Amazon SimpleDB

Felipe Martins Mesquita, Renato Caminha Juaçaba Neto
Departamento de Computação
Universidade Federal do Ceará
felipe.mesquita@lsbd.ufc.br

22 de janeiro de 2013

Resumo

A criação e a manutenção de uma infraestrutura de banco de dados costuma ser um dos principais gastos(em termos de tempo e finanças) de sistemas de software comerciais modernos. Os gastos incluem a aquisição da plataforma de hardware necessária para suportar o sistema de gerenciamento de banco de dados(*SGBD*), conexões de rede e — dependendo do porte do sistema — contratação de um administrador de banco de dados(*DBA*). Tais gastos são rotineiramente um fator limitante para a criação de sistemas comerciais por *startups* e empresas de pequeno porte que não têm condições de arcar com esses custos.

Com a popularização da computação em nuvem e a oferta de serviços segundo o modelo PaaS(*Platform as a Service*), surgiram alternativas ao modelo de banco de dados *in house* mencionado acima que provêem infraestruturas de bancos de dados como um serviço, geralmente cobrando de acordo com a carga de trabalho utilizada(armazenamento, transferência de dados, processamento, etc).

Este relatório técnico tem como objetivo prover uma visão geral sobre as características, o conceito e o uso de um serviço específico desse gênero, o banco de dados NoSQL em nuvem **Amazon SimpleDB**, bem como esclarecer o contexto em que a ferramenta se apresenta.

Sumário

Capítulo 1

Introdução

Neste capítulo, será feita uma revisão do contexto em que os serviços de bancos de dados em nuvem(em particular, nesse relatório, o serviço Amazon SimpleDB) se apresentam e suas características gerais.

1.1 Bancos de dados relacionais

1.1.1 História

Os bancos de dados relacionais foram introduzidos na década de 1970 em [1] como uma alternativa aos modelos *de navegação*[5]. Os sistemas de bancos de dados que seguiam os modelos de navegação(presentes na década de 1960) apresentavam ao programa cliente a capacidade de percorrer um conjunto de dados segundo um esquema de lista encadeada, onde os dados requisitados pelo programa eram encontrados seguindo-se uma cadeia de referências e verificando se o objeto atualmente referenciado possui a propriedade buscada — não apresentando, portanto, funcionalidade de busca(a própria aplicação é responsável por filtrar os dados desejados).

O conceito apresentado por Codd foi implementado pela IBM em seu SGBD *System R*, que também foi o primeiro a apresentar, em 1978, o uso da então padronizada linguagem de consulta **SQL**. Constatada a viabilidade e superioridade dos sistemas de bancos de dados relacionais em relação às demandas existentes, várias outras empresas passaram a implementar seus SGBDs comerciais baseando-se no modelo relacional. Entre as principais, figuram **Oracle**(com o SGBD homônimo), Microsoft(com o **Microsoft SQL Server**) e ainda a IBM(com seu **DB2**, derivado do System R).

Alternativamente, surgiram também implementações disponíveis para uso gratuito: na Suécia, foi iniciado o projeto *open source* **MYSQL** por Michael Widenius [**?**]; na Universidade de Berkeley, em um projeto liderado por Michael Stonebraker, criou-se o SGBD **PostgreSQL**(o nome vem do fato dele ter se baseado no antigo projeto **Ingres**, ou seja, *Post Ingres* [7])

1.1.2 Características

No sistema proposto por Codd, os dados se organizavam em *tabelas* com entradas de tamanho fixo, não em uma lista de referências como proposto no modelo de navegação. Cada entrada em uma tabela conteria atributos e um certo subconjunto(possivelmente próprio) desses atributos identificaria unicamente uma entrada nessa tabela.

Um dos problemas que o modelo de Codd se propunha a resolver era a representação de dados *esparsos*: isto é, como representar e armazenar de maneira eficiente dados que possam apresentar entradas ou atributos vazios. No modelo de navegação, a solução para esse cenário é o uso de um marcador específico para denotar que o atributo está vazio, o que é bastante ineficiente em termos de armazenamento. O modelo relacional, por sua vez, sugeria colocar esses atributos que tinham a possibilidade de serem vazios em uma tabela separada daquela em que se encontra a entrada à qual eles se referem.

Por exemplo, suponha que nossos dados consistam de um cadastro onde as pessoas podem ou não fornecer um telefone para contato. Esses dados seriam representados como uma duas tabelas: uma contendo uma entrada para cada pessoa com as respectivas informações pessoas e a outra contento, em cada entrada, um número de telefone e uma *referência* para a entrada na primeira tabela cuja pessoa forneceu aquele número(ver figura 1.1).

