CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CURSO DE ENGENHARIA DE COMPUTAÇÃO

PERSISTÊNCIA POLIGLOTA

JOSÉ FRANCISCO CAMPOS LIMONGI

Orientador: Evandrino Barros
Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG

BELO HORIZONTE
JULHO DE 2014

JOSÉ FRANCISCO CAMPOS LIMONGI

PERSISTÊNCIA POLIGLOTA

Modelo canônico de trabalho monográfico acadêmico em conformidade com as normas ABNT apresentado à comunidade de usuários LaTeX.

Orientador: Evandrino Barros
Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CURSO DE ENGENHARIA DE COMPUTAÇÃO
BELO HORIZONTE
JULHO DE 2014

JOSÉ FRANCISCO CAMPOS LIMONGI

PERSISTÊNCIA POLIGLOTA

Modelo canônico de trabalho monográfico acadêmico em conformidade com as normas ABNT apresentado à comunidade de usuários LaTeX.

Trabalho aprovado. Belo Horizonte, 24 de novembro de 2014

Evandrino Barros
Orientador

Professor
Convidado 1

Professor
Convidado 2

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CURSO DE ENGENHARIA DE COMPUTAÇÃO
BELO HORIZONTE
JULHO DE 2014

Espaço reservado para dedicatória. Inserir seu texto aqui...

Agradecimentos

Inserir seu texto aqui... (esta página é opcional)

"Seja realista, exija o impossível." (Roland Castro)

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Lista de Algoritmos

SQL Structure Query Language 1

Redis Remote Dictionary Server 1

SGBD Sistema de Gerenciamento do Banco de Dados 5

UML Unified Model Language vi

UoD universo de discurso 4

ACID Atomicidade, Consistência, Isolamento e Durabilidade 10

XML *eXtensible Markup Language* 12

Sumário

1 Introdução

A necessidade de persistência de dados sempre esteve presente na computação. A medida que os sistemas evoluíram a complexidade da forma que os dados eram armazenados aumentou significativamente. Com isso, houve a necessidade da criação de um sistema computadorizado de manutenção de registros (DATE, 2004), o banco de dados.

O modelo relacional foi um dos primeiros gêneros de banco de dados, sua estrutura são tabelas de duas dimensões com linhas e colunas. Os dados armazenados são tipados podendo variar a quantidade de tipos de acordo com o banco utilizado. Para interagir com esse gênero é necessário realizar consultas com a linguagem Structure Query Language (SQL). Alguns exemplos de banco de dados relacional são MySQL [1], Oracle [2] e PostgreSQL [3].

Durante anos, o banco de dados relacional tem sido considerado a melhor opção para os problemas de escalabilidade, porém surgiram novas soluções com novas alternativas de estruturas, replicação simples, alta disponibilidade e novos métodos de consultas (REDMOND, 2012). Essas opções são conhecidas como NoSQL ou banco de dados não-relacional

Existem diversos gêneros de banco de dados não-relacional, entre eles chave-valor, orientado a documento, orientado a coluna e orientado a nó. Com o surgimento dessas novas soluções, o questionamento sobre qual banco de dados é melhor para resolver certo tipo de problema, vem à tona. A partir disso, o conhecimento e compreensão sobre os bancos de dados em geral se torna necessário para realizar uma boa escolha.

Entendendo que cada banco se destaca em determinados tipos de problema, é nítido perceber que sistemas que trabalham com mais de um banco de dados podem oferecer um melhor desempenho, dando origem à persistência poliglota.

Este trabalho consiste na comparação de dois sistemas, o primeiro que utilizará apenas um banco de dados, que será do gênero orientado a documento, e o segundo que utilizará dois bancos de dados, sendo um do gênero orientado a documento e um outro banco do gênero chave-valor. O segundo sistema por utilizar dois bancos de dados caracteriza a persistência poliglota, que é alvo desse trabalho. Os bancos escolhidos para fazer esses sistemas foram o MongoDB e o Remote Dictionary Server (Redis). O autor escolheu esses bancos de dados por ter experiência na linguagem *Ruby on Rails* que

[1] Sítio oficial <http://www.mysql.com>
[2] Sítio oficial <http://www.oracle.com/technetwork/oem/db-mgmt/db-mgmt-093445.html>
[3] Sítio oficial <http://www.postgresql.org/>

oferece excelentes bibliotecas para esses bancos. O intuito desse trabalho é comprovar que o uso da persistência poliglota melhora o desempenho da aplicação.

MongoDB é um banco de dados do gênero orientado a documento e foi desenhado para ser gigante. O próprio nome é uma derivação da palavra inglesa *humongous* que significa gigantesco. O diferencial desse gênero é a maneira que os registros são armazenados. Cada registro fica armazenado em um documento que é análogo à tupla no modelo relacional. O documento é composto por um identificador único e um conjunto de valores de tipos e estruturas aninhadas. Esse gênero é bem flexível, pois não tem catálogo, ou seja, não existe tabela e permite valores multivalorados. Ao criar a arquitetura do sistema temos que identificar se as entidades criadas são expressivas como um documento (REDMOND, 2012). Além disso, tem soluções para tratar concorrência e foi desenhado para trabalhar em *clusters*.A linguagem utilizada para fazer consulta no MongoDB é JavaScript. Nesse trabalho iremos utilizar o banco MongoDB nos dois sistemas que serão criados. Esse banco está sendo utilizado em grandes empresas, como Cisco, eBay, Codeacademy, Microsoft, Craiglist, The Guardian e outras, conforme sítio oficial do MongoDB [4].

