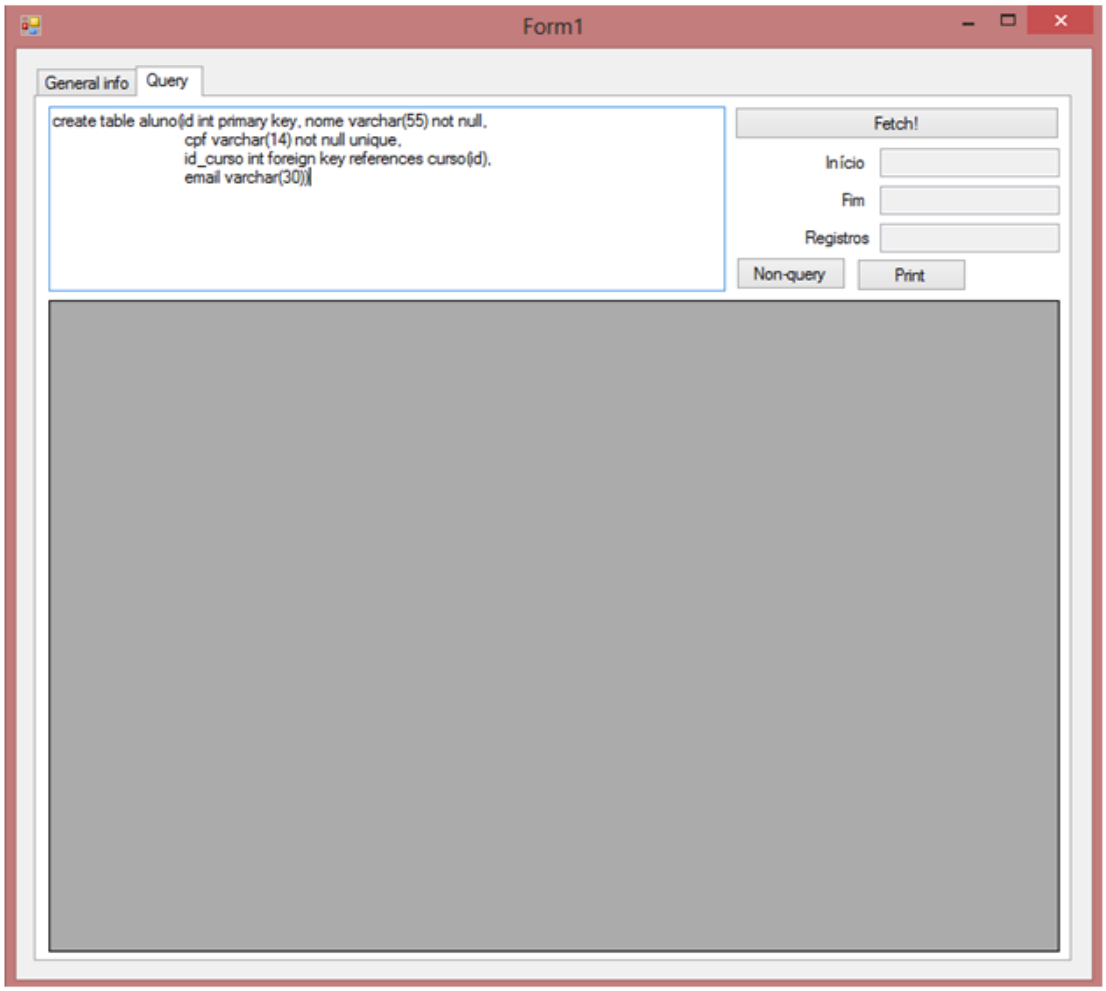

Figura 9: Interface em funcionamento normal.

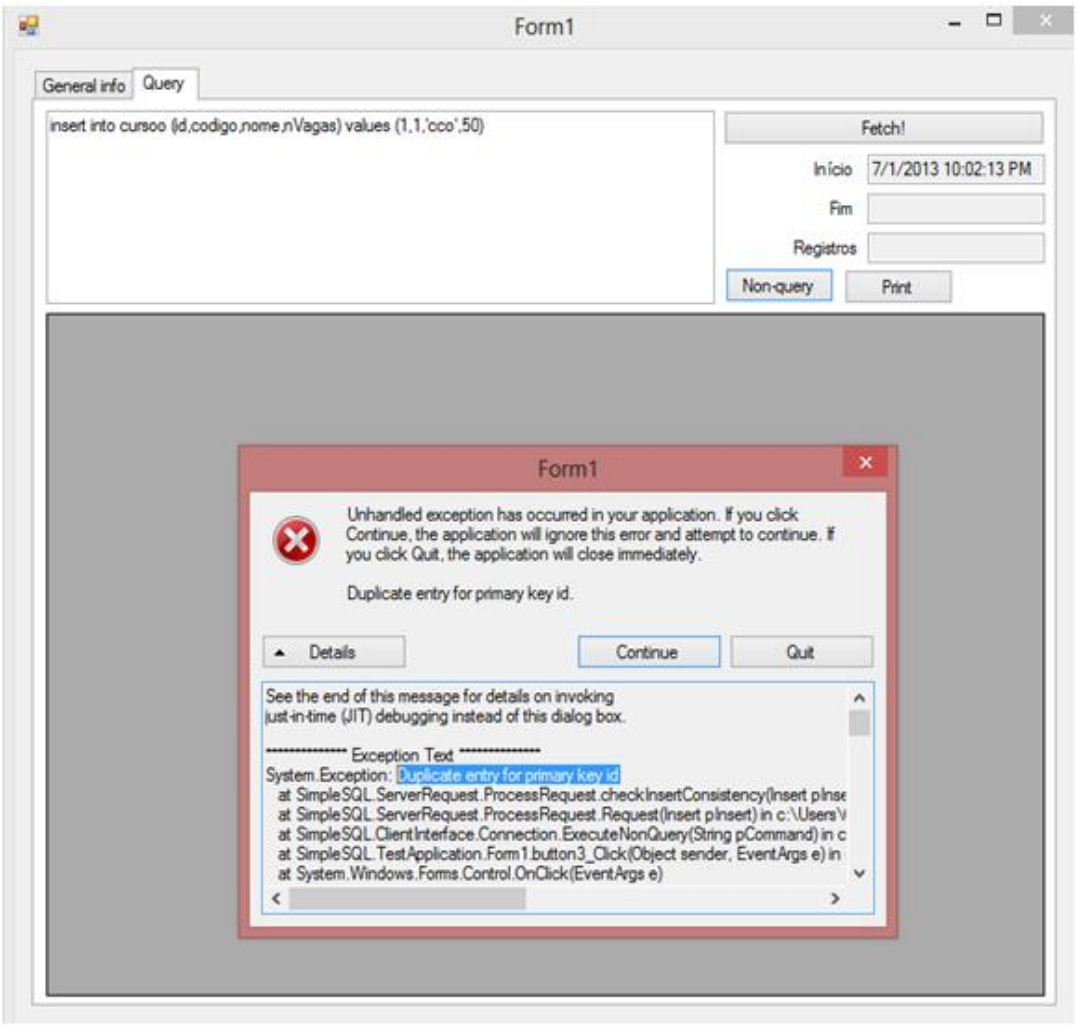

Figura 10: Interface na ocorrência de um erro de duplicação de chave primária em um comando insert.