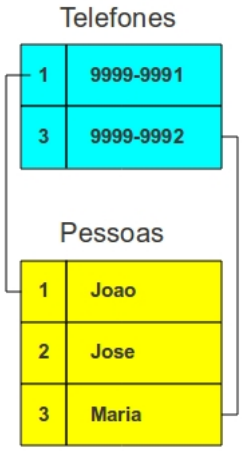

Figura 1.1: Tabelas inter-relacionadas

Uma característica que se tornou peculiar nos SGBDs relacionais, apesar de não fazer parte da proposta original do modelo relacional, foi o uso de uma linguagem de consulta interpretada pelo SGBD para a extração de dados, a **SQL**(*Structured Query Language*). Baseada na álgebra relacional, ela permite realizar operações de busca e atualização sobre as tabelas e os dados contidos no banco expressando-se de maneira declarativa sobre **conjuntos** de dados. Dessa forma, o programa cliente não recupera dados do SGBD através de chamadas a uma *API*, mas sim submetendo consultas SQL que serão interpretadas pelo SGBD.

1.2 Bancos NoSQL

1.2.1 Escalabilidade de SGBDs relacionais

Os SGBDs relacionais ganharam rápida popularidade a partir da década de 1980, passando a ser o modelo padrão usado em aplicações comerciais. Com o crescimento das bases de dados, no entanto, as técnicas convencionais de organização de dados segundo o modelo relacional passaram a apresentar performance insatisfatória, exigindo a criação de uma série de estratégias no sentido de corrigir esse problema.

A mais simples delas é comumente classificada como **escalabilidade vertical**, que consiste em adicionar recursos(armazenamento, processamento, memória, etc) às unidades individuais(computadores) que compõem o sistema de banco de dados. Tal estratégia em geral não exige trabalho de reconfiguração do software que gerencia o banco de dados, salvo o ajuste de alguns parâmetros dependendo do contexto.

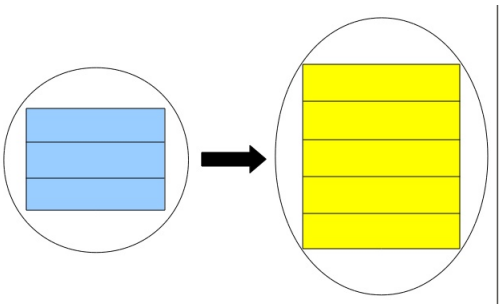

Figura 1.2: Escalabilidade Vertical

Uma alternativa é replicar o banco de dados em várias máquinas para permitir que o acesso seja feito de forma distribuída, evitando que um único nó torne-se um gargalo para o acesso aos dados. Ela é vantajosa quando predominam operações de leitura por parte da aplicação cliente já que, quando ocorre uma escrita no banco de dados, esta deve ser propagada para todos os nós que o compõem. Existem maneiras de amenizar este problema, porém elas esbarram na dificuldade de manter a **consistência** dos dados ao longo dos vários nós que formam o sistema. Tais abordagens que envolvem a adição de nós formando um sistema gerenciador de banco de dados distribuído são classificadas como **escalabilidade horizontal**.

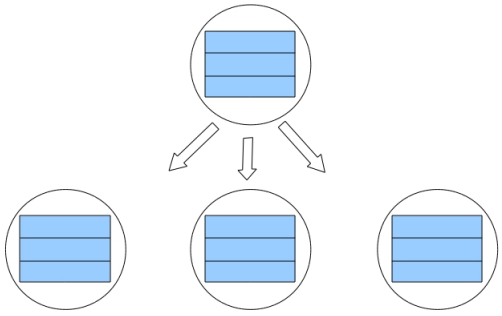

Figura 1.3: Escalabilidade Horizontall

A técnica de escalabilidade vertical tem como vantagem sua simplicidade: em geral nao há necessidade de reconfigurar o SGBD após sua aplicação. A escalabilidade horizontal, no entanto, apresenta um ganho de performance muito maior ao custo de um aumento de complexidade na manutenção da consistência do banco de dados, cuja responsabilidade passa a ser da aplicação cliente e não mais do SGBD.

1.2.2 NoSQL

Devido à dificuldade de escalabilidade horizontal dos SGBDs relacionais, foram propostas alternativas que abrem mão de certas propriedades e garantias dadas pelo modelo relacional em troca de simplicidade e fácil escalabilidade. Os bancos de dados **NoSQL** são, em uma classificação genérica, sistemas de gerenciamento de bancos de dados que não aderem ao modelo relacional - isto é, não organizam os dados em tabelas com esquemas fixos e geralmente não usam SQL para manipulação de dados[8].

Por organizarem seus dados de formas simples(abrindo mão da flexibilidade fornecida pelo modelo relacional), em geral os SGBDs NoSQL simplificam e tornam mais robusto o processo de escalabilidade horizontal[10]. Eles se popularizaram com a adoção desses sistemas em aplicações web de larga escala, como o *Google*, *Facebook* e *Reddit*.