O segundo banco que iremos utilizar se chama Redis do gênero chave-valor. Esse tipo de armazenamento é mais simples, como próprio nome indica, é armazenado um valor para determinada chave. O valor armazenado pode ter uma estrutura variável. Essa escolha foi feita devido ao cache que esse sistema realiza antes de efetivar a operação no disco. Esse cache tem um ganho muito alto em desempenho, porém poderá ocorrer perda de dados, caso ocorra uma falha de hardware (REDMOND, 2012). A forma de como estruturar esse banco é muito parecida com um tipo estruturado chamado *hash* que são implementadas em algumas linguagens de computação, como Java e Ruby. Esse banco será utilizado no segundo sistema a ser desenvolvido. O Redis está sendo utilizado no Twitter, Github, Craiglist e outros conforme referência do sítio oficial [5].

1.1 Motivação

A persistência poliglota é uma alternativa para melhorar o desempenho de uma aplicação. Utilizando-a conseguimos adaptar cada tipo de problema com um gênero de banco de dados.

Exitem duas variáveis opostas no ambiente de persistência de dados, consistência e disponibilidade. Quanto mais consistente um dado, menos disponível ele será e quanto mais disponível um dado menos consistente ele estará. Em aplicações é comum termos um conjunto de dados que deve ser sempre consistente e um outro conjunto de dados

[4] <http://www.mongodb.com/customers>
[5] <http://redis.io/topics/whos-using-redis>

que deve estar sempre disponível. Logo, para atender a esses conjuntos de dados devem ser utilizados dois banco de dados, um que garante disponibilidade e outro que garante consistência. Utilizando esse ambiente misto é esperado que haja um ganho de desempenho.

Atualmente poucos trabalhos apresentam uma comparação entre sistemas que utilizam apenas um banco, sistema monoglota, e sistemas que utilizam persistência poliglota.

2 Fundamentação Teórica

Para entendermos o porquê de utilizar mais de um banco de dados em uma mesma aplicação, temos que entender o que é um banco de dados, quais gêneros existem e para qual tipo de problema cada gênero se destaca.

2.1 Banco de Dados

Banco de dados é um sistema computadorizado de manutenção de registros, análogo à um armário de arquivamento eletrônico. Podemos entendê-lo como um repositório para manter a coleção de arquivos de dados computadorizados (DATE, 2004). Elmasri (2005) define banco de dados como uma coleção de dados relacionados e que dados são fatos com um significado implícito. Porém, a definição de Elmasri (2005) é muito abrangente, logo ele aponta três propriedades implícitas para restringir a definição de banco de dados.

A primeira propriedade é que um banco de dados deve representar alguns aspectos do mundo real, chamado de *universo de discurso (UoD)*. As alterações que ocorrem nesse universo são refletidas em um banco de dados. A segunda propriedade define que o banco de dados é uma coleção lógica e coerente de dados com algum significado inerente, ou seja, uma coleção de dados randômicos não pode ser considerado um banco de dados. A terceira propriedade afirma que banco de dados é projetado, construído e povoado com dados, atendendo a uma proposta específica. Além disso, possui um grupo de usuários definido e algumas aplicações preconcebidas, de acordo com o interesse desse grupo.

Os bancos de dados têm contribuído para o aumento do uso do computador (ELMASRI, 2005) e podemos afirmar que eles apresentam um papel crucial em quase todas as áreas em que os computadores são utilizados. Devido a essa importância o estudo sobre banco de dados é extremamente necessário para os profissionais da computação.

Antes da existência dos bancos de dados, a aplicação devia gerenciar e processar arquivos para manter os dados persistidos. Para justificar o uso de banco de dados, Elmasri (2005) cita quatro características: natureza autodescritiva, abstração de dados, suporte para as múltiplas visões de dados e processamento de transações de multiusuários.

A primeira característica, natureza autodescritiva do banco de dados, apresenta o catálogo do banco de dados como uma grande vantagem sobre o processamento

tradicional dos arquivos. Pois, o catálogo identifica as estruturas dos arquivos, formato e tipo de dados. Logo, não é necessário conhecer a aplicação para trabalhar com os dados. Já o processamento tradicional dos arquivos, mantém essas definições de estrutura na própria aplicação (ELMASRI, 2005).

Em relação à característica de abstração de dados, Elmasri (2005) afirma que não é feita no processamento tradicional de arquivos, pois é a aplicação que define a estrutura dos dados. Suponha que tenhamos diversos programas utilizando o mesmo arquivo para armazenar uma coleção de dados. Se um desses programas precisar de acrescentar algum campo novo, todos os outros programas que acessam esse arquivo, devem modificados para contemplar o novo campo adicionado. Já quando utilizamos banco de dados, a alteração da estrutura dos dados pode não influenciar no funcionamento dos outros programas.