4 AVALIAÇÃO EXPERIMENTAL

Para avaliar o impacto dos comandos DDL sobre o desempenho do SimpleSQL, foram realizados experimentos sobre uma amostra de dados relacionais, referentes ao sistema de vestibular da UFSC. Foram coletados dados referentes a dois tipos de testes: no primeiro foi medido o tempo de execução para comandos DDL, e no segundo foi medido o tempo de execução de consultas com validações sobre o esquema.

. A amostra utilizada nos testes consiste em seis tabelas que representam candidatos, suas opções de curso, seus resultados e a qual evento (vestibular) eles estão associados. A Figura 11 apresenta o esquema relacional da amostra, que conta com mais de 500 mil tuplas no total.

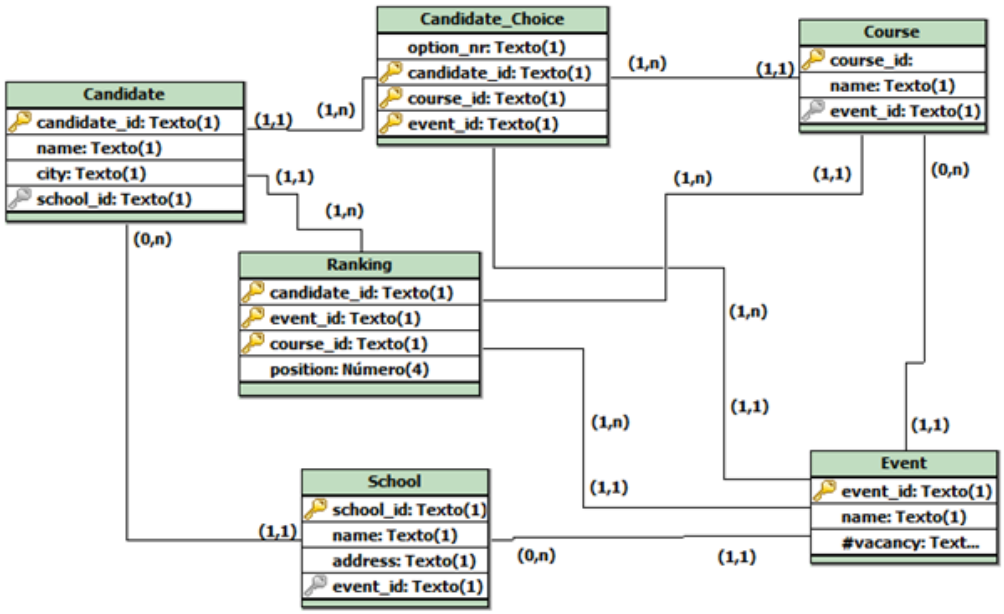

Figura 11: Esquema relacional usado nos testes.

Os experimentos foram realizados no seguinte ambiente computacional:

- Notebook LG A550;
- Processador Intel Core i5;
- Memória RAM 4GB DDR3;
- Conexão de internet 10Mbps ADSL2.

No que diz respeito às configurações do SimpleDB, todos os dados foram persistidos em um único domínio, localizado na região Leste dos EUA, sendo esta a única configuração que a Amazon deixa visível ao usuário.

4.1 Comandos DDL

Nos primeiros testes, foram calculados os tempos para execução de comandos DDL utilizando o SimpleDB sozinho e utilizando o SimpleSQL juntamente com o SimpleDB. Esse teste teve como objetivo avaliar o impacto da utilização do SimpleSQL sobre os comandos DDL, afim de determinar se o *overhead* causado pelo SimpleSQL torna essa execução viável ou não.

A execução dos testes utilizando o SimpleDB sozinho foi feito através de um programa externo que simulava a execução de comandos DDL no SimpleDB, uma vez que o mesmo não oferece suporte a DDL nativamente. Esse programa fez chamadas diretas a API do SimpleDB para executar as operações, sem fazer nenhum tipo de validação ou mapeamento de comando, estando nesses itens o *overhead* observado na execução com SimpleSQL

A tabela 1 apresenta o tempo necessário para executar os comandos DDL

suportados pelo SimpleSQL. Foi executado cada comando três vezes e pego a média dos tempos de cada execução. Na tabela é mostrado o tempo gasto com a criação e remoção de duas tabelas, assim como o tempo gasto na modificação (adição e remoção de colunas) das mesmas. Os testes para os comandos ALTER TABLE e DROP TABLE foram executados sobre a tabela *Candidate* com 50.000 tuplas e a tabela *Course* com 605 tuplas.

Table	Command	Mode	Duration
Candidate	`CREATE TABLE`	SimpleDB	36ms
Candidate	`CREATE TABLE`	*SimpleSQL*	47ms
Course	`CREATE TABLE`	SimpleDB	37ms
Course	`CREATE TABLE`	*SimpleSQL*	51ms
Candidate	`ALTER TABLE (ADD)`	SimpleDB	2h47m51s
Candidate	`ALTER TABLE (ADD)`	*SimpleSQL*	3h14m39s
Candidate	`ALTER TABLE (DROP)`	SimpleDB	3h20m18s
Candidate	`ALTER TABLE (DROP)`	*SimpleSQL*	3h40m51s
Course	`ALTER TABLE (ADD)`	SimpleDB	2m21s
Course	`ALTER TABLE (ADD)`	*SimpleSQL*	2m33s
Course	`ALTER TABLE (DROP)`	SimpleDB	2m24s
Course	`ALTER TABLE (DROP)`	*SimpleSQL*	2m42s
Candidate	`DROP TABLE`	SimpleDB	3h12m41s
Candidate	`DROP TABLE`	*SimpleSQL*	3h27m37s
Course	`DROP TABLE`	SimpleDB	2m18s
Course	`DROP TABLE`	*SimpleSQL*	2m33s

Tabela 1: Tempo de processamento para os comandos DDL.

A Tabela 1 apresenta um tempo de processamento rápido para a criação de ambas as tabelas. Isso se deve ao fato de que apenas dois itens são inseridos pelo SimpleSQL no domínio *Schema* para registrar informações sobre as tabelas criadas. A tabela *Candidate* foi criada com 16 colunas, uma chave primária e duas chaves

estrangeiras, enquanto a tabela *Course* foi criada com 16 colunas, uma chave primária e uma chave estrangeira. Neste caso, tem-se um *overhead* de 33,5% com execução no modo SimpleSQL em relação ao SimpleDB.