Tipos

Os SGBDs NoSQL diferem nas abordagens de organização de dados. Os tipos mais comuns são:

- **Orientados a documentos**: nesses sistemas, os dados são separados em **documentos**, geralmente um por cada "linha"no equivalente relacional. A cada documento é atribuído um identificador único(uma *chave*) e os atributos presentes em cada documento podem variar. Os dados podem ser acessados especificando a chave do documento procurado ou alguma propriedade do conteúdo do documento. Alguns exemplos desse tipo são *Apache CouchDB*, *Lotus Notes* e *MongoDB*. Em particular, o serviço ilustrado nesse relatório, o *Amazon SimpleDB*, faz parte dessa categoria;

- **Orientados a grafos**: aqui os dados são organizados segundo um modelo de **grafo**, onde cada entidade possui referências diretas a outras entidades do banco de dados. Eles são particularmente úteis para representar, por exemplo, relações de amizade em redes sociais e conectividade de estradas. São SGBDs orientados a grafos o *Neo4J*, o *InfiniteGraph* e *FlockDB*;

- **Bancos chave-valor**: os bancos de dados chave-valor permitem que os dados sejam armazenados sem o uso de um esquema fixo. Cada entrada de dados consiste em uma *string* que representa a **chave** e os dados correspondentes, que formam o **valor** associado à chave. SGBDs NoSQL como *Apache Cassandra*, *Redis* e *Amazon DynamoDB* seguem essa abordagem.

1.3 Nuvem

O paradigma que veio a ser conhecido como **Computação em Nuvem** surgiu da confluência de interesses diversos.
Por parte dos provedores, havia (1) a constatação de que infraestruturas mantidas *in house* para a implantação de aplicações comerciais ficam subutilizadas na maior parte do tempo(mesmo aqueles serviços de larga escala), representando portanto um desperdício de recursos. Além disso, (2) tornava-se cada vez mais evidente a demanda por serviços que terceirizassem tarefas de montagem e manutenção de infraestrutura de forma barata, dado que tais custos representavam uma grande barreira para a implantação e oferta de aplicações de software por pequenos empreendedores e *startups*.
Os consumidores dessas aplicações, por sua vez, (3) demandavam o acesso a serviços de software que aproveitassem a disponibilidade de acesso à Internet para que suas funcionalidades e seus dados estivessem disponíveis em qualquer ambiente no qual eles os acessassem.

A computação em nuvem, de maneira geral, atende a essas demandas através do fornecimento de recursos computacionais(hardware e software) como serviços acessados via rede[9]: a primeira levou à solução de particionar recursos de hardware para diferentes usos a fim de aproveitar ao máximo as capacidades daquela plataforma; a segunda motivou a criação de serviços que disponibilizam infraestrutura de hardware e software compartilhada a baixo custo via acesso remoto; a terceira, por fim, foi atendida pelo surgimento do modelo de *Software como Serviço* descrito a seguir.

1.3.1 Modelos de serviço

Os serviços de computação em nuvem diferem em seu escopo e público-alvo. Essas categorias em que se classificam os serviços de nuvem são conhecidas como *modelos de serviço*.Os principais modelos de serviços em nuvem são[9]:

- **Infraestrutura como Serviço**(*IaaS*): este é o modelo de serviço em nuvem mais básico. Aqui o serviço consiste em fornecer acesso a máquinas físicas ou, o que é mais comum, a instâncias de máquinas virtuais e outros recursos. Os usuários acessam essas máquinas remotamente e instalam o ambiente de software necessário para o seu consumo. O serviço é responsável por controlar o acesso aos recursos computacionais do sistema de nuvem de acordo com a necessidade e cobrar de acordo com o uso(por exemplo, se o cliente necessita de mais poder computacional, o sistema pode disponibilizar núcleos de processamento adicionais e cobrar pelo uso deles). Serviços que se encaixam nessa categoria incluem **Amazon EC2**, **Windows Azure** e **Google Compute Engine**;

- **Plataforma como Serviço**(*PaaS*): no modelo PaaS os provedores fornecem uma **plataforma de desenvolvimento e implantação** que inclui um sistema operacional, sistema de gerenciamento de banco de dados, servidor web, etc. Os usuários desses serviços são desenvolvedores de software que desejam desenvolver e implantar sistemas sem arcar com os custos de tempo e dinheiro associados à compra, montagem e manutenção dessa plataforma necessária para o desenvolvimento e oferta do sistema.

- **Software como Serviço**(*SaaS*): o modelo de software como serviço é caracterizado pela oferta de **funcionalidades de software** em uma infraestrutura de nuvem online. Os usuários não precisam gerenciar a estrutura do provedor de nuvem para acessar o serviço: eles apenas enxergam a funcionalidade da aplicação. O modelo apresenta uma série de vantagens para os fornecedores e clientes do serviço:

– O serviço é facilmente escalável, bastando replicar o serviço em várias máquinas diferentes e usar um *load balancer* para definir em tempo de execução qual máquina provê o serviço a cada cliente;
– A logística de atualiações de software é drasticamente simplificada, pois as atualizações só precisam ser aplicadas na infraestrutura de nuvem que abriga o serviço;
– Os clientes podem acessar o serviço sem precisar instalar programas adicionais em suas máquinas e têm seus dados do serviço disponíveis em qualquer ambiente em que eles o acessarem.

Exemplos de serviços em nuvem populares que seguem o modelo SaaS são *Google Docs* e *OnLive*.