Em relação à característica suporte para múltiplas visões dos dados, Elmasri (2005) diz que quando é utilizado o banco de dados, é possível ter diferentes visões sobre os dados, fazendo o cruzamento das tabelas. Com a abordagem de processamento de arquivo tradicional isso não é usual.

A última característica citada por Elmasri (2005) é o processamento de transação multiusuários, essa característica é essencial para que várias aplicações possam acessar e alterar os dados a partir de usuários diferentes e simultâneos. Porém, o Sistema de Gerenciamento do Banco de Dados (SGBD) deve ter implementado um controle de concorrência para garantir a atomicidade das transações e a consistência dos mesmos.

Elmasri (2005) não cita a existência de bancos de dados sem catálogo, chamados de *schemaless*. Apesar de não ter a declaração do tipo de estruturas de dados contidas no banco, os bancos de dados *schemaless* fazem a abstração dos dados da mesma maneira que os bancos de dados tradicionais, têm suporte para múltipla visões e multiusuários.

Como revelado acima, a utilização do banco de dados facilita o desenvolvimento das aplicações, faz a abstração entre aplicação e dados e, além disso, faz o controle de concorrência. Após verificarmos que o uso de banco de dados é imprescindível, nos deparamos com uma outra dificuldade, qual banco de dados utilizar. Os bancos de dados, chamados de NoSQL, chama a atenção da comunidade científica, depois da publicação de dois artigos BigTable (CHANG et al., 2006) e Dynamo (DECANDIA DENIZ HASTORUN; VOGELS, 2007)

2.2 Gêneros de Banco de dados

Durante anos, o banco de dados relacional tem sido considerado a melhor opção para a maioria dos problemas de pequeno ou grande volume de dados. O aumento

do volume de dados fez com que os especialistas buscassem novas soluções, que permitissem o armazenamento paralelo dos dados, pois o modelo relacional não foi desenhado para funcionar em *clusters*. Logo, os bancos de dados NoSQL se destacaram por funcionar bem no ambiente paralelo e ter um melhor desempenho com grande volume de dados (SADALAGE, 2013).

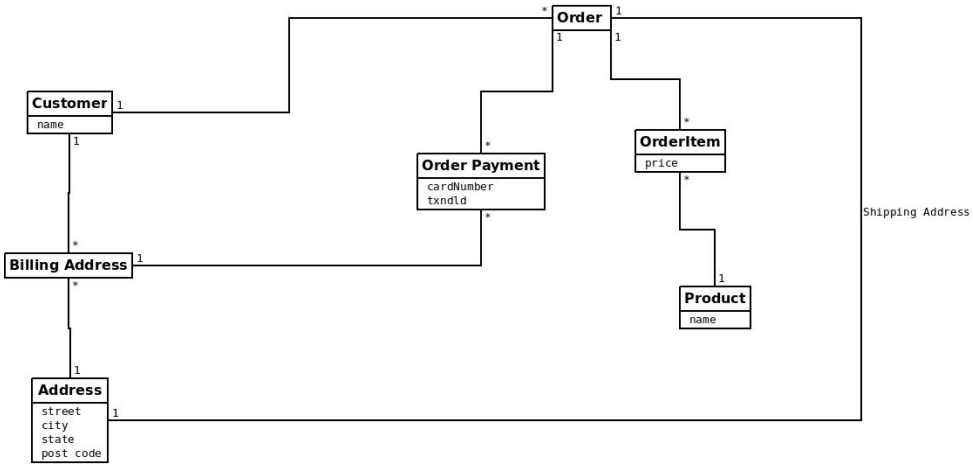
Os bancos de dados NoSQL têm as seguintes características: não usa o modelo relacional, foram desenhados para funcionar em *clusters*, são *open source* e não tem catálagos (*schemaless*) (SADALAGE, 2013). Carlo Strozzi foi o primeiro a utilizar o nome NoSQL, mas não no sentido que a palavra tem hoje. Strozzi denominou um banco de dados relacional, *open source* de NoSQL, pois não usava SQL como linguagem de consulta. Uma conferência, realizada em São Francisco nos Estados Unidos em Junho de 2009, foi responsável por denominar esses bancos de dados de NoSQL. Johan Oskarsson que organizou essa conferência escolheu esse nome, porque era uma boa hashtag no Twitter: pequeno, memorável e tinha poucos resultados no Google. Isso facilitaria os interessados a encontrar a conferência. Apesar de o termo não significar explicitamente o que são esses bancos de dados atendeu bem a intenção de Oskarsson (SADALAGE, 2013).

2.2.1 Banco de dados Relacional

O modelo relacional é o mais comum atualmente. Esse gênero armazena os dados em tabelas de duas dimensões em linhas e colunas. A interação com esse banco é feito por um SGBD que utiliza o SQL como linguagem de consulta de dados. Os dados armazenados são valores tipados e podem ser numéricos, texto, data e outros tipos, que são configurados e forçados pelo sistema. É possível fazer relações entre tabelas e cruzar informações de maneira simples. MySQL, Oracle e PostgreSQL são alguns exemplos de banco de dados relacional (REDMOND, 2012). A representação do modelo relacional, pode ser feita com o diagrama UML. Um sistema de *e-commerce* poderia ser desenhado conforme o diagrama da Figura 5 e os valores ficariam dispostos conforme Figura 6.