Foi relatado um grande tempo de processamento para execução do ALTER TABLE na tabela *Candidate* devido ao grande número de linhas da mesma e também pelo fato de que o SimpleDB tem que acessar todos os itens relacionados a esta tabela a fim de proporcionar a atualização da coluna. Na prática, cada item atualizado corresponde a execução de dois comandos POST no SimpleDB: um para remover o item antigo e um outro para criar um novo item com a modificação desejada. O tempo para atualização do domínio *Schema* também foi incluído para execução no modo SimpleSQL.

Este tempo de processamento elevado para atualização de dados no SimpleDB e na maior parte dos bancos de dados NoSQL revela que estes sistemas são muito mais focados em suportar grandes quantidades de consultas do que grandes quantidades de atualizações. Os métodos de acesso simples, baseados na recuperação de um item de cada vez, não oferecem boas otimizações de busca, diferentemente dos bancos de dados relacionais. Considerando estes fatos, para a tabela *Candidate* tem-se um *overhead* de 15% para adição de coluna e de 10% para remoção de coluna com o SimpleSQL. Para a tabela *Course*, tem-se um *overhead* de 8% para adição e de 12,5% para remoção de coluna.

A execução do comando DROP TABLE sobre a tabela *Candidate* também sofre com o tempo de processamento elevado, pois, como explicado anteriormente, todos os itens relacionados a esta tabela devem ser acessados e excluídos, além da atualização do domínio *Schema*. O *overhead* com o SimpleSQL foi de 7%. Para a tabela *Course*, o *overhead* foi de 10%.

4.2 Validação de Consultas

Foram executadas três consultas simples e uma consulta complexa, com *joins,* sobre o SimpleSQL, ilustradas na Figura 12. Nestes testes, foi avaliado o *overhead* entre executar a consulta sem validações e com validações. O tempo de processamento dessas consultas executadas através do SimpleSQL contra a sua execução apenas no SimpleDB já foi analisado em um trabalho anterior (CALIL; MELLO, 2012).

A tabela 2 mostra os tempos coletados, considerando a média de tempo para três execuções de cada consulta. A diferença no tempo de processamento de cada consulta simples, sem considerar a validação, está relacionada ao número de tuplas retornadas. O grande tempo de processamento para consultas complexas se deve ao fato de que o SimpleDB não suporta nativamente consultas com *joins*. Devido a isso, o SimpleSQL precisa decompor este tipo de consulta em várias consultas simples e executá-las sobre uma única tabela, coletando os resultados individuais e computando todos os *joins* para gerar o resultado final. O *overhead* com a validação das consultas ficou na faixa de 1% até 33%, com um tempo adicional de no máximo 10 segundos para executar tais validações contra o domínio *Schema*.

Query 1:
```
SELECT candidate_id, race
FROM Candidate
WHERE city like 'FLORIAN%';
```

Query 2:
```
SELECT candidate id, race
FROM Candidate
WHERE sex = 'F';
```

Query 3:
```
SELECT *
FROM Course
WHERE area_id = 1 AND name LIKE 'ENGE%'
AND #vacancy >= 100 AND #applications > 1000;
```

Query 4:
```
SELECT Ranking.position, Candidate.city,
School.name, Event.name
FROM Candidate
INNER JOIN School ON Candidate.school_id =
School.school_id
INNER JOIN Ranking ON Candidate.candidate_id =
Ranking.candidate_id
INNER JOIN Event ON Ranking.event_id = Event.event_id
WHERE
Event.event_id = 25 AND School.event_id = 25 AND
Ranking.event_id = 25 AND Candidate.event_id = 25;
```

Figura 12: Consultas executadas nos testes.

Query	no Validation	Validation
1	2m22s	2m24s
2	3m09s	3m13s
3	3s	4s
4	24m51s	25m01s

Tabela 2: Tempos de execução com validação de consultas.

4.3 Comandos com erros

Para certificar que o SimpleSQL realmente é capaz de detectar erros de consistência nos comandos submetidos, vários testes com comandos contendo erros foram executados. As figuras de 13 até 19 mostram alguns destes testes feitos, apresentando o comando submetido e uma mensagem contendo o erro de definição encontrado no comando.