O relacionamento entre esses modelos está ilustrado na figura 1.4(retirada de [9])

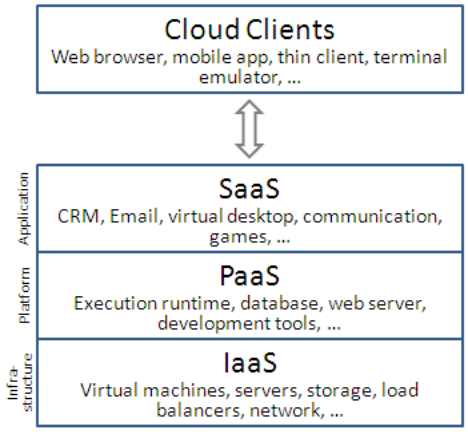

Figura 1.4: Modelos de serviço em nuvem

Outros modelos que não serão abordados em detalhes são **Rede como Serviço**(*NaaS*), **Armazenamento como Serviço**(*STaaS*), **Dados como Serviço**(*DaaS*), **Virtualização de Desktops**, entre outros.

Capítulo 2

Amazon SimpleDB

Neste capitulo faremos uma introdução ao Amazon SimpleDB, suas características e sua interface do ponto de vista do usuário.

2.1 Introdução

O **Amazon SimpleDB** foi o primeiro serviço de banco de dados a ser ofertado como parte da plataforma de serviços de nuvem **Amazon Web Services** segundo o modelo PaaS. Ele é implementado como um sistema gerenciador de banco de dados NoSQL distribuído, implementado na linguagem de programação *Erlang* devido às facilidades que ela oferece para o desenvolvimento de aplicações distribuídas.
Ele representa um serviço barato para aplicações que precisam de uma infraestrutura de banco de dados escalável, simples e cujos dados possam ser representados sem se ajustarem ao modelo relacional. O serviço cobra uma taxa de acordo com o armazenamento, transferência e *throughput* de dados pela rede.

2.2 Características

Por ser um SGBD NoSQL distribuído, o Amazon SimpleDB não possui algumas garantias que são encontradas em SGBDs relacionais não-distribuídos. Em princípio, seria desejável obter um sistema que garantisse as seguintes propriedades:

- **Disponibilidade**: o sistema deve continuar utilizável mesmo que alguns dos nós que o compõem falhem;
- **Tolerância a particionamento**: se alguns nós do sistema perderem o contato entre si na rede que os interconecta, o sistema deve continuar operacional;
- **Consistência**: Os dados presentes em um nó que compõe o banco não devem entrar em conflito com os dados em outro nó.

O teorema CAP, no entanto, assevera que é impossível que um sistema distribuído apresente as três propriedades relacionadas acima ao mesmo tempo[12]. O Amazon SimpleDB abre mão da consistência em favor de uma *consistência eventual*: ela apenas assegura que, dado tempo suficiente sem atividades de alteração no banco de dados, eventualmente todas os nós do banco estarão consistentes entre si[13].
A API do serviço permite que a aplicação cliente especifique diferentes graus de consistência(leituras consistentes, semântica transacional, etc) ao realizar operações sobre o banco. Tais exigências, no entanto, pioram a performance do acesso ao serviço e devem ser usadas somente quando necessárias.

2.2.1 Limites

O serviço possui as seguinte limitações[11]:

- máximo de 250 domínios por conta;
- máximo de 10GB em cada domínio;
- máximo de 1 milhão de atributos por domínio;
- máximo de 256 atributos por item;
- máximo de 1KB por atributo;
- máximo de 2500 itens retornados por consulta;
- timeout de 5 segundos por consulta;
- no máximo 1 atributo referenciado em cada predicado de consulta;
- máximo de 22 operadores em cada predicado de consulta;
- máximo de 20 predicados em cada consulta.

2.2.2 Modelo de dados

O Amazon SimpleDB difere-se de um SGBD relacional convencional principalmente no fato de não organizar os dados em tabelas, mas sim em **domínios**. Estes, por sua vez, são compostos por **itens** com nomes únicos que agrupam um conjunto de **atributos**, cada um consistindo num par chave-valor, sendo que todo atributo possui um índice criado automaticamente pelo sistema. Deve-se ressaltar que um atributo pode ter múltiplos valores(o seu valor é um vetor). Essa organização se encontra ilustrada na figura 2.2.2. Os itens em um mesmo domínio não precisam ter os

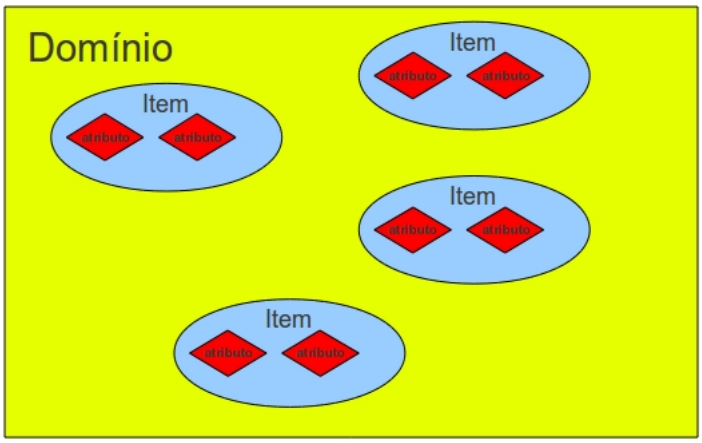

Figura 2.1: Modelo de dados do Amazon SimpleDB

mesmos atributo e todos os atributos são opcionais(isto é, aceitam valores vazios).