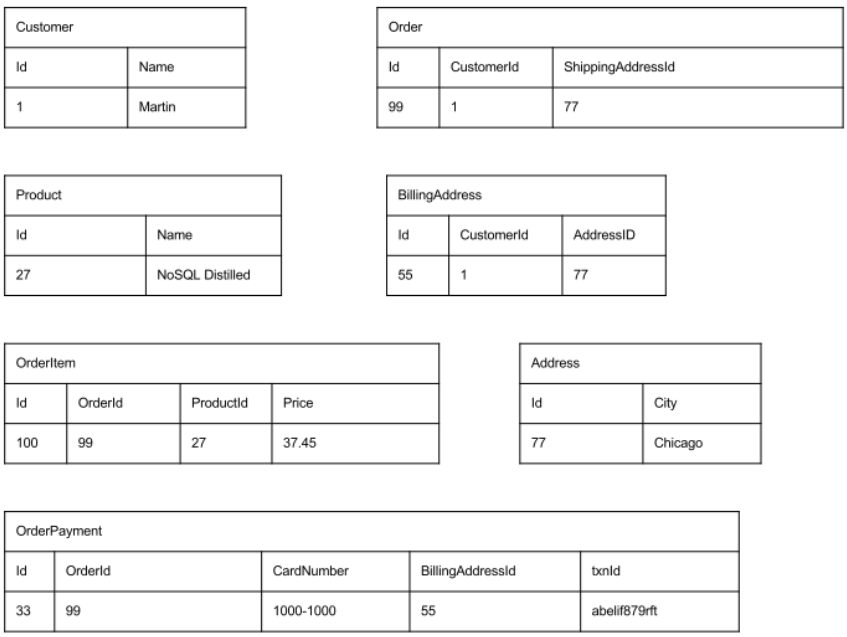
O gênero relacional funciona muito bem para diversas aplicações, pois é bem flexível em relação às consultas, permite concorrência, transações e pode ser integrado com várias aplicações. Porém, há uma desvantagem que causa frustação em muitos desenvolvedores, chamada de Impedância de Correspondência ou *Impedance Mismatch*. Isso ocorre, pois nem sempre o tipo do campo no banco de dados irá corresponder com o tipo esperado da linguagem utilizada, então é necessário criar uma forma de associação entre o tipo da variável da linguagem com o tipo do valor da tabela. Outra desvantagem é que esse gênero não aceita valores multivalorados, distanciando a aplicação ainda mais do modelo relacional.

Figura 1 – Exemplo de um diagrama UML para o modelo relacional



Fonte: (SADALAGE, 2013)

Figura 2 – Exemplo da disposição dos dados no modelo relacional



Fonte: (SADALAGE, 2013)

2.2.2 Banco de dados não-relacional

Os bancos de dados NoSQL, foram construídos para suprir a necessidade de se trabalhar com grande quantidade de dados e em *clusters*. NoSQL abrange diversos gêneros de banco de dados, entre eles o orientado a documento, chave-valor, orientado a coluna e orientado a nó.

Todos esses gêneros não possuem catálogo, ou seja, não é definido previamente qual estrutura os dados serão armazenados. Isso permite uma flexibilidade no sistema, pois a estrutura dos dados pode ser alterada facilmente. É possível adicionar um novo campo, sem ter que preocupar com qual valor colocar para a base legada, pois os objetos de uma mesma coleção podem ter diferentes campos. Da mesma maneira, para remover um campo, basta parar de armazená-lo, pois os registros antigos, que tinham esse campo continuarão com eles, e os objetos novos não irão armazenar esse campo, que já não faz parte da aplicação (SADALAGE, 2013).

Com isso é possível trabalhar com dados não uniformes: são dados que para cada registro há um conjunto diferente de atributos. Para que o banco de dados relacional lide com um objeto dessa natureza, é necessário uma tabela com os campos de todos os objetos e consequentemente, isso traria uma quantidade grande de campos vazios.