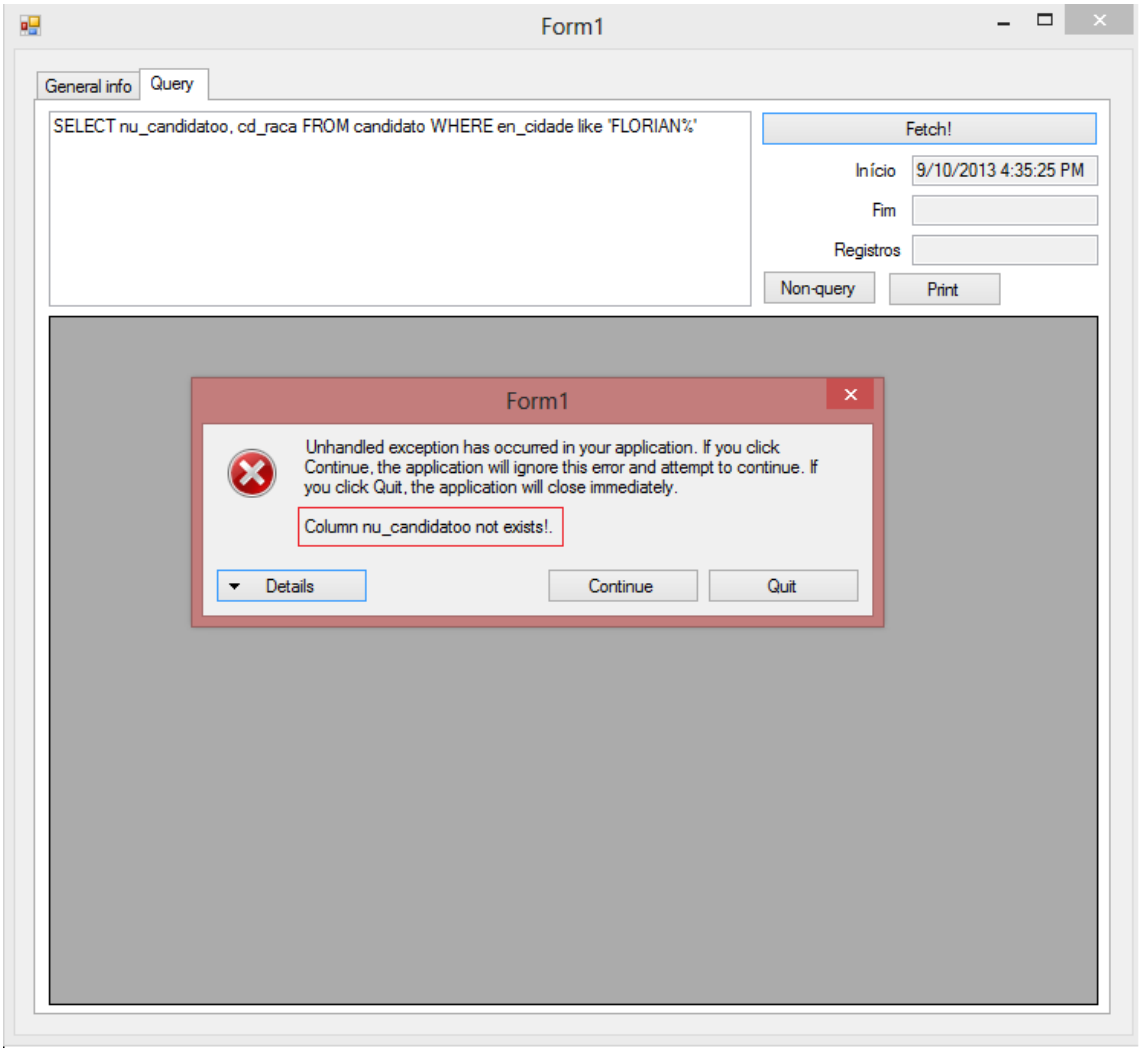

Figura 13: Erro de coluna não existente.

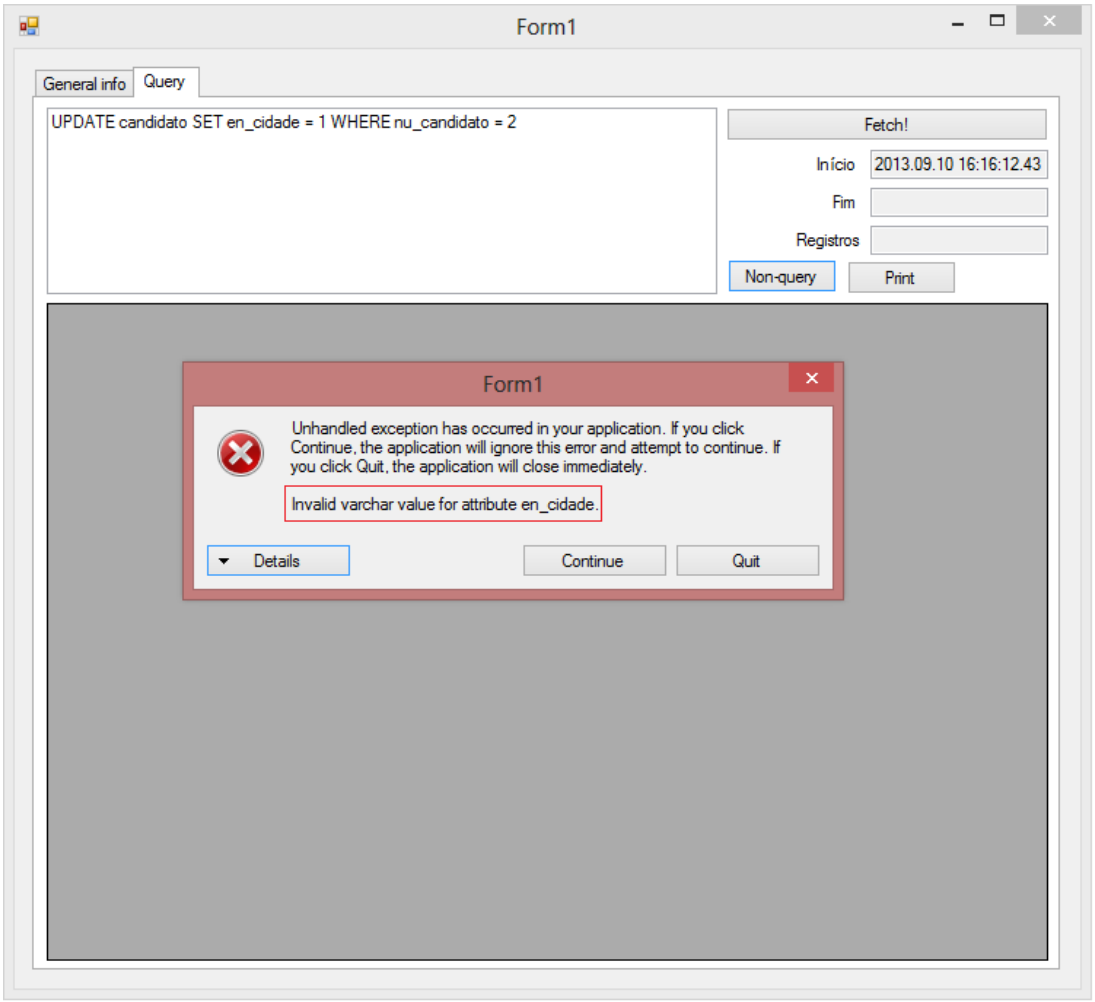

Figura 14: Erro de definição de tipo.