Os valores são armazenados como strings independentemente do seu tipo - fica a cargo da aplicação cliente fazer as conversões e operações de *marshalling* necessárias. Uma consequência desse fato é que o programador deve ter cuidado para converter os tipos de forma que operações de comparação mantenham-se consistentes. Valores numéricos devem ser completados com zeros à esquerda para evitar que, por exemplo, o número 123 seja classificado como anterior ao número 54(já que, lexicograficamente, esso é o comportamento esperado); caso se deseje trabalhar com números negativos, deve-se adicionar um offset ao número armazenado e, quando ele for requisitado, o offset deve ser subtraído para se obter o valor original; datas devem ser convertidas para formatos que preservem ordenamento lexicográfico, como o ISO 8601 define.

Uma diferença fundamental entre SGBDs relacionais e o Amazon SimpleDB é a ausência de suporte a operações de *joins* neste último. Se o modelo de dados utlizados na aplicação cliente contém dados relacionados entre si, a própria aplicação deve implementar algum mecanismo para agregar entradas de dados relacionadas (ou usar um SGBD relacional ao invés de uma opção NoSQL). Note que essa abordagem pode levar a um modelo de dados *não-normalizado* — isso, porém, faz parte do tradeoff envolvido no uso de soluções NoSQL: abre-se mão da normalização de dados e restrições de consistência em favor de performance, escalabilidade e simplicidade.

2.3 Operações

A API do Amazon SimpleDB expõe uma série de operações para que a aplicação cliente crie e modifique os dados do banco. Os detalhes do uso dessa API na linguagem python serão vistos no próximo capítulo, porém alguns pontos são independentes de linguagem e serão vistos brevemente nesta seção(as descrições a seguir foram tiradas de [3]).

2.3.1 Domínios

A API provê métodos para criação, listagem e remoção de domínios de uma determinada conta do serviço.
Na criação de um domínio, deve-se especificar o nome do domínio criado possuindo de 3 a 255 caracteres, sendo os caracteres válidos letras, números, *underline*, hífen e ponto. Os nomes de domínios também devem ser únicos no contexto de uma mesma conta. A chamada pode retornar 3 erros:

- **MissingParameter**: retornado quando há alguma falha na passagem do parâmetro(nome do domínio);
- **InvalidParameterValue**: retornado quando o nome de domínio requisitado é inválido;
- **NumberDomainsExceeded**: retornado quando o limite na quantidade de domínios já foi atingido.

Note que se já existir um domínio com o nome especificado, a chamada não retorna falha e também não há qualquer efeito sobre o domínio existente.
Outras operações expostas pela API são a remoção de um domínio(espeficando-se o nome dele) e a listagem dos domínios existentes.

Por fim, é possível obter metadados a respeito de um domínio. As propriedades retornadas ao se requisitar os metadados são

- Número de itens no domínio;
- Tamanho em bytes de todos os nomes de itens no domínio;
- Número de nomes únicos de atributos no domínio;
- Tamanho em bytes de todos os nomes(únicos) de atributos no domínio;
- Número de pares atributo/valor no domínio;
- Tamanho em bytes dos valores armazenados no domínio;
- Momento em que os metadados foram computados, retornado como um timestamp com formato Unix time.

2.3.2 Itens

Não existem operações que lidem diretamente com itens no Amazon SimpleDB. A criação de itens se dá indiretamente pela adição de um atributo. Ao fazer isso, especifica-se o nome do item que terá aquele atributo e, se o item não existir, ele será criado.
Da mesma forma, um item é removido quando o último atributo pertencente ao item é removido.

2.3.3 Atributos

Criação e atualização

A API permite a criação de atributos passando-se como parâmetros o nome do domínio, o nome do item ao qual o atributo pertencerá, o nome do atributo e seu valor. Para adicionar mais de um valor a um atributo, basta adicionar os valores individualmente no mesmo atributo em várias chamadas à operação.Por exemplo, se a operação se dá pela chamada a um método **putAttribute**, podemos fazer as seguintes chamadas para armazenar múltiplos valores no atributo 'genero' do item 'livro1' pertencente ao domínio 'livros':

putAttribute('livros', 'livro1', 'genero', 'romance')
putAttribute('livros', 'livro1', 'genero', 'juvenil')

É possível especificar uma flag *Replace* para informar que o valor passado deve substituir o valor atual do atributo ao invés de ser anexado a ele.
A aplicação cliente pode adicionar semântica transacional à chamada dessa operação ao passar como parâmetro(opcional) um par atributo/valor e uma flag *ExpectedExists*: caso a flag esteja ativada, a chamada à API só retornará sucesso se esse par existir no banco de dados(ou se o atributo existir, caso o valor seja omitido no par passado como parâmetro); se a flag não estiver ativada, a chamada retornará sucesso se o par **não** existir.