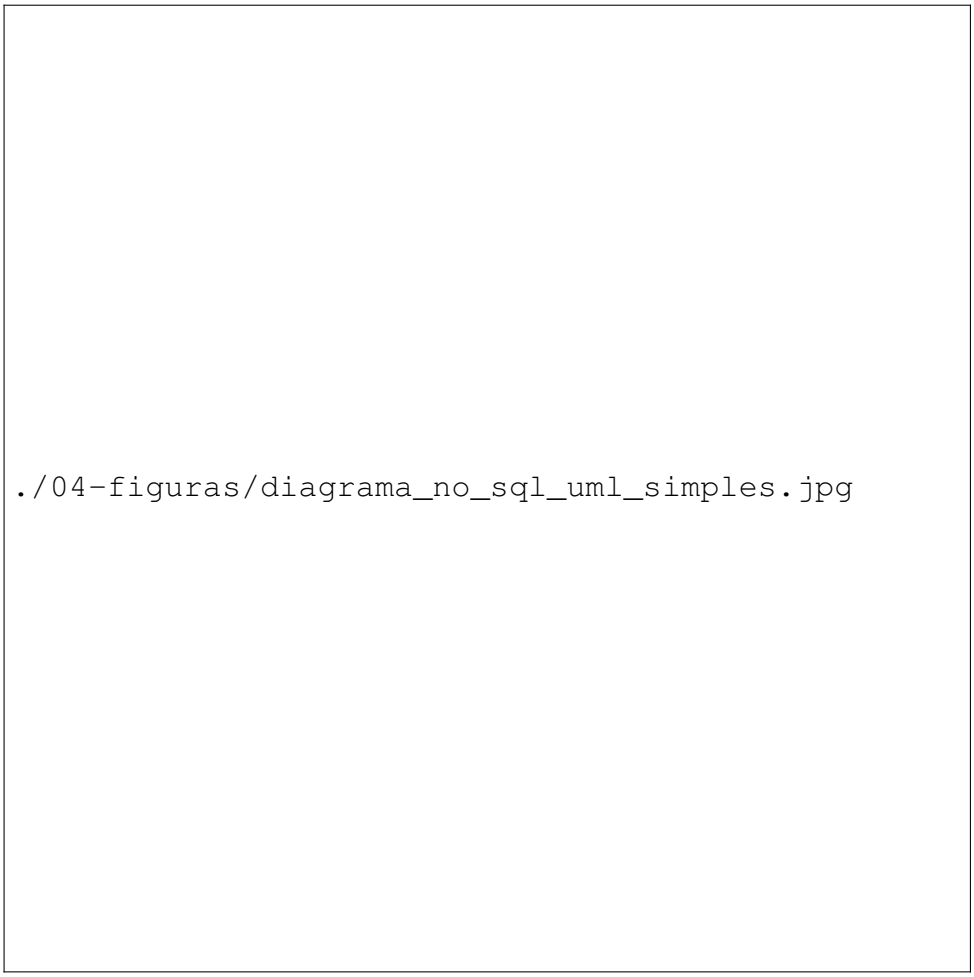
Basicamente os banco de dados NoSQL deslocam a definição do esquema para a aplicação que acessa o banco. Isso pode se tornar problemático quando há muitas aplicações acessando o mesmo banco, mas existem soluções para resolver isso, como encapsular toda a interação com o banco de dados, fazendo esse funcionar como um *web service* (SADALAGE, 2013).

Além dessa flexibilidade, a maioria dos gêneros NoSQL permitem estruturar dados multivalorados [1]. Esses dados são modeladas como uma agregação, que é uma coleção de objetos relacionados que desejamos tratar como um único objeto (**??**). A agregação facilita a manipulação e a consistência desses dados, pois é tratado com uma unidade, ou seja, é lido e escrito, sempre, todo o conjunto de dados. Como, geralmente, ao utilizar esse relacionamento buscamos operações atomicas, essa abordagem se encaixa perfeitamente com a aplicação (SADALAGE, 2013). A modelagem em UML do mesmo exemplo anterior utilizando agregação, ficaria como a **??** e a disposição de dados ficaria como a **??** [2].

Nesse modelo, utilizamos duas agregações principais: cliente e pedido. O cliente contém uma lista de endereços, o pedido contém uma lista de itens de pedido, endereços e pagamentos. E por fim, o pagamento contém uma lista de endereços. Endereço aparece três vezes, mas ao invés de utilizar uma referência com um identificador, como no modelo relacional, o valor é copiado. Essa replicação de dados se adequa ao problema, pois nesse caso não queremos que o endereço em pagamento, ou em pedido seja alterado caso o cliente atualize a própria lista. Utilizando o modelo relacional temos duas maneiras de lidar com o problema. A primeira, tratar para que não seja permitido alterar o endereço, após desse ser vinculado à pedido ou pagamento. A segunda e mais usada, seria duplicar o endereço e associar à pedido ou à pagamento.

[1] O banco de dados orientado a nó, é um dos gêneros que não implementam agregação.
[2] É usual a utilização de JSON para mostrar os dados em NoSQL (SADALAGE, 2013)

Figura 3 – Exemplo de um diagrama UML utilizando agregação



Fonte: (SADALAGE, 2013)