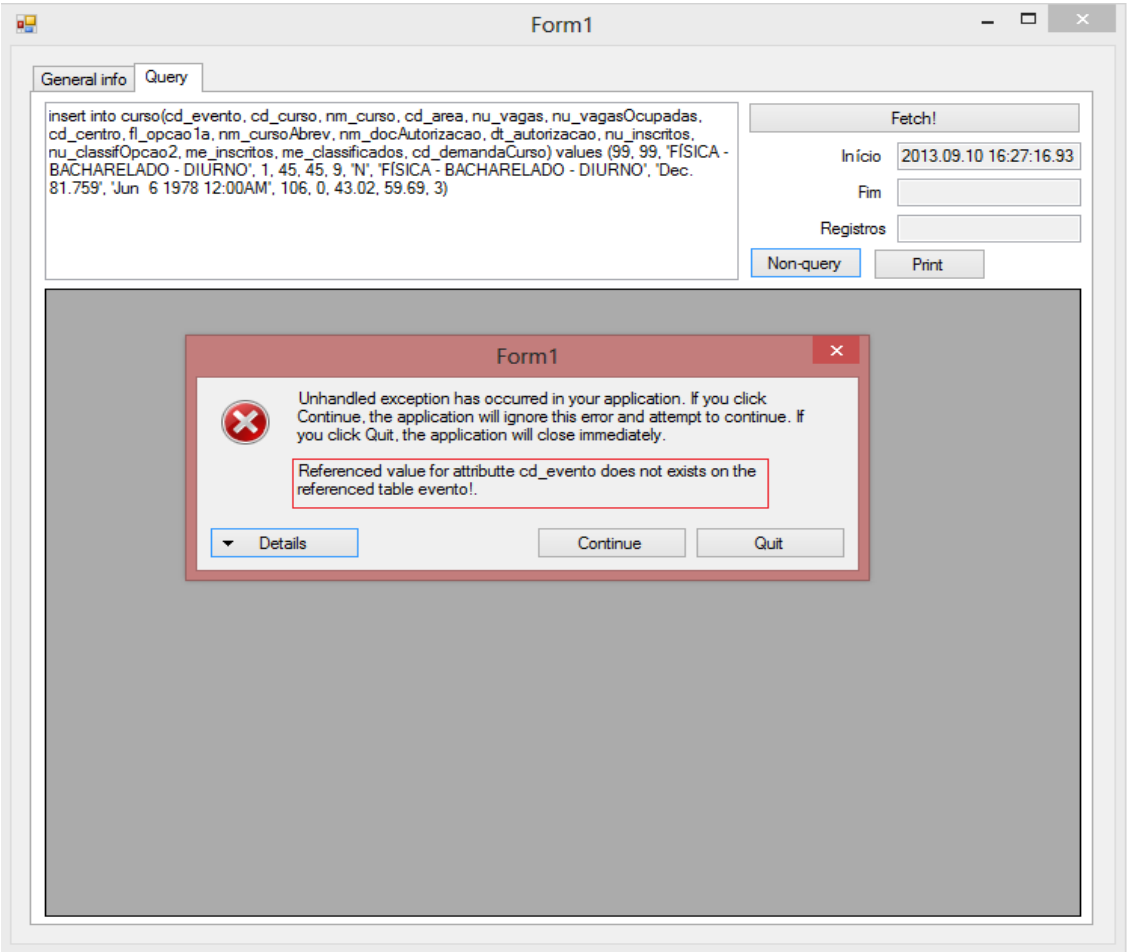

Figura 15: Erro de referência a chave estrangeira.

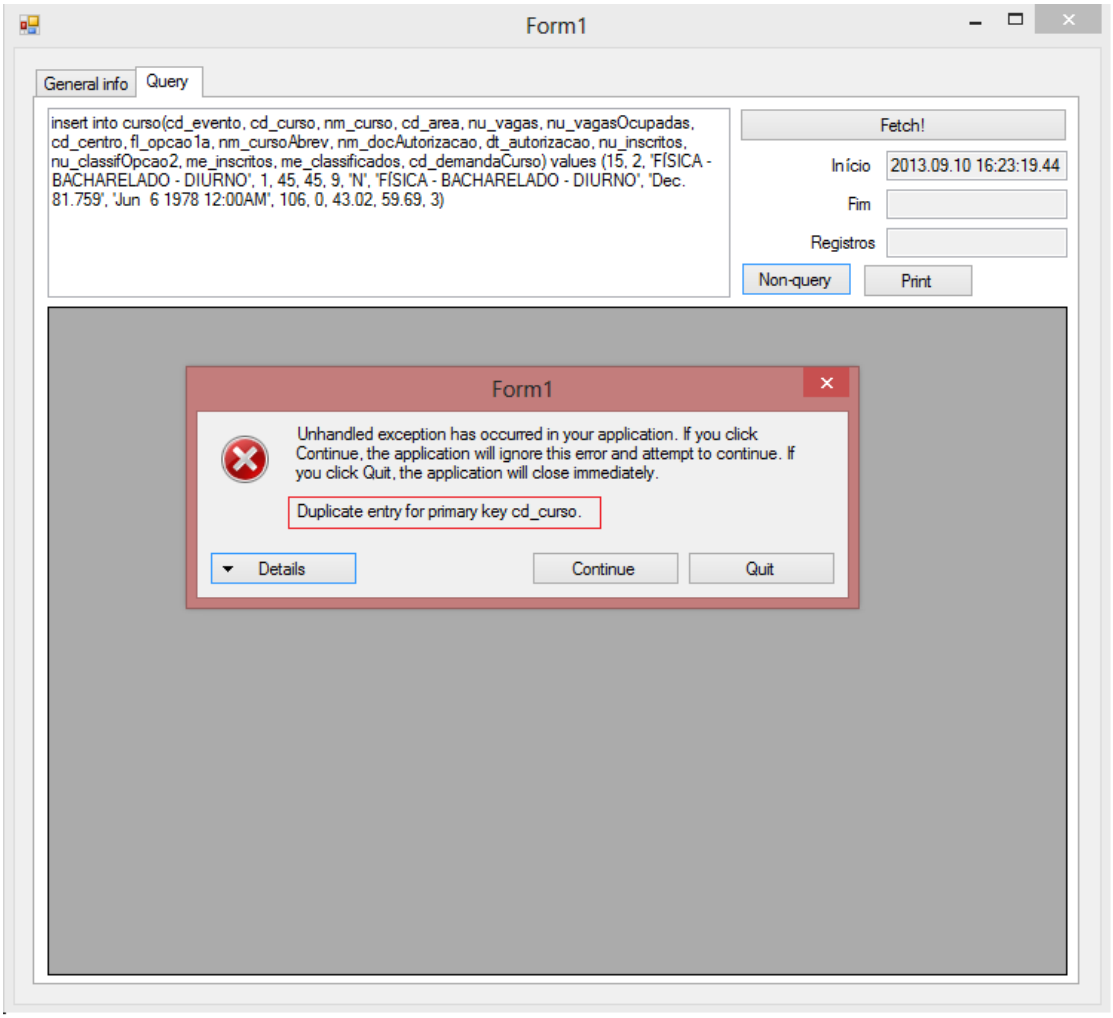

Figura 16: Erro de duplicação de chave primária.