Caso nenhum erro seja retornado, as alterações foram submetidas e, devido à consistência eventual, podem não ter seus efeitos percebidos até alguns instantes depois. Os erros que podem ser retornados são:

- **InvalidParameterValue**: erro retornado quando o limite de 1KB é ultrapassado pelo nome do item, nome do atributo ou valor do atributo;
- **MissingParameter**: análogo ao erro retornado na criação de domínios;
- **NoSuchDomain**: retornado se o domínio passado não existir;
- **NumberDomainBytesExceeded**: esse erro é retornado quando o limite de 10GB por domínio é excedido;
- **NumberDomainAttributesExceeded**: retornado quando o limite de 1 bilhão de atributos por domínio é alcançado;
- **NumberItemAttributesExceeded**: retornado quando o limite de 256 atributos por item é alcançado;

Um detalhe importante a respeito do funcionamento da adição de atributos é que, quando o limite de dados do domínio é alcançado, a chamada a essa operação irá falhar mesmo que ela esteja substituindo o valor de um atributo e diminua a quantidade de dados armazenados — isto é, enquanto o domínio estiver em seu limite de dados, toda chamada de adição ou atualização de atributos irá falhar, devendo a aplicação cliente remover atributos para corrigir o problema. Por causa disso, recomenda-se que cada item tenha no máximo 255 atributos(ao invés do limite real de 256), pois assim pode-se realizar operações de atualização sem precisar remover atributos antes.

Consulta

A consulta a atributos é feita passando-se como parâmetros o nome do domínio e nome do item cujos atributos estão sendo requisitados. Caso não se queira consultar todos os atributos, pode-se

passar como parâmetro opcional uma lista com os nomes dos atributos requisitados.
A API permite que uma flag *ConsistentRead* seja passada como argumento opcional à operação de consulta para garantir que a leitura seja consistente com as operações de inserção/atualização realizadas antes dela. Como mencionado na seção anterior, ela deve ser usada somente quando necessária, pois a garantia de leituras consistentes aumenta a latência de acesso ao banco de dados comprometendo a performance da aplicação.
Caso não ocorra nenhum erro, os atributos são retornados como uma lista de pares chave-valor, sendo que atributos com múltiplos valores aparecem várias vezes nessa lista(com um valor diferente em cada par).
Os erros que podem ser retornados são **NoSuchDomain** e **InvalidParameterValue**, ambos análogos aos discutidos no item anterior.
Em relação aos erros, é importante salientar que requisitar os atributos de um item que não existe não se configura como uma exceção: uma lista vazia é retornada nesse cenário. Esse comportamento é permitido porque, devido à consistência eventual, é possível que itens existentes em uma réplica do seu domínio não tenham se propagado para a réplica que atende a operação de requisição de atributos naquele momento.

Remoção

Ao se remover um item, pode-se especificar se o item inteiro(com todos os seus atributos e respectivos valores) será removido, se apenas determinados atributos ou ainda(no caso de atributos com múltiplos valores) valores específicos de um atributo.
A operação recebe como parâmetros obrigatórios o nome do domínio e do item. Se nenhum outro parâmetro for passado, o item será removido inteiramente do domínio. É possível passar como parâmetro opcional uma lista de pares atributo/valor especificando quais valores em quais atributos devem ser removidos. Caso se queira remover um atributo inteiro independente de seus valores, passa-se esse atributo na lista como uma entrada sem valor.
Analogamente à operação de inserção e atualização de atributos, pode-se adicionar semântica transacional à operação de remoção especificando-se um par atributo/valor como parâmetro opcional. As possíveis exceções dessa operação são análogas às anteriores, sendo que basta um atributo ou valor inválido na lista passada como parâmetro para que a operação inteira seja considerada inválida.

Criação e atualização em lote

Quando o número de atualizações ou inserções a ser feito é muito grande(na ordem de dezenas), submeter essas operações individualmente implica um *overhead* bastante sensível, já que serão feitas várias requisições ao servidor do banco de dados. Felizmente, a API do Amazon SimpleDB oferece a funcionalidade de submeter essas operações **em lote** realizando apenas uma chamada ao serviço. Os parâmetros passados são análogos à criação e atualização convencional de atributos, sendo que os nomes dos itens são passados como uma lista. Associado a cada nome de item, há uma lista de pares de atributos e valores a serem criados/atualizados. Também é possível utilizar a flag *Replace* para informar que o valor passado deve substituir o valor atual do atributo, ao invés de ser anexado a ele. Essa flag é passada como uma lista contendo uma entrada para cada entrada na lista de pares atributo/valor.
Assim como na versão tradicional da operação de criação/atualização, podem ser retornados os erros **InvalidParameterValue**, **NoSuchDomain**, **NumberDomainBytesExceeded**, **NumberDomainAttributesExceeded** e **NumberItemAttributesExceeded**. Além desses, também podem ser retornados os erros:

- **DuplicateItemName**: retornado se houver dois itens com o mesmo nome em entradas diferentes da lista de itens;
- **NumberSubmittedItemsExceeded**: esse erro é retornado quando se tenta passar mais de 25 itens em um lote;
- **TooLargeRequest**: retornado quando a requisição ultrapassa o limite de dados de 1MB;

Salienta-se que, caso algum parâmetro cause um erro, a operação inteira é abortada e nenhuma das criações e atualizações será submetida ao banco de dados.