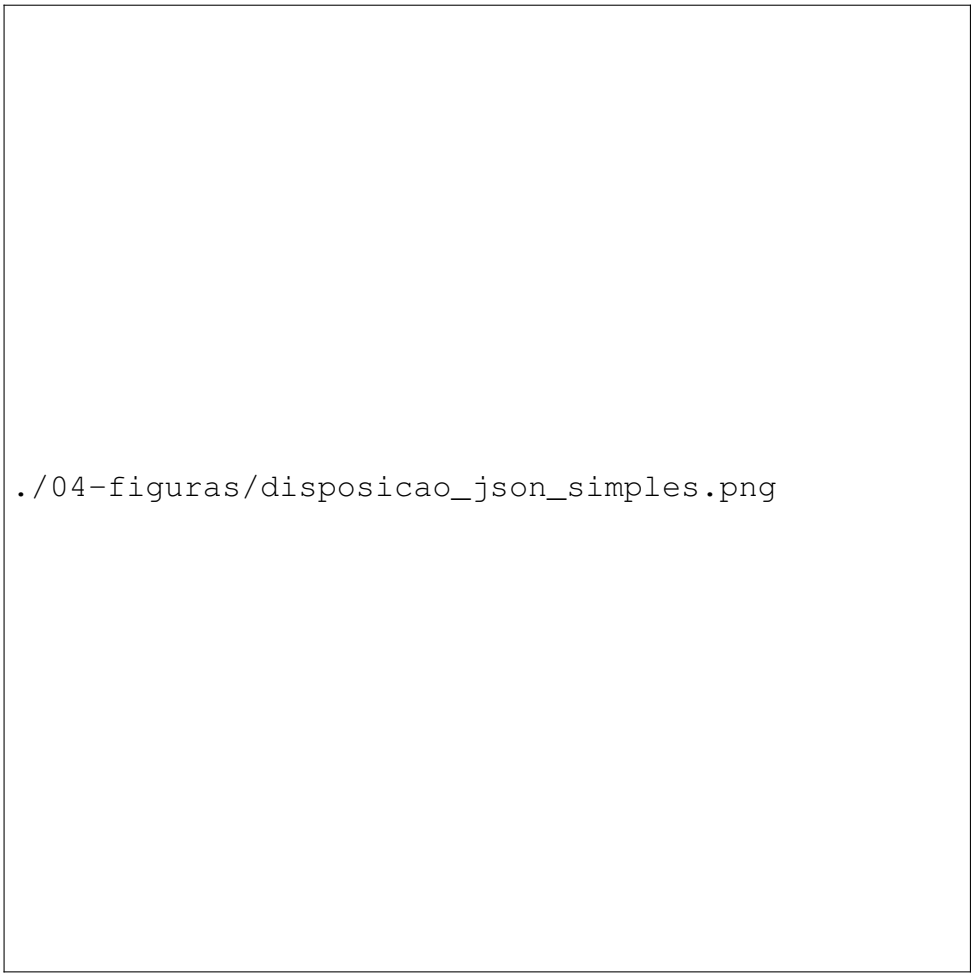
A ligação entre pedido e cliente é uma relação que ocorre da mesma maneira que o modelo relacional. O pedido mantém um identificador do cliente, mas também tem como utilizar a agregação para essa relação. O modelo em UML ficaria como a **??** e os dados ficariam dispostos como a **??**.

Como na maioria dos problemas de modelamento não existe a melhor solução, mas sim uma que se adequa melhor para um certo tipo de problema, depende completamente da natureza da aplicação. Se for interessante para a aplicação listar o histórico dos pedidos, o segundo modelo, utilizando NoSQL, não é o ideal, pois será necessário entrar em cada usuário para ler os pedidos. Já no primeiro modelo, utilizando NoSQL, essa consulta do histótico se torna trivial (SADALAGE, 2013).

Outra razão para utilizar agregação é que isso facilita o uso do banco de dados em *clusters*, pois isso informa ao sistema, quais *bits* dos dados devem ser manipulados juntos e se devem estar no mesmo nó do {cluster.

O modelo relacional por não tratar o conceito de agregação é chamado de

Figura 4 – Exemplo da disposição dos dados utilizando agregação



Fonte: (SADALAGE, 2013)

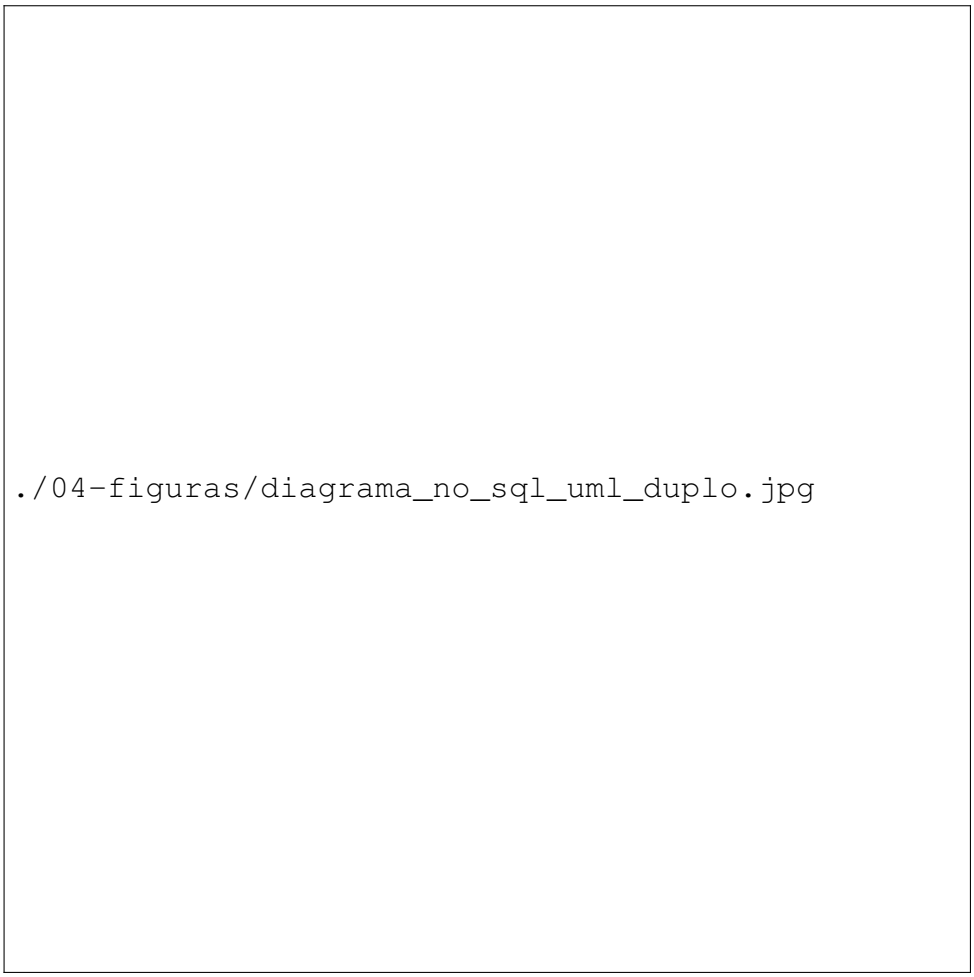
aggregate-ignorant. Apesar dessas vantagens apresentadas sobre agregação, o modelo relacional tem suas vantanges por não tratar. A principal é que ao utilizar o modelo relacional, podemos analisar os dados em diversas perspectivas, já no banco de dados NoSQL algumas consultas podem não ser triviais ou até mesmo mais lentas.