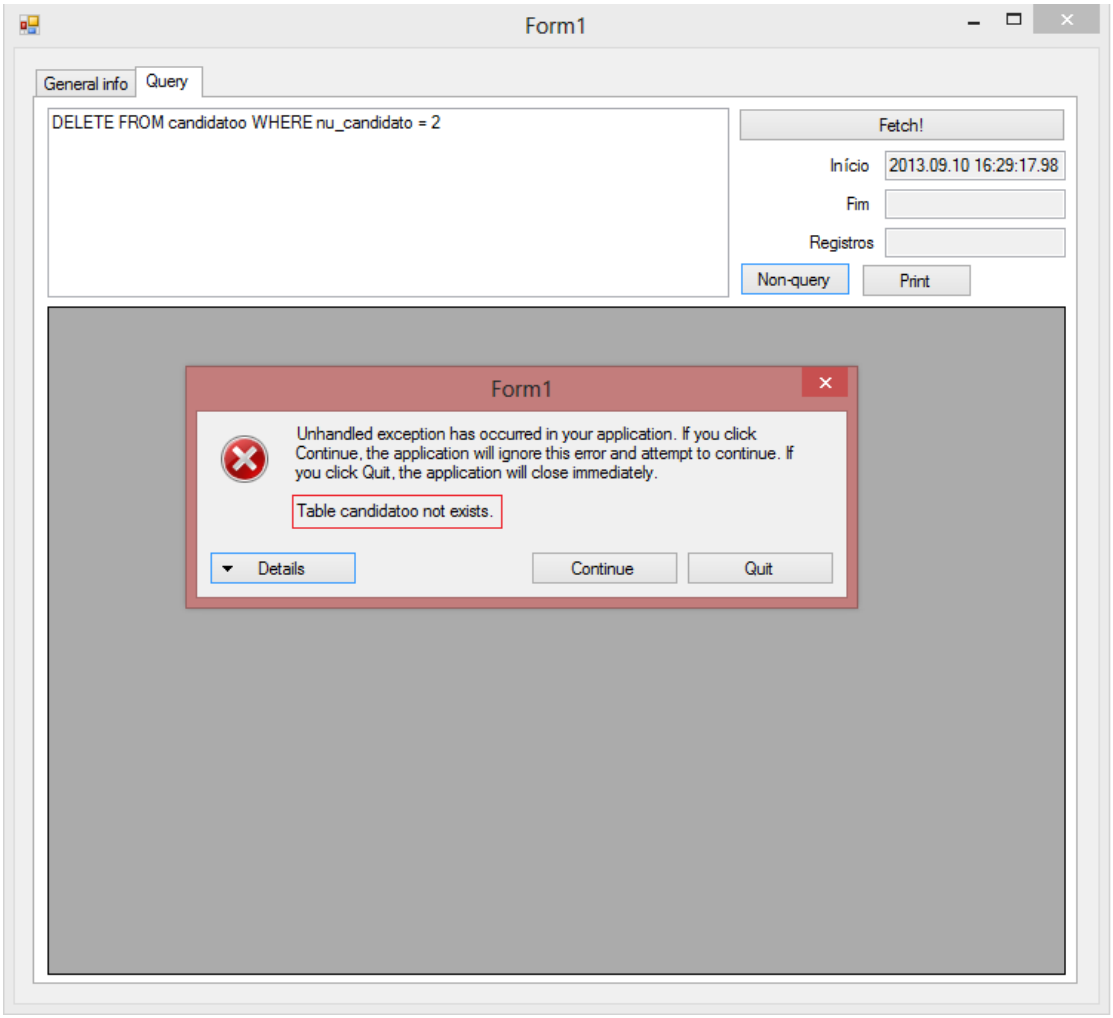

Figura 17: Erro de tabela não existente.

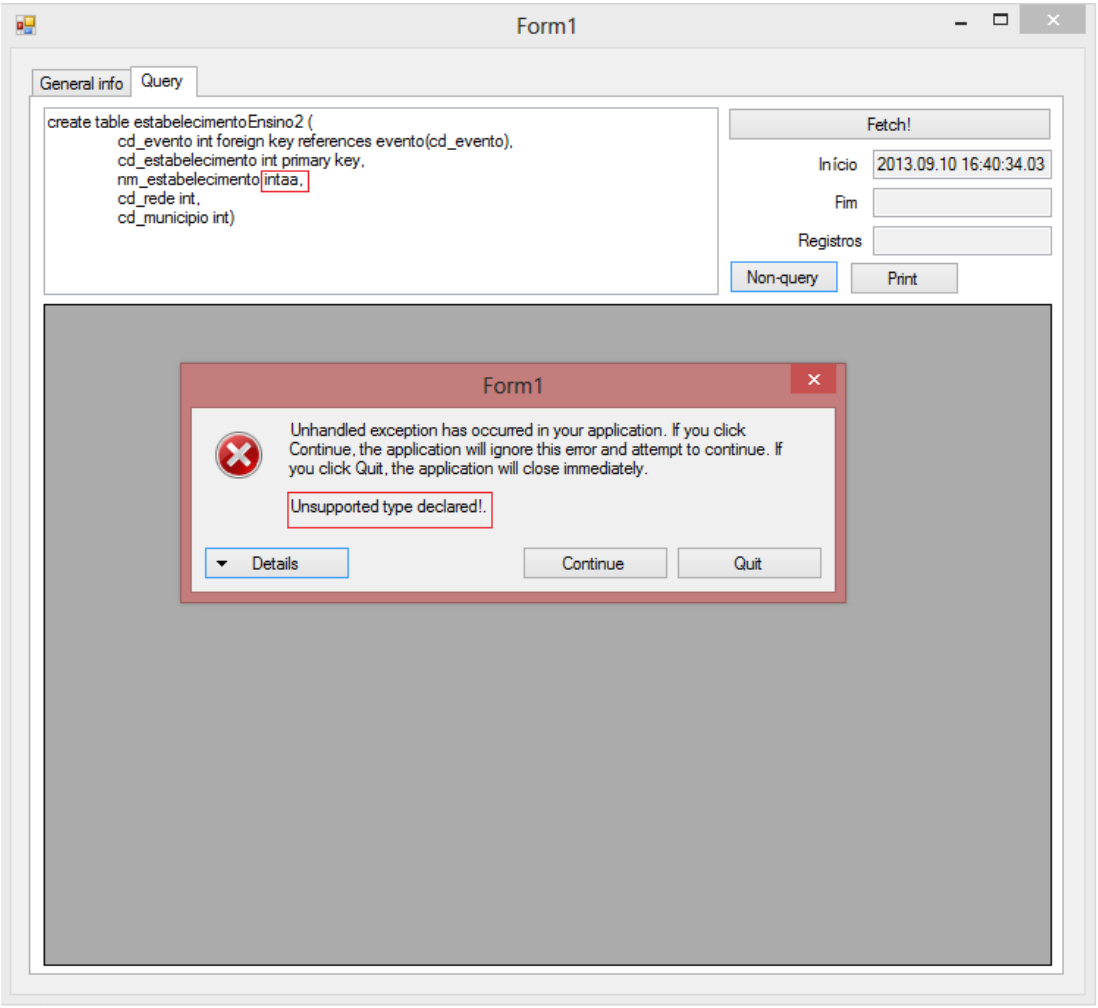

Figura 18: Erro de tipo não suportado.