2.3.4 Seleção

A API do Amazon SimpleDB expõe uma operação de *Select* que permite que a aplicação consulte dados do banco usando uma linguagem de consulta similar à SQL. Assim como na operaç ao de consulta, pode-se forçar que as leituras sejam feitas de forma consistente pela passagem de uma flag na chamada à API.
Os detalhes dessa linguagem de consulta serão apresentados na seção a seguir.

2.4 Consultas

O Amazon SimpleDB utiliza uma linguagem de consulta parecida com o SQL tradicional. O formato básico de uma consulta é

SELECT ***lista de atributos ou COUNT***
FROM ***domínio***
[*WHERE* ***expressão***]
[***instrução de ordenação***]
[*LIMIT* ***limite***]

Note que a linguagem de consulta do Amazon SimpleDB não permite o uso de expressões INSERT, UPDATE e DELETE como no SQL tradicional. Essas operações são feitas através de chamadas diretas à API do SGBD. Outra diferença — já mencionada — é a ausência de JOINs.

2.4.1 Regras gerais

Nomes de atributos e de domínios podem conter letras, números, crifões e *underlines* e não podem ser iniciados por um número. Caso se viole essa regra, esses nomes devem vir entre aspas nas consultas. Elas também não podem, naturalmente, ser uma das palavras-chaves da sintaxe de consulta do Amazon SimpleDB. As palavras-chaves são

SELECT	IN	NULL
FROM	BETWEEN	ORDER
WHERE	AND	BY
LIKE	EVERY	DESC
INTERSECTION	IS	ASC
OR	NOT	LIMIT

Assim como em SQL, pode-se requisitar todos os atributos em um SELECT usando-se um asterisco(*) e pode-se especificar os atributos requisitados separando-os por vírgula. A única função de agregação da linguagem de consulta do Amazon SimpleDB é a função COUNT, usada de maneira similar à SQL.
Os predicados da cláusula WHERE podem usar os operadores tradicionais da SQL, incluindo "IS (NOT) NULL", LIKE e BETWEEN. Além desses, há um operador IN() que permite testar se um atributo está dentro de um conjunto discreto de valores. Por exemplo:

*SELECT * FROM clientes WHERE nome IN('jose', 'joao', 'maria')*

Os valores usados em predicados sempre são strings, logo eles devem estar entre aspas(simples ou duplas) independente do atributo correspondente possuir um valor numérico ou textual.

2.4.2 Cláusula de Ordenação

A ordem em que os itens são retornados como resultado de uma consulta sem cláusula de ordenação não é definida — os itens não são ordenados por nenhum atributo específico e tampouco pela ordem de inserção. Para se forçar uma ordenação nos itens retornados, deve-se usar uma cláusula de ordenação na consulta através da expressão ORDER BY após a cláusula WHERE(caso ela exista) seguido do atributo sobre o qual será feita a ordenação(seguido, ainda, de uma expressão opcional ASC ou DESC que especifica se a ordenação será feita em ordem crescente ou decrescente, respectivamente. Por padrão, a ordem usada é **crescente**).

Uma restrição imposta pelo Amazon SimpleDB é que o atributo sobre o qual será feita a ordenação não pode conter valores vazios. A própria sintaxe da consulta deve dar alguma garantia de que a ordenação será feita apenas em itens que não contenham valores vazios naquele atributo. Por exemplo, a consulta abaixo

*SELECT * FROM clientes ORDER BY nome*

seria considerada inválida, pois não há garantia de que todos os itens retornados possuam algum valor no atributo 'nome'. Uma maneira simples de tornar a consulta válida e garantir que esse cenário não ocorra é acrescentar um filtro NOT NULL à consulta:

*SELECT * FROM clientes WHERE nome IS NOT NULL ORDER BY nome*

Deve-se notar que quando há uma cláusula WHERE sobre o mesmo atributo de ordenação, a consulta será válida já que a cláusula WHERE por si filtrará os itens que possuem valor vazio no atributo.

2.4.3 Cláusula LIMIT

A cláusula LIMIT á análoga à utilizada na SQL. Caso ela não seja usada, o valor default do número de resultados retornados em uma consulta do Amazon SimpleDB é 100.
Um detalhe importante a ser observado em relação ao número de itens retornados em uma consulta é que nem sempre ela irá retornar o número de itens esperados. Se o tamanho da resposta exceder 5MB ou a consulta demorar mais de 5 segundos, a chamada ao banco irá retornar os itens processados até aquele momento. Recomenda-se, então, sempre verificar se esta condição foi encontrada para que ela seja tratada adequadamente.