Outra desvantagem do NoSQL, é não implementar transações Atomicidade, Consistência, Isolamento e Durabilidade (ACID) sobre várias agregações. Esses gêneros de banco de dados conseguem implementar essas transações apenas sobre uma agregação como dito anteriormente. Caso seja necessário o controle dessas transações, deverá ser feito no código da aplicação.

O objeto desse trabalho são duas aplicações que irão utilizar dois gêneros de banco de dados, orientado a documento e chave-valor.

O gênero chave-valor funciona como uma simples tabela *hash* que dado uma chave única encontrará um valor. O sistema desse banco não tem conhecimento algum sobre esse valor, logo poderá ser um texto, um objeto multivalorado, um valor binário,

Figura 5 – Exemplo de um diagrama UML para utilizando agregação entre cliente e pedido

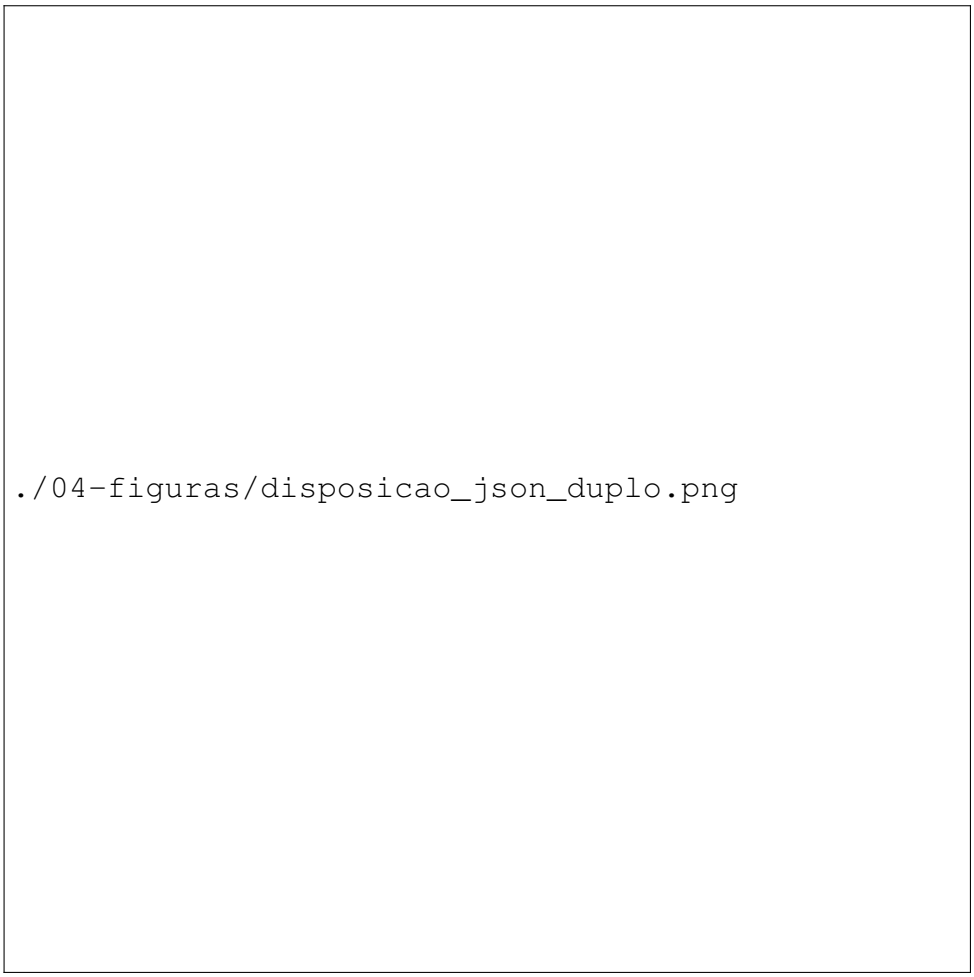


Fonte: (SADALAGE, 2013)