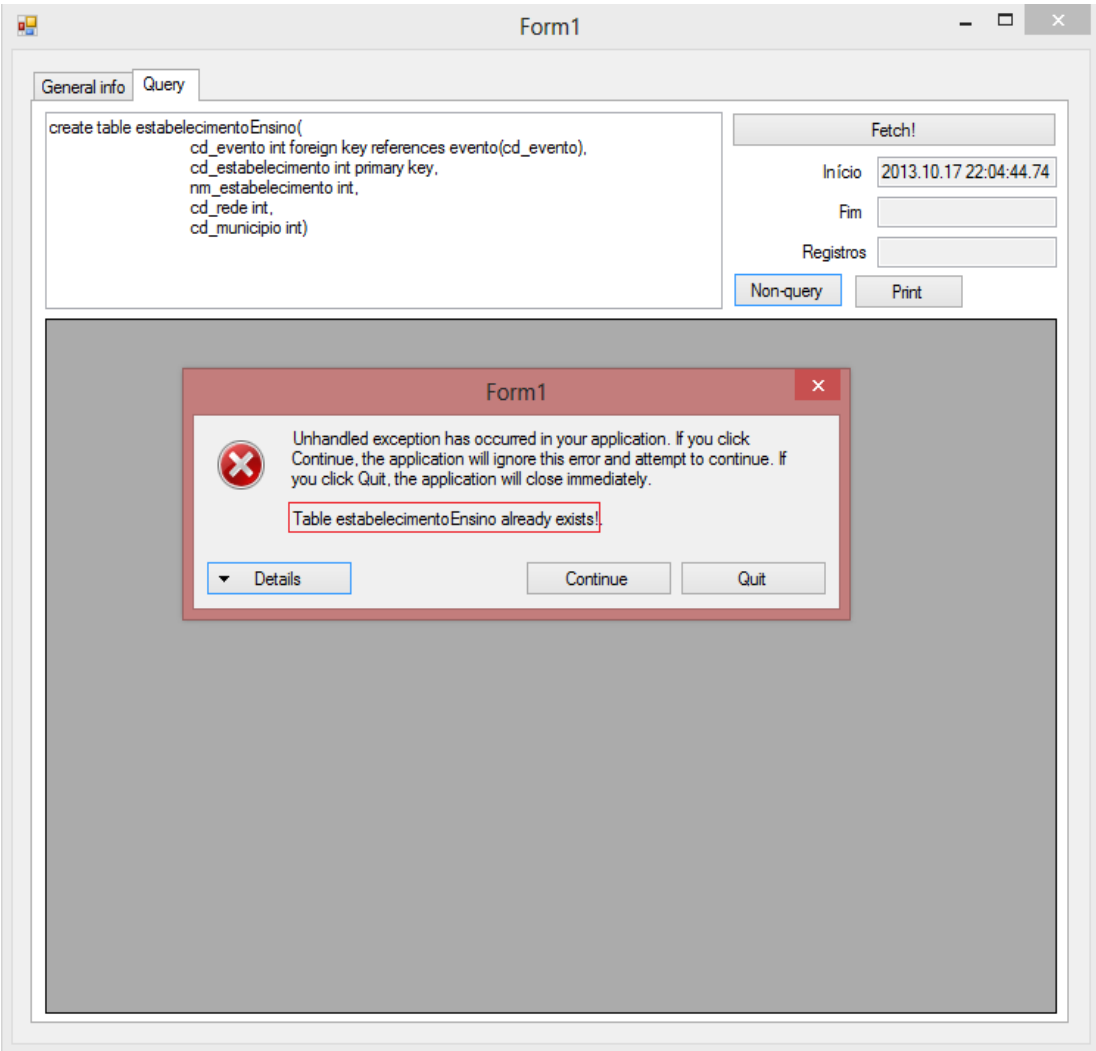

Figura 19: Erro de tabela já existente.

Cada um dos testes foi executado três vezes, calculando-se em seguida a média dos tempos obtidos em cada um. Os tempos para os testes das figuras 13 até 19 tiveram tempo de 0.84 segundos, 0.9 segundos, 1.76 segundos, 3.28 segundos, 0.43 segundos, 0.25 segundos e 0.23 segundos, respectivamente.

5 TRABALHOS RELACIONADOS

O interesse pela relação entre bancos de dados relacionais e de nuvem vem crescendo cada vez mais. Como principal trabalho relacionado e que serviu de inspiração para este, cita-se o trabalho que propôs a criação do SimpleSQL (CALIL; MELLO, 2012).

Vários trabalhos propõem sistemas para gestão de dados relacionais na nuvem, com o objetivo de proporcionar instruções SQL escaláveis. Curino et al. (2012) visa dar suporte adicional para as propriedades ACID, a fim de garantir a disponibilidade dos dados e consistência. Este trabalho propõe o armazenamento de dados relacionais explicitamente na nuvem, diferentemente do SimpleSQL, que trabalha como uma camada que faz a tradução do modelo relacional para o não relacional, armazenando os dados na nuvem de forma não relacional(NoSQL).

Dois trabalhos relacionados propõem a utilização dos bancos de dados NoSQL Scalaris (ARNAUT; SCHROEDER; HARA, 2011) e Cloudy (EGGER, 2009) para persistir dados do MySQL, proporcionando um mapeamento entre os seus esquemas de dados. A estratégia de mapeamento nestes trabalhos foi codificada para um SGBD relacional específico, seguindo a mesma idéia de Bernstein et al. (2011), que estende o SQL Server para executar na nuvem. Diferentemente destes, o SimpleSQL propõe uma abordagem muito mais simples, servindo como uma API independente de SGBD, que pode ser usada por qualquer aplicação cliente que queira persistir dados na nuvem e manter uma visão relacional destes dados.

Outro trabalho semelhante a este é o DQE, que também é uma camada SQL, sobre o banco de dados NoSQL HBase (VILAÇA et al., 2013). Diferentemente do

SimpleSQL, DQE faz o mapeamento apenas de comandos DML para o HBase, sem oferecer suporte para comandos DDL.

É importante notar que nenhum destes trabalhos apresenta uma aproximação específica para tratar da questão de definição dados e mapeamento de comandos DDL, o que demonstra o pioneirismo deste trabalho neste quesito.

6 CONCLUSÃO

A utilização de bancos de dados na nuvem vem crescendo cada vez mais e traz consigo grandes vantagens, dentre as quais se destacam a redução de custos e de esforço de manutenção. Apesar destas vantagens, ainda existe muita dificuldade para se utilizar efetivamente esses bancos de dados, pois a grande maioria das aplicações atuais trabalha sobre o modelo relacional, enquanto que a maioria das soluções de gerenciamento de dados na nuvem adota algum modelo de dados NoSQL.

Devido a essa problemática, foi proposto o SimpleSQL, uma camada de mapeamento relacional ao SimpleDB, um banco de dados NoSQL que segue o modelo orientado a documentos. Apesar de ser uma solução específica para o SimpleDB, o SimpleSQL pretende evoluir para uma proposta genérica de mapeamento entre o modelo relacional e o modelo orientado a documentos, que é um tema para pesquisas futuras.

Inicialmente, o SimpleSQL provia suporte apenas para os comandos DML do SQL. Contudo, os comandos DDL são de fundamental importância para organização e uso eficiente do banco de dados, assim como são necessários para prover ao usuário final do SimpleSQL um mapeamento adequado para o modelo relacional.

Este trabalho contribui com essa problemática adicionando ao SimpleSQL suporte a comandos de definição de dados. Desta forma, o usuário além de poder fazer cargas e consultas sobre seus dados na nuvem de forma relacional, é capaz também de definir o esquema de dados, da mesma forma como é feito nos SGBDs relacionais, tendo assim garantias que seus os comandos DML respeitam o que foi

definido no esquema de dados, como nomes de tabelas, colunas, *constrains* e tipos de dados.

Como mostrado na avaliação experimental, o *overhead* no processamento dos comandos DDL no SimpleSQL não é proibitivo se comparado a sua execução com SimpleDB puro. Para a criação de tabelas, o *overhead* observado foi menos de 34%. Apesar de parecer um grande *overhead* a primeira vista, os tempos de execução foram muito pequenos, não ultrapassando a ordem de dezenas de milissegundos. No que diz respeito à atualização do esquema de dados (ALTER TABLE e DROP TABLE), foram obtidos resultados muito bons, tendo um *overhead* máximo de 15% para um volume grande de dados mantidos na nuvem.