2.4.4 Atributos com múltiplos valores

A linguagem de consulta do Amazon SimpleDB possui mecanismos específicos para se trabalhar com atributos de múltiplos valores: o operador INTERSECTION e a função EVERY().
Para ilustrar as dificuldades de se trabalhar com tais atributos usando a sintaxe especificada até esse ponto, considere o seguinte cenário: temos um domínio 'livros' com itens que possuem o atributo 'genero', que pode assumir mais de um valor dentre opções como 'romance', 'tecnico', 'biografia', etc. Digamos que se queira obter todos os itens cujo atributo 'genero' possua entre seus valores as strings 'romance' e 'juvenil'. A consulta

*SELECT * FROM livros WHERE genero = 'romance' AND genero = 'juvenil'*

não irá retornar nenhum item, já que a condição pertence a um único predicado. A consulta procurará itens que possuam dois valores diferentes ao mesmo tempo no mesmo item, o que garantidamente não será satisfeito por nenhum item.
O operador INTERSECTION é usado nesse cenário da seguinte forma:

*SELECT * FROM livros WHERE genero = 'romance' INTERSECTION genero = 'juvenil'*

Aqui a consulta é tratada como duas consultas de predicados diferentes, retornando por fim apenas os itens presentes nos resultados de ambas as consultas.

A função EVERY() permite testar uma mesma condição para todos os valores de um atributo. Se quisermos, no cenário anterior, recuperar os itens que não possuem o valor 'tecnico' no atributo 'genero', podemos usar a função EVERY() da seguinte forma:

*SELECT * FROM livros WHERE EVERY(genero) != 'tecnico'*

Capítulo 3

Exemplo de aplicação

Neste capítulo, descreveremos a construção de um simples aplicativo web para gerenciamento de tarefas. Mostraremos como acessar a API do Amazon SimpleDB usando a linguagem python e os resultados obtidos.

3.1 Python API

3.1.1 The Boto API

Boto é um projeto que proporciona API's simples de manuseio dos web services da amazon para a linguagem python

3.1.2 Pricipais Classes

boto.sdb.connection

Proporciona a abstração da conexão com o amazon simple db com a sintaxe:

```
sdb.connection.SDBConnection(aws_access_key_id=None, aws_secret_access_key=None, is_secure=True,
port=None, proxy=None, proxy_ port=None, proxy_ user=None, proxy_ pass=None, debug=0,
https_ connection_ factory=None, region=None, path='/', converter=None, security_ token=None,
validate_ certs=True)
```

Mas é normalmente usado na forma:
`sdb.connection.SDBConnection('your_access_key', 'your_private_access_key', region='sua regiao de preferencia')`
A região é uma das listadas na documentaão

Dada uma conexão vocẽ pode executar as operaões descritas abaixo:

- batch_delete_attributes(domain_or_name[1], items[2])
- batch_put_attributes(domain_or_name, items, replace=True)
- create_domain(domain_name)
- delete_attributes(domain_or_name, item_name, attr_names=None, expected_value=None)
- delete_domain(domain_or_name)

- domain_metadata(domain_or_name)
- get_all_domains(max_domains=None, next_token=None)
- get_attributes(domain_or_name, item_name, attribute_names=None, consistent_read=False, item=None)
- get_domain(domain_name, validate=True)
- put_attributes(domain_or_name, item_name, attributes[2], replace=True, expected_value=None)
- select(domain_or_name, query=", next_token=None, consistent_read=False)

boto.sdb.domain

Essa classe fornea abstração dos dominios registrados na sua conta aws e permite a execuão de das mesmas instrucoes de manipulaão de dados presentes na classe boto.sdb.connection, a criaão desse objeto é possivel com a chamada:
`myconnection.get_domain('mydomain')`

[1] 'domain_or_name' deve ser uma string com o nome do dominio ou o proprio objeto de dominio, como descrito na proxima seão

[2] 'items' e 'attributes' deve ser um objeto dict-like

3.2 Descrição

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis,

fermentum vel, eleifend faucibus, vehicula eu, lacus.

3.3 Construção

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Capítulo 4

Conclusão

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Referências Bibliográficas

[1] CODD, E. F. *A Relational Model of Data for Large Shared Data Banks*, Communications of the ACM, 1970.

[2] SOUSA, F.R.C, *Gerenciamento de Dados em Nuvem*. 2010.

[3] Mocky Habeeb, *A Developer's Guide to Amazon SimpleDB*. Addison-Wesley, 1st edition, 2011.

[4] Verbete sobre história dos SGBDs na Wikipedia, `http://en.wikipedia.org/wiki/Database_management_system#History`

[5] Verbete sobre bancos de dados de navegação na Wikipedia `http://en.wikipedia.org/wiki/Navigational_database`

[6] Verbete sobre o SGBD MySQL na Wikipedia `http://en.wikipedia.org/wiki/MySQL`

[7] Verbete sobre o SGBD PostgreSQL na Wikipedia `http://en.wikipedia.org/wiki/PostgreSQL`

[8] Verbete sobre NoSQL na Wikipedia `http://en.wikipedia.org/wiki/NoSQL`

[9] Verbete da Wikipedia sobre computação em nuvem `http://en.wikipedia.org/wiki/Cloud_computing`

[10] Introdução ao NoSQL parte I `http://escalabilidade.com/2010/03/08/introducao-ao-nosql-parte-i/`

[11] Verbete da Wikipedia sobre o Amazon SimpleDB `http://en.wikipedia.org/wiki/Amazon_SimpleDB`

[12] Verbete da Wikipedia sobre o teorema CAP `http://en.wikipedia.org/wiki/CAP_theorem`

[13] Verbete da Wikipedia sobre Consistência Eventual `http://en.wikipedia.org/wiki/Eventual_consistency`