qualquer tipo de informação, apenas o tamanho é limitado, dependendo do banco. Alguns exemplos de bancos desse gênero são: Redis[3] e Riak[4] A consistência utilizando esse gênero existe apenas para operação realizada com uma chave. Então não temos consistência para um grupo de chaves, isso pode ser implementado, mas é muito custoso (SADALAGE, 2013). Em relação a consistência, no ambiente paralelo, cada banco implementa de uma maneira. O Riak, por exemplo, utiliza o chamado *Quorums*, que é uma abordagem de consistência que elege o valor mais atual fazendo uma consulta entre os nós. Aquele valor, que tiver na maioria dos nós, será eleito como mais atualizado. A consulta sobre os valores é feita apenas buscando a chave, qualquer outro filtro necessário, deverá ser feito pela aplicação, após ler todo o valor retornado. Também é possível programar o tempo de expiração da chave. Essa maneira que o banco chave-valor trabalha o faz com que ele se adeque bem para manter dados da

[3] Sítio oficial do Redis <http://redis.io/>
[4] Sítio oficial do Riak <http://basho.com/riak/>

Figura 6 – Exemplo da disposição dos dados no NoSQL, utilizando agregação entre cliente e pedido



Fonte: (SADALAGE, 2013)

sessão de um usuário, carrinhos de compra de um *e-commerce*, perfil de usuários e outros. Não é indicado para armazenar valores que precisam ser filtrados.

O gênero orientado a documento se trabalha com o conceito de documento. Documento pode ser um **XML!** (**XML!**), JSON, BSON ou outros formatos. Esses documentos são armazenados em coleções. Documentos da mesma coleção são semelhantes, mas podem não ser idênticos. Se fizermos uma analogia, a coleção seria a tabela no modelo relacional e o documento seria a tupla no modelo relacional. Exemplo desse gênero são MongoDB [5] e CouchDB [6](SADALAGE, 2013). Existem dois tipos de relacionamento, um que funciona semelhante ao modelo relacional que é referenciando um documento de uma coleção pelo identificador, análogo a chave estrangeira no modelo relacional . A outra maneira é embutir documentos dentro de um documento, ou seja, agregação. Consistência existe apenas para os objetos da agregação como relatado anteriormente.

[5] Sítio oficial do MongoDB <http://www.mongodb.com/>
[6] Sítio oficial do CouchDB <http://couchdb.apache.org/>

Quando distribuído, o MongoDB pode trabalhar com um série de políticas que são configuráveis. Esse gênero não implementa transações(operações de inserir, atualizar ou deletar com a opção de fazer o *commit* ou *rollback*). O MongoDB permite que seja feito consultas, a linguagem utilizada é o JavaScript. Consultas mais complexas que utilizam os documentos embutidos em outros documentos são realizadas com *MapReduce* (SADALAGE, 2013).

2.2.3 Persistência Poliglota

Diferentes bancos de dados foram desenhados para resolver diferentes problemas (SADALAGE, 2013). Conseguir identificar diferentes problemas dentro de uma aplicação pode ser um indicativo que mais de um banco de dados deve ser utilizado. Da mesma maneira que diferentes paradigmas de linguagem de programação são utilizados em um desenvolvimento *web*, diferentes gêneros de banco de dados podem ser usado em uma mesma aplicação (**??**)

Referências

CHANG, F.; DEAN, J.; GHEMAWAT, S.; HSIEH, W. C.; WALLACH, D. A.; BURROWS, M.; CHANDRA, T.; FIKES, A.; GRUBER; E., R. Bigtable: A distributed storage system for structured data. In: **Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7**. Berkeley, CA, USA: USENIX Association, 2006. (OSDI '06), p. 15–15. Disponível em: <http://dl.acm.org/citation.cfm?id=1267308.1267323>. Citado na página 5.

DATE, C. **Introdução a Sistema de Bancos de Dados**. 8º. ed. [S.l.]: Elsevier Editora LTDA, 2004. Citado 2 vezes nas páginas 1 e 4.

DECANDIA DENIZ HASTORUN, G. K. A. L. A. P. S. S. P. V. G.; VOGELS, W. Dynamo: Amazon's highly available key-value store. **ACM Symposium on Operating Systems Principles**, v. 21st, 2007. Citado na página 5.

ELMASRI, S. B. N. R. **Sistema de Banco de Dados**. 4º. ed. [S.l.]: Pearson Education do Brasil Ltda, 2005. 7,8,9 p. Citado 2 vezes nas páginas 4 e 5.

REDMOND, J. R. W. E. **Seven Databases in Seven Weeks**. 1º. ed. [S.l.]: Pragmatic Programers, LLC, 2012. Citado 3 vezes nas páginas 1, 2 e 6.

SADALAGE, M. F. P. J. **NoSql, A Brief Guide to the Emerging World of Polyglot Persistence**. 8º. ed. [S.l.]: Pearson Education, Inc, 2013. Citado 8 vezes nas páginas 6, 7, 8, 9, 10, 11, 12 e 13.

Apêndices

APÊNDICE A – Nome do Apêndice

Inserir seu texto aqui...

APÊNDICE B – Nome do Apêndice

Inserir seu texto aqui...

Anexos

ANEXO A – Nome do Anexo

Inserir seu texto aqui...

ANEXO B – Nome do Anexo

Inserir seu texto aqui...