Os testes de validação de comandos DML também apresentaram um resultado satisfatório, tendo *overhead* abaixo de 34% e tempo adicional de processamento de no máximo 10 segundos, até mesmo para consultas complexas que faziam a junção de varias tabelas do esquema relacional.

Vale ainda ressaltar que os resultados obtidos com o desenvolvimento e avaliação deste trabalho geraram um artigo científico que foi aceito para apresentação e publicação na conferência *iiWAS 2013 (XV International Conference on Information Integration and Web-based Applications and Services)*. Essa é uma conferência classificada como B3 no Qualis da CAPES.

Por fim, assim como a criação do SimpleSQL em um trabalho anterior inspirou a criação deste trabalho, espera-se que este trabalho sirva de alguma forma para que outros trabalhos nessa área venham a ser feitos. Algumas sugestões para trabalhos futuros:

- Aprimorar as operações SQL atualmente suportadas para aceitar mais recursos da linguagem, como suporte a gerenciamento de índices, por exemplo;
- Elaborar uma ferramenta que opere sobre um volume de dados existente, preparando-o para o SimpleSQL, já que versão atual do SimpleSQL só é capaz de trabalhar com dados que ele mesmo tenha inserido, devido a dependência do atributo *SimpleSQL_TableName*;
- Propor uma interface de acesso padrão a bancos de dados NoSQL, que permita ao SimpleSQL trabalhar com outros bancos de dados na nuvem além do SimpleDB, sendo este o principal problema a ser tratado atualmente.
- Buscar otimizações no processamento das instruções SQL dentro do SimpleSQL e assim melhorar os tempos dos experimentos já realizados sobre o SimpleSQL, tanto para instruções DML quanto para DDL.

BIBILIOGRAFIA

[1] Abadi, D. J.: Data management in the cloud: Limitations and opportunities. IEEE Data Eng. Bull., 32:3-12 (2009).

[2] Cattell, R.: Scalable SQL and NoSQL Data Stores. SIGMOD (2010).

[3] G. Coulouris, J. Dollimore, T. Kindberg: Distributed Systems: Concepts and Design, 5ª edition. Addison-Wesley, (2011).

[4] Amazon SimpleDB, http://aws.amazon.com/simpledb/. Acesso em setembro de 2013.

[5] Amazon SimpleDB, Getting Start Guide, http://docs.amazonwebservices.com/AmazonSimpleDB/latest/GettingStartedGuide/Welcome.html?r=1. Acesso em setembro de 2013.

[6] Amazon Web Services .NET SDK, http://aws.amazon.com/pt/sdkfornet/. Acesso em junho de 2013.

[7] Working with Amazon SimpleDB, http://www.codeproject.com/Articles/186625/Working-with-Amazon-SimpleDB. Acesso em setembro de 2013.

[8] Fowler, Martin. Nosql Definition, http://martinfowler.com/bliki/NosqlDefinition.html. Acesso em junho de 2013.

[9] Amoroso, Danilo. O que é computação em nuvens? http://www.tecmundo.com.br/computacao-em-nuvem/738-o-que-e-computacao-em-nuvens-.htm. Acesso em setembro de 2013.

[10] D. F. Arruda, J. A. F. Moura Júnior. Banco de dados em nuvem: conceitos, gerenciamento e desafios, http://www.slideshare.net/darlanarruda/banco-de-

dados-em-nuvem-conceitos-gerenciamento-e-desafios. Acesso em setembro de 2013.

[11] Pritchett, D.: BASE, an ACID alternative. ACM Queue (2008).

[12] A. Calil and R. dos Santos Mello. SimpleSQL: A Relational Layer for SimpleDB. 16th East-European Conference on Advances in Databases and Information Systems (ADBIS) - Lecture Notes in Computer *Science*, 39:99–110, September 2012.

[13] MSDN Library, Regular Expression Language, http://msdn.microsoft.com/en-us/library/az24scfc.aspx. Acesso em 03 de setembro de 2013.

[14] MSDN Library, C# Programming Guide, http://msdn.microsoft.com/en-us/library/67ef8sbd(v=vs.90).aspx. Acesso em 03 de setembro de 2013.

[15] Amazon Relational Database Service. http://aws.amazon.com/rds/, Acesso em 03 de setembro de 2013.

[16] M.-J. Hsieh, C.-R. Chang, L.-Y. Ho, J.-J. Wu, and P. Liu. SQLMR: A Scalable Database ManagementSystem for Cloud Computing. In *International Conference on Parallel Processing (ICPP)*, page 315, September 2011.

[17] D. G. Campbell, G. Kakivaya, and N. Ellis. Extreme Scale with Full SQL Language Support in Microsoft SQL Azure. In *ACM SIGMOD International Conference on Management of Data*, pages 1021–1024, June 2010.

[18] C. Curino, E. P. C. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational Cloud: a Database Service for the Cloud. In *5th Biennial Conference on Innovative Data SystemsResearch (CIDR)*, pages 235–240, January 2011.

[19] D. Arnaut, R. Schroeder, and C. S. Hara. Phoenix: A Relational Storage Component for the Cloud. In *IEEE CLOUD*, pages 684–691, July 2011.

[20] D. Egger. SQL in the Cloud. Master's thesis, Swiss Federal Institute of Technology Zurich (ETH), Swiss, 2009.

[21] P. A. Bernstein, I. Cseri, N. Dani, N. Ellis, A. Kalhan, G. Kakivaya, D. B. Lomet, R. Manne, L. Novik, and T. Talius. Adapting Microsoft SQL Server for Cloud Computing. In *27th International Conference on Data Engineering (ICDE)*, pages 1255–1263, April 2011.

[22] R. Vilaça, F. Cruz, J. Pereira, and R. Oliveira. An Effective Scalable SQL Engine for NoSQL Databases. In *Distributed Applications and Interoperable Systems-13th IFIP WG 6.1 International Conference (DAIS)*, pages 155–168, June 2013.

[23] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing. UC Berkeley Reliable Adaptive Distributed Systems Laboratory (2009).

[24] Sousa, F. R. C., Moreira, L. O., de Macêdo, J. A. F., Javam, C. M.: Gerenciamento de Dados em Nuvem: Conceitos, Sistemas e Desafios. Em: Tópicos em sistemas colaborativos, interativos, multimídia, web e bancos de dados. Mini-cursos do XXV Simpósio Brasileiro de Banco de Dados (2010).

[25] Jacobs, D. R., Carvalho, V. C.: DDL, Lidando com as diferenças das instruções SQL nos diferentes SGBD's. http://www.sirc.unifra.br/artigos2006/SIRC-Artigo25.pdf. Acesso em outubro de 2